VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SEGMENTACE STRÁNEK VE WEBOVÉM PROHLÍŽEČI

PAGE SEGMENTATION IN A WEB BROWSER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. TOMÁŠ ZUBRIK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Zubrik Tomáš, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Informační systémy a databáze
Název: **Segmentace stránky ve webovém prohlížeči**
Page Segmentation in a Web Browser
Kategorie: Web
Zadání:

1. Prostudujte současné přístupy k segmentaci webových stránek. Zaměřte se na vizuálně orientované metody jako např. VIPS a BCS.
2. Seznamte se s technologiemi pro implementaci aplikací v JavaScriptu na klientské i serverové straně.
3. Navrhněte architekturu aplikace v JavaScriptu, která implementuje zvolenou metodu segmentace stránek. Svá návrhová rozhodnutí konzultujte s vedoucím práce.
4. Implementujte navrženou aplikaci pomocí vhodných technologií.
5. Proveďte vyhodnocení funkčností výsledné aplikace na vhodné množině webových dokumentů.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Zeleny, J.; Burget, R.; Zendulka, J.: Box clustering segmentation: A new method for vision-based web page preprocessing. Information Processing & Management. vol. 53, no. 3. 2017: pp. 735 - 750. ISSN 0306-4573.
- Cai, D.; Yu, S.; Wen, J.-R.; et al.: VIPS: a Vision-based Page Segmentation Algorithm. Microsoft Research. 2003.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 23. října 2020

Abstrakt

Táto práca sa zaoberá segmentáciou webových stránok vo webovom prehliadači. V rámci práce bola vytvorená implementácia metódy Box Clustering Segmentation (BCS) v jazyku JavaScript s využitím automatizovaného prehliadača. Samotná implementácia pozostáva z dvoch hlavných krokov, ktorými sú extrakcia boxov (listových uzlov DOM) z kontextu prehliadača a ich následné zhlukovanie na základe modelu podobnosti definovanom podľa BCS. Výsledkom práce je funkčná implementácia metódy BCS použiteľná na segmentáciu stránok. Vyhodnotenie funkčnosti a presnosti implementácie prebehlo na základe porovnania s referenčnou implementáciou vytvorenou v jazyku Java.

Abstract

This thesis deals with the web page segmentation in a web browser. The implementation of Box Clustering Segmentation (BCS) method in JavaScript using an automated browser was created. The actual implementation consists of two main steps, which are the box extraction (leaf DOM nodes) from the browser context and their subsequent clustering based on the similarity model defined in BCS. Main result of this thesis is a functional implementation of BCS method usable for web page segmentation. The evaluation of the functionality and accuracy of the implementation is based on a comparison with a reference implementation created in Java.

Kľúčové slová

segmentácia webových stránok, algoritmus Box Clustering Segmentation, BCS, zhlukovanie, model podobnosti, automatizácia prehliadača, Playwright

Keywords

web page segmentation, Box Clustering Segmentation algorithm, BCS, clustering, similarity model, browser automation, Playwright

Citácia

ZUBRIK, Tomáš. *Segmentace stránek ve webovém prohlížeči.* Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Segmentace stránek ve webovém prohlížeči

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Radka Burgeta, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Tomáš Zubrik
23. mája 2021

Poďakovanie

Týmto by som veľmi rád poďakoval Ing. Radkovi Burgetovi, Ph.D. za poskytnutý čas, cenné rady a odbornú pomoc pri riešení tejto diplomovej práce.

Obsah

Kapitola 1

Úvod

V dnešnej dobe sa pre mnohých z nás stal internet každodennou súčasťou nášho života. Je to najmä z dôvodu, že predstavuje primárny a v podstate nekonečný zdroj rýchlo dostupných informácií. Základnými nosičmi informácií na internete sú webové stránky, ktoré sa navzájom odlišujú svojou funkciou a zameraním. Neustály vývoj a narastajúci počet webových stránok súvisí so zvyšujúcimi sa nárokmi na nástroje, ktorých hlavným cieľom je automatizácia pri spracovaní webových dokumentov. Väčšina týchto nástrojov je zameraná práve na extrakciu informácií, získavanie znalostí a optimalizácie vo vyhľadávaní.

Aj keď sa webové stránky javia ako atomické nosiče informácií, obsahujú okrem užitočného informatívneho obsahu aj časti so zanedbateľnou informačnou hodnotou. Práve preto je väčšina webových stránok logicky rozdelená na menšie sémanticky alebo vizuálne konzistentné časti, tzv. bloky. Jednotlivé bloky s odlišným významom síce môžu byť jednoducho identifikovateľné používateľmi, to ale neplatí pre programy. Cieľom je tento proces identifikácie a následného označovania konzistentných blokov zautomatizovať, čomu odpovedá proces segmentácie webových stránok.

Segmentácia stránok je predovšetkým vnímaná ako základný krok predspracovania pri získavaní znalostí z webových dokumentov, keďže v tejto oblasti nachádza najširšie uplatnenie. Proces segmentácie pozostáva z vytvorenia štruktúry odpovedajúcej vstupnému dokumentu a následného výberu tých častí stránky, ktoré majú potenciál obsahovať relevantné informácie. V priebehu dlhoročného výskumu segmentácie vzniklo množstvo segmentačných metód, ktoré sa líšia najmä použitým prístupom, ale aj výslednou presnosťou a rýchlosťou segmentácie. Vzhľadom na to, že spôsob vytvárania webových stránok sa neustále vyvíja a mení, množstvo skôr navrhnutých metód je na moderných webových stránkach nepoužiteľných. Pri návrhu novej segmentačnej metódy by sa mala brať do úvahy nezávislosť na zdrojovom kóde, implementačných detailoch a spôsobe písania stránok.

Jednou zo segmentačných metód založených na zhlukovaní elementov na základe ich vizuálnych vlastností je *Box Clustering Segmentation* (BCS). Existuje viacero implementácií [38, 22] v jazyku *Java*, ktorých hlavnou nevýhodou je čiastočná závislosť na experimentálnom vykresľovacom jadre *CSSBox*. Hlavným cieľom tejto práce je implementácia metódy BCS v jazyku *JavaScript* s využitím automatizovaného prehliadača. Segmentácia stránky vo webovom prehliadači má význam predovšetkým z dôvodu automatického načítania a vykreslenia stránky integrovaným vykresľovacím jadrom, ktoré je súčasťou reálneho prehliadača. V prehliadači je tým pádom sprostredkovaný vykresľovací strom (*rendering tree*) s prepočítanými kaskádovými štýlmi jednotlivých elementov, z ktorého je možné získať relevantné informácie o ich vizuálnych a štrukturálnych vlastnostiach. Na základe získaných informácií o pozícii na stránke a reprezentatívnej farbe, je možné elementy zoskupovať do zhlukov.

V nasledujúcej kapitole *Segmentácia webových stránok* je všeobecne popísaný proces segmentácie a spôsoby jej využitia v praxi. V rámci kapitoly sú ďalej popísané rôzne segmentačné prístupy a ich rozdelenia, so zameraním na vizuálne orientované (*vision-based*) metódy. Podrobnejšie je opísaný algoritmus VIPS [7], ktorý predstavuje jednu z najznámejších segmentačných *vision-based* metód.

V rámci kapitoly 3 je detailne vysvetlená metóda BCS, ktorej implementáciou sa táto práca zaoberá. V kapitole je ďalej popísaný základný princíp metódy, jej vlastnosti a výhody. Zároveň sú tu uvedené aj jednotlivé kroky algoritmu a všetky dôležité definície, ktoré sú nevyhnutné pre proces zhlukovania.

Kapitola 4 pojednáva o technológiách pre implementáciu aplikácií v jazyku *JavaScript*. Súčasťou kapitoly je aj porovnanie dostupných frameworkov na automatizáciu prehliadača z prostredia *Node.js*. Kapitola 5 – *Návrh riešenia*, obsahuje popis problémov a ich navrhnutých riešení v súvislosti s automatizáciou prehliadača a jednotlivými krokmi algoritmu BCS. Implementácia metódy BCS je predstavená v kapitole 6. Kapitola obsahuje popis a spôsob implementácie a programovej realizácie jednotlivých definícií a riešených problémov uvedených v návrhu.

V kapitole 7 je v prvom rade popísaná dátová sada webových stránok použitá na vyhodnotenie, nasledovaná popisom metrík, na základe ktorých bude overená funkčnosť vytvorenej implementácie. Najdôležitejšou časťou kapitoly je porovnanie vytvorenej implementácie s referenčnou implementáciou, ktorá je súčasťou nástroja *FitLayout*. Zhrnutie dosiahnutých výsledkov a zhodnotenie miery splnenia vytýčených cieľov sa nachádza v kapitole 8 – *Záver*.

Kapitola 2

Segmentácia webových stránok

Segmentácia webových stránok je proces, ktorým sa webová stránka rozdeľuje na niekoľko blokov, ktoré sú vizuálne alebo významovo konzistentné. Proces segmentácie je zvyčajne rozdelený do dvoch hlavných krokov: detekcia a označovanie súvislých blokov, a ich následná klasifikácia do logických funkcií, ktorú na webovej stránke zastávajú [34].

Aj keď sa z technologického hľadiska webové stránky považujú za atomické nosiče informácií, obsahujú okrem užitočného informatívneho obsahu aj časti so zanedbateľnou informačnou hodnotou. Práve preto je väčšina webových stránok logicky rozdelená na menšie významové časti, ktoré nazývame sémantické bloky. Významovo oddelené bloky vytvárajú logickú štruktúru webovej stránky, vďaka ktorej sa používateľ dokáže na stránke jednoduchšie orientovať a rozlišovať medzi relevantným a menej užitočným obsahom. Na bežnej webovej stránke môžeme okrem hlavného obsahu nájsť rôzne ďalšie bloky, ktoré sa líšia svojou funkciou. Typicky sa jedná o navigačný panel umiestnený v hornej časti stránky, pomocou ktorého sa používateľ naviguje po stránke prostredníctvom odkazov. Ďalšími často sa vyskytujúcimi časťami sú napríklad záhlavie (*header*) a zápätie (*footer*).

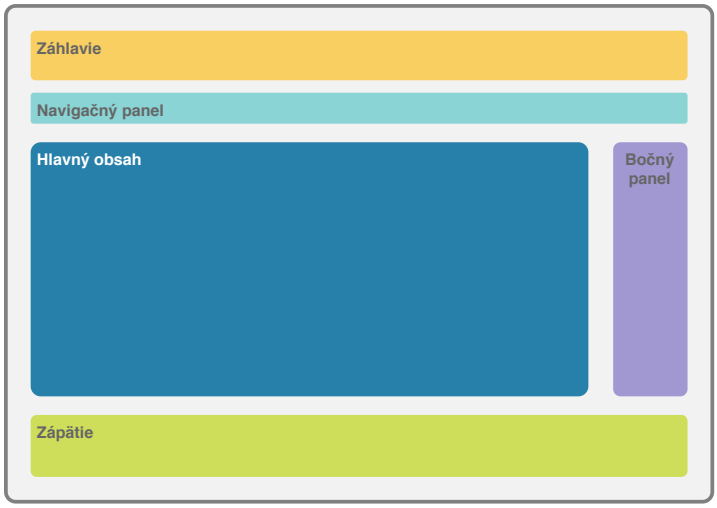
Vo všeobecnosti sa segmentácia neobmedzuje na sémantické bloky identifikovateľné používateľom, pričom jednotlivé segmentačné metódy dokážu rozpoznať bloky na rôznej úrovni podrobnosti. Aj na webových stránkach s úplne novou alebo nezvyčajnou štruktúrou a dizajnom, dokážu používatelia dobre rozlišovať rôzne segmenty. Cieľom segmentačných algoritmov je tento proces zautomatizovať a jednotlivé bloky rozpoznávať, prevažne za účelom extrakcie informácií.

Príklad rozdelenia webovej stránky na významové bloky reprezentuje štruktúra znázornená na obrázku 2.1, ktorá predstavuje základné poňatie segmentácie.

Medzi jednotlivými elementmi a sémantickými blokmi na stránke existujú vzťahy, na základe ktorých ich môžeme analyzovať a spracovávať. Schopnosť správne analyzovať štruktúru a obsah webovej stránky môže byť veľmi užitočná aj vo vyhľadávaní, pri ktorom sa webové vyhľadávače snažia nájsť stránky obsahujúce zadaný reťazec v hlavnom obsahu. Ďalšie možnosti využitia segmentácie sú popísané v kapitole 2.2.

2.1 Webové stránky a ich spracovanie

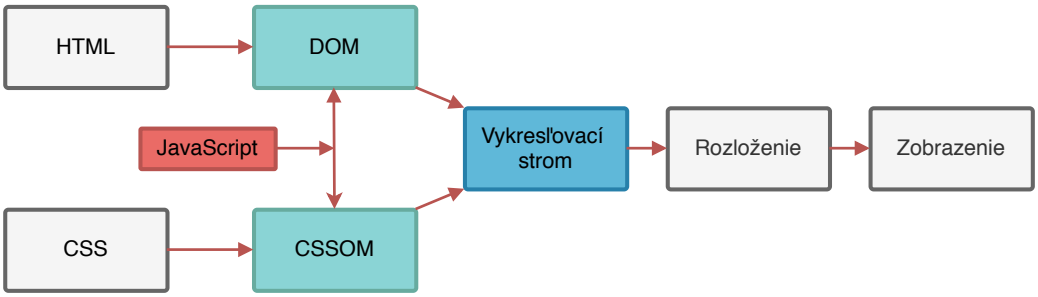
Na internete sa vyskytuje nespočetné množstvo webových stránok, ktoré sa odlišujú svojimi vlastnosťami, účelom aj kvalitou. V začiatkoch vývoja webových stránok vznikali prevažne jednoduché statické webové stránky, pri ktorých webový server na základe požiadavky od klienta odoslal odpoveď bez nutnosti ďalšieho spracovania. Odpoveďou webového serveru



Obr. 2.1: Znázornenie rôznych sémantických blokov na webovej stránke, ktoré sa líšia svojou funkciou a významom.

bola požadovaná webová stránka, ktorá bola následne zobrazená v prehliadači. Narozdiel od statických webových stránok sa moderné webové stránky vyznačujú dynamickosťou, komplexnosťou a širokým spektrom zameraní, čo taktiež ovplyvnilo výskum v oblasti segmentácie.

Webové stránky, s ktorými môže užívateľ interaktívne pracovať môžeme nazývať webové aplikácie. Na ich vývoj sa používajú webové technológie, ktorými sú JavaScript, hypertextový značkovací jazyk HTML a káskádové štýly CSS. Jednotlivé HTML dokumenty obohatené o kaskádové štýly sprostredkujú inštrukcie pre prehliadač, na základe ktorých dokáže odpovedajúce webové stránky vykresliť. Okrem toho, že vykresľovacie jadro (z ang. *rendering engine*) webového prehliadača zobrazuje užívateľovi naformátovaný obsah stránky, vytvára jej hierarchický model, ktorý sa nazýva DOM (*Document Object Model*) strom [34]. V kroku predspracovania pri procese vykresľovania sa taktiež vytvára hierarchický model kaskádových štýlov CSSOM [14]. Prostredníctvom týchto modelov je rôznym programom a skriptom umožnené dynamicky pristupovať a aktualizovať obsah, štruktúru a štýly vykreslenej stránky.



Obr. 2.2: Schéma kooperujúcich súčastí pri vytváraní vykresľovacieho stromu pre jeho následné zobrazenie v prehliadači. Prevzaté z [14].

Po dokončení predspracovania a kombinácie oboch modelov má prehliadač dostatok informácií na zostavenie výslednej webovej stránky a jej následné zobrazenie. Dokument,

ktorý zahŕňa všetok obsah a štýly spracované vo vykresľovacom procese sa nazýva vykresľovací strom (z ang. *rendering tree*) [13]. DOM strom pôvodného HTML dokumentu a vykresľovací strom pre rovnakú webovú stránku nie sú identické. Vykresľovací strom totiž obsahuje okrem zdrojového HTML kódu aj aplikované kaskádové štýly, ktoré môžu značne ovplyvniť štruktúru stránky a pozície či viditeľnosť jednotlivých HTML elementov. Rovnakým spôsobom môže tento strom ovplyvniť aj JavaScript kód spúšťaný na stránke. Proces vytvárania vykresľovacieho stromu v zjednodušenej forme je znázornený na obrázku 2.2.

Segmentácia webových stránok úzko súvisí so spomenutými pojmami, pretože tvoria základný koncept pri vytváraní a spracovaní webovej štruktúry a jej obsahu. Mnohé segmentačné metódy sa spoliehajú práve na DOM strom, ktorý predstavuje rozhranie pre prístup k jednotlivým HTML elementom alebo značkám (*tagom*). Iné prístupy sa naopak sústredia predovšetkým na vizuálne vlastnosti a využitie vykresľovacieho stromu.

2.2 Aplikácia segmentácie

V oblasti rozdeľovania webových stránok na vizuálne oddelené, sémanticky súvisiace časti došlo v priebehu posledných dvoch desaťročí k výraznému pokroku. Zatiaľčo niektoré z navrhnutých metód sú z hľadiska využitia výsledkov všeobecné, iné sa zameriavajú na riešenie konkrétnych úloh [38]. Rôzne metódy segmentácie webových stránok sa od seba navzájom odlišujú využitím, úrovňou podrobnosti (granularitou) a prístupmi, na ktorých sú založené.

Segmentácia stránok sa stala dôležitou úlohou predspracovania a analýzy webových dokumentov, pretože predstavuje základ pre mnoho aplikačných domén. Hlavným využitím väčšiny segmentačných metód je **získavanie informácií** a **dolovanie dát** z webových dokumentov.

Už zo začiatku vývoja sa niektoré z metód [37, 30, 10] snažili detekovať a eliminovať časti stránky s neinformatívnym obsahom. Identifikácia blokov hlavného obsahu a odstránenie nepotrebných častí stránky viedli k zlepšeniu presnosti a zvýšeniu efektivity pri **klasifikácii obsahu** a získavaní informácií. Podobné využitie majú aj ďalšie modernejšie metódy [25, 23, 35, 32]. Prvá z nich [25] na extrakciu hlavného obsahu článku využíva DOM strom a heuristické pravidlá, pričom všetkým uzlom priradzuje skóre na základe množstva obsiahnutého textu a počtu odkazov. Princíp metódy [23] spočíva v transformácii DOM stromu na strom blokov obsahujúcich uzly. Jednotlivé uzly sú ohodnotené na základe vizuálnych vlastností špecifikačnými vektormi, pomocou ktorých je možné v strome blokov dohľadať hlavný blok obsahu. Metóda [35] sa svojim algoritmom *EIFCE* snaží pokryť nedostatky a limity predchádzajúcich metód založených čisto na DOM strome alebo vizuálnych vlastnostiach kombináciou oboch prístupov. Odstránenie "šumu", resp. neinformatívnych častí metódou [32] prebieha prostredníctvom označovania blokov rôznou dôležitosťou. Následne sú užitočnejšie bloky vybrané a zoskupované pomocou zhlukovania. Extrahovanie relevantných informácií z webových stránok je hlavným cieľom väčšiny súčasných segmentačných metód. Vzhľadom na to, že objem webových stránok má stále rastúci trend, dolovanie dát z webových dokumentov je predmetom veľkého množstva súčasných výskumov.

Uplatnenie ďalšej navrhnutej metódy [12] založenej na označovaní hraníc v grafe tvoriacich segmenty pomocou Viterbiho algoritmu a neurónovej siete vidia autori v efektívnejšom **webovom prehľadávaní** (*web crawling*), ktoré môže byť pomocou segmentácie zjednodušené, čím sa dokáže predísť obmedzeniam súčasných webových prehľadávačov.

Segmentácia sa tiež využíva pri transformácii **adaptívneho zobrazenia** pre mobilné a iné špecializované zariadenia s menšími obrazovkami. Identifikácia súvislých častí webových stránok, ktoré by mali ostať nerozdelené je pri tomto procese obzvlášť dôležitá. Prispô-

sobenie a prípadné prekódovanie stránky vyžaduje rýchlu a efektívnu extrakciu informácií hlavného obsahu bez zbytočných neinformatívnych blokov. Jednou z prvých segmentačných metód so zameraním na adaptívne zobrazenie bola metóda [8], ktorá rozdeľovala stránku na menšie časti tak, aby sa ju bolo možné prehliadať bez nutnosti horizontálneho "skrolovania". Ďalšou metódou, ktorá diskutuje využitie segmentácie v adaptívnom zobrazení je už spomínaná metóda [25]. Nedávno publikovaná metóda [9] je založená čisto na vizuálnych vlastnostiach obsahu stránky. Autori vidia využitie segmentačnej metódy v poskytovaní alternatívneho zobrazenia pre používateľov so špeciálnymi potrebami, ktorí na pohodlné prehliadanie webu potrebujú podporné technológie. Motiváciou autorov bolo vylepšenie zvukových čítačiek textu pre zrakovo postihnutých za účelom lepšej prístupnosti webu. Navrhnutá metóda tiež podporuje selektívnu prezentáciu celého obsahu redukciou menej dôležitých a zdôraznením ústredných prvkov, čo môže byť prínosné hlavne pre starších užívateľov.

Okrem už spomenutých využití sa môže segmentácia uplatniť v rade ďalších aplikácií a predpokladáme, že jej potenciál ešte nie je úplne využitý. Ďalšími možnými použitiami segmentácie sú podľa preskúmaných metód v [36] oblasti: detekcia vizuálnych podobností stránok na odhalenie *phishingu* a duplikátov, archivácia webu, vyhodnotenie vizuálnej kvality, identifikácia používateľských oblastí záujmu, a i.

2.3 Rozdelenie segmentačných metód

Metódy segmentácie webových stránok je možné rozdeliť na základe viacerých kritérií. Vzhľadom na to, že niektoré metódy isté vlastnosti definujú explicitne, má zmysel ich priraďovať podľa kritérií len do určitých rozdelení na základe použitého princípu a ich charakteru.

V závislosti od cieľovej aplikácie môže byť vyžadovaná rozdielna granularita (úroveň podrobnosti) segmentácie, ktorá zodpovedá vizuálnej konzistencii segmentov identifikovaných na stránke [38]. Podľa požadovanej úrovne granularity môžu byť segmentačné metódy rozdelené do troch hlavných skupín [38]:

- Úroveň sémantických blokov – V oblasti získavania informácií [35] a čistenia dokumentov je potrebná segmentácia stránky na identifikovanie základných informatívnych blokov, ktorými sú napríklad hlavná obsahová časť, záhlavie alebo zápätie. Táto granularita predstavuje najmenej podrobnú úroveň.
- Úroveň odsekov – V niektorých aplikáciách, akou je napríklad klasifikácia informácií do logických segmentov založená na vizuálnych vlastnostiach elementov [6], sa vyžaduje jemnejšia granularita, ktorá odpovedá jednotlivým logickým častiam obsahu ako sú nadpisy, tabuľky, odseky, či položky zoznamu.
- Úroveň dátových polí – Najjemnejšia úroveň granularity sa zvyčajne vyžaduje v oblasti extrakcie informácií, kde je potrebná identifikácia a získanie jednotlivých dátových polí.

Množstvo súčasných metód segmentácie stránok je založených na jednom z dvoch možných spôsobov spracovania [34]. Každý z nich definuje akým spôsobom prebieha prechod a spracovanie DOM stromu v procese extrakcie jednotlivých blokov:

- Zhora-nadol (*top-down*) – Pri tomto prístupe sa na začiatku spracovania považuje celá webová stránka za jeden základný blok. Jeho iteratívnym delením sa potom vytvárajú

menšie bloky na základe rôznych vlastností (vizuálnych, obsahových, významových). Typickým príkladom tohto prístupu je algoritmus *VIPS* [7], ktorý je bližšie popísaný v podkapitole 2.3.3.

- Zdola-nahor (*bottom-up*) – Narozdiel od predošlého prístupu, proces začína výberom listových uzlov DOM stromu považovaných za atomické jednotky. Následne sú tieto uzly iteratívne zlučované do väčších celkov (grafov, zhlukov), až dokým neplatí definovaná podmienka. Príkladom je zhlukovacia segmentačná metóda [3].

V priebehu dlhoročného výskumu segmentácie webových stránok vzniklo množstvo segmentačných algoritmov, ktoré sa od seba odlišujú hlavne použitým prístupom. Existuje viacero rozdelení [11, 21, 34], na základe ktorých je možné jednotlivé segmentačné metódy klasifikovať a porovnať. Základnú klasifikáciu, ktorú používa väčšina výskumníkov predstavuje [11], ktorá rozdeľuje segmentačné metódy podľa prístupu do 4 hlavných skupín. Toto rozdelenie zahŕňa všetky najvýznamnejšie prístupy, ktoré boli používané v priebehu vývoja segmentácie webových stránok [21]. V nasledujúcich podkapitolách sú popísané jednotlivé prístupy a pri každom z nich je uvedený niektorý zo zástupcov. Podrobnejšie sa venujeme vizuálne orientovaným metódam, keďže sa ukázali v procese segmentácie ako najpresnejšie.

2.3.1 DOM-based metódy

Metódy založené na tomto prístupe využívajú pri segmentácii čistú objektovú reprezentáciu HTML kódu bez pridaných vizuálnych vlastností (CSS), ktorú predstavuje DOM strom. Mnohé z týchto metód využívajú pri spracovaní konečnú množinu heuristických pravidiel, čo predstavuje isté obmedzenie, ktoré negatívne ovplyvňuje ich presnosť a efektivitu segmentácie.

Základným predpokladom pre správne fungovanie tohto prístupu je, že HTML štruktúra odráža sémantiku webovej stránky. Množstvo moderných webových stránok ale tento predpoklad nespĺňa a niektoré z nich tiež nedodržiavajú HTML štandard, čo rovnako neprispieva k správnym výsledkom segmentácie. Ďalším problémom je, že DOM strom môže byť bez aplikovania CSS a spustenia JavaScriptu interpretovaný nevhodným spôsobom. Samotná implementácia týchto metód je ale pomerne jednoduchá a časovo nenáročná, keďže nie je vyžadované vykreslenie stránky [21].

Príkladom tohto prístupu môže byť algoritmus `PageSegmenter` [33], ktorého hlavnou myšlienkou je hľadanie podobnosti medzi cestami od koreňa k listovým uzlom s cieľom identifikovať uzly patriace do rovnakého sémantického bloku. Ďalším podobným algoritmom založenom na spracovaní DOM stromu je algoritmus `WISH` [16], ktorý na extrakciu informácií z HTML dokumentu využíva tzv. dátové záznamy. Tie predstavujú uzly DOM stromu, ktoré sa nachádzajú na rovnakej úrovni a obsahujú sekvencie rovnakých potomkov ale s odlišným obsahom.

2.3.2 Text-based metódy

Tento prístup sa od predchádzajúceho líši práve v tom, že pri spracovaní vôbec neberie do úvahy stromovú štruktúru HTML, teda pozície, vzťahy a úrovne zanorenia jednotlivých HTML elementov. Analyzované sú výhradne vlastnosti textového obsahu, ktorými sú napríklad hustota textu a odkazov v rôznych častiach stránky. Samotný text môže byť pritom extrahovaný buď priamo zo zdrojového kódu alebo z vykresľovacieho stromu, čo závisí od konkrétnej metódy [27].

Metódy založené na spracovaní textu vychádzajú z kvantitatívnej lingvistiky, ktorá naznačuje, že textové bloky s podobnými vlastnosťami pravdepodobne patria k sebe, a je ich teda možné zlúčiť do jedného bloku. Určenie optimálnej prahovej hodnoty podobnosti závisí od požadovanej granularity a u väčšiny metód je potrebné ju určiť experimentálne [21].

Text-based algoritmy bývajú rýchle a jednoducho implementovateľné, pretože fungujú nezávisle na DOM strome. Každopádne neberú do úvahy obrázky, formátovanie obsahu ani vizuálne a štrukturálne vlastnosti stránky, čím sa tento prístup segmentácie považuje za neúplný [27] a v súčasnosti sa nepoužíva.

Príkladom tohto prístupu môže byť algoritmus `BlockFusion` [20], ktorý definuje heuristiku hustoty textu nazývanú *hustota tokenov*. Jej hodnota sa vypočítava ako podiel počtu slov a riadkov, ktorých dĺžka je v časti textového uzla obmedzená na 80 znakov. Hustota je vyjadrená pre každý atomický blok textu a na základe určenej prahovej hodnoty sú jednotlivé bloky následne zlučované.

2.3.3 Vision-based metódy

Vizuálne prístupy sa zameriavajú na analýzu vizuálnych vlastností obsahu webových dokumentov, podobným spôsobom ako ich vníma človek. Používatelia nevnímajú vykreslenú webovú stránku v prehliadači ako jeden sémantický celok, ale podvedome ju rozdeľujú na viacero funkčných blokov pomocou priestorových a vizuálnych vlastností [27].

Tieto metódy síce sprostredkujú o stránke najviac informácií, zároveň sú ale výpočetne najnáročnejšie. Získanie potrebných vizuálnych informácií vyžaduje spracovanie dokumentu pomocou vykresľovacieho jadra, aby bolo možné vypočítať štýly a rozloženie jednotlivých prvkov. Zohľadnenie vizuálnych informácií v porovnaní s predošlými prístupmi umožňuje dosiahnuť vyššiu presnosť segmentácie. Na druhej strane nevyhnutnosť vykresľovania a spracovávanie zložitejších webových dokumentov zvyčajne vo viacerých krokoch, spôsobujú, že prístupy založené na vizuálnych vlastnostiach sú výrazne pomalšie a menej škálovateľné [21].

Vizuálne vlastnosti, s ktorými pracuje väčšina metód okrem iných zahŕňajú farbu pozadia, štýl, rozloženie, viditeľnosť, veľkosť, farbu, váhu a typ písma. Na stránke zvyčajne rozlišujeme textové a obrázkové elementy, prípadne uzly, ktoré dané elementy obsahujú a istým spôsobom definujú štruktúru a rozloženie webovej stránky. Pre všetky elementy na stránke je taktiež možné určiť minimálny ohraničujúci obdĺžnik (*minimum bounding box*), taktiež použiteľný v procese segmentácie, ktorý je definovaný svojou výškou, šírkou a počiatočnými súradnicami x, y.

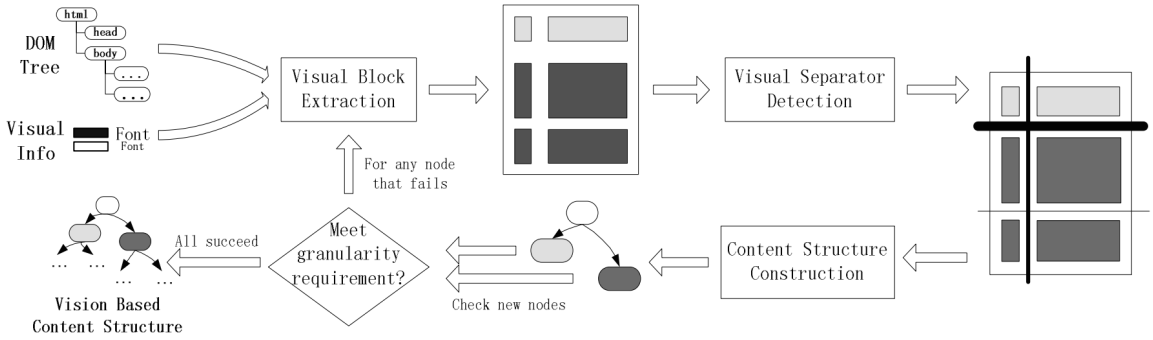
VIPS

Jednou z prvých a najznámejších vision-based metód je pravdepodobne algoritmus VIPS [7] (*Vision-based Page Segmentation Algorithm*), ktorý pracuje nad vykresľovacím stromom. Cieľom metódy je segmentácia webovej stránky na sémantické bloky na základe jej vizuálnej reprezentácie. Ako už bolo spomenuté rozdeľovanie stránky na významovo oddelené časti, ktoré odpovedá vizuálnemu vnímaniu používateľom lepšie odráža jej sémantickú štruktúru [7].

Medzi hlavné charakteristiky tohto algoritmu patrí prechod webovou stránkou zhoranadol, použitie heuristických pravidiel, nezávislosť segmentácie iba na zdrojovom HTML kóde a vytváranie hierarchickej štruktúry na výstupe. Autori definujú webovú stránku ako rekurzívnu štruktúru blokov, ktoré sa nepretínajú. Na základe toho sa získava množina vizuálnych oddeľovačov, pomocou ktorých je stránka rozdelená na jednotlivé segmenty. Nad

každým blokom je rekurzívne vykonávaný segmentačný proces, až dokým nie je dosiahnutá požadovaná granularita [7].

Proces segmentácie prebieha vo viacerých iteráciách, v ktorých sa spracovávajú podstromy identifikované v predchádzajúcich priechodoch stromu. Každá iterácia pritom pozostáva z troch základných krokov: extrakcia vizuálnych blokov, detekcia vizuálnych oddeľovačov a konštrukcia vizuálnej štruktúry obsahu. Jednotlivé kroky algoritmu a ich nadväznosť je znázornená na obrázku 2.3.



Obr. 2.3: Proces segmentácie algoritmom VIPS. Prevzaté z [7].

1. Extrakcia vizuálnych blokov
   V tomto kroku prebieha identifikácia a extrakcia vizuálnych blokov v danom podstrome, pričom vo všeobecnosti platí, že vizuálny blok môže predstavovať každý uzol stromu. Na základe heuristických pravidiel definovaných v [7] priebeha rozhodovanie, či má byť daný uzol rozdelený. Ak rozdelený byť nemá, je extrahovaný.

2. Detekcia vizuálnych oddeľovačov
   Po extrakcii všetkých blokov na danej úrovni prebieha detekcia oddeľovačov. Vizuálne oddeľovače (separátory) sú zvislé alebo vodorovné čiary, ktoré sa nepretínajú so žiadnymi vizuálnymi blokmi a sú definované počiatočným a koncovým bodom [7]. Separátory sa používajú na rozlíšenie blokov s rôznou sémantikou, z čoho vyplýva možnosť priradiť im váhu na základe vizuálnych rozdielov medzi ich susediacimi blokmi.

3. Konštrukcia vizuálnej štruktúry obsahu
   Po detekcii separátorov a nastavení ich váhy môže byť zostavená odpovedajúca štruktúra obsahu. Proces konštrukcie začína od separátorov s najnižšou váhou a bloky, ktoré separátory oddeľujú sú zlúčené, čím vytvoria nový uzol. Následne je pre každý extrahovaný uzol reprezentujúci vizuálny blok vypočítaná hodnota stupňa súdržnosti DoC (*Degree of Coherence*), ktorá je určená na základe maximálnej váhy separátorov v danom uzle. Pre každý uzol, ktorý nespĺňa požadovanú granularitu ($DoC > PDoC$), sa pokračuje fázou extrakcie vizuálnych blokov. V prípade, že všetky uzly spĺňajú určenú granularitu, je zostavená výsledná vizuálna štruktúra [7].

Algoritmus VIPS priniesol do sveta segmentácie nový prístup, ktorým sa inšpirovalo mnoho ďalších navrhnutých metód. S príchodom *HTML5*, ktorý obohatil pôvodný štandard o množstvo nových značiek bola navrhnutá vylepšená verzia algoritmu [2], ktorá rozširuje pôvodný algoritmus o množinu heuristických pravidiel.

2.3.4 Hybridné metódy

Hybridné prístupy kombinujú viacero rozličných prístupov, pričom ich hlavným cieľom je dosiahnutie vyššej presnosti segmentácie.

Najčastejšie je kombinovaný DOM-based a vision-based prístup. Príkladom takého prístupu môže byť algoritmus navrhnutý v [28], ktorý kombinuje obsahovú štruktúru DOM stromu s vizuálnymi informáciami získanými z webového prehliadača. Ide v podstate o modifikovanú verziu algoritmu VIPS s cieľom zvýšiť presnosť extrakcie vizuálnych blokov ešte hodnovernejšie takým spôsobom, akým ich skutočne vníma používateľ.

Kapitola 3

Box Clustering Segmentation

Táto práca sa zaoberá implementáciou segmentačnej metódy Box Clustering Segmentation vo webovom prehliadači, preto je dôležité ju detailne popísať z jej teoretického hľadiska. Metóda BCS patrí do kategórie vizuálne orientovaných (*vision-based*) metód, ktoré na vytvorenie segmentácie stránky využívajú predovšetkým vizuálne vlastnosti elementov. V tejto kapitole sú detailne popísané teoretické základy, na ktorých je metóda postavená. Na začiatku kapitoly sú v krátkosti popísané jej základné vlastnosti, princíp a spôsob spracovania webovej stránky. Vysvetlené sú taktiež jednotlivé výpočetné kroky. Z hľadiska vytvárania segmentácie na základe zhlukovania je najdôležitejším aspektom práve vhodne definovaný model podobnosti, ktorého popis je uvedený v podkapitole 3.4. Všetky uvedené definície a informácie použité v tejto kapitole boli prevzaté z publikácie [38].

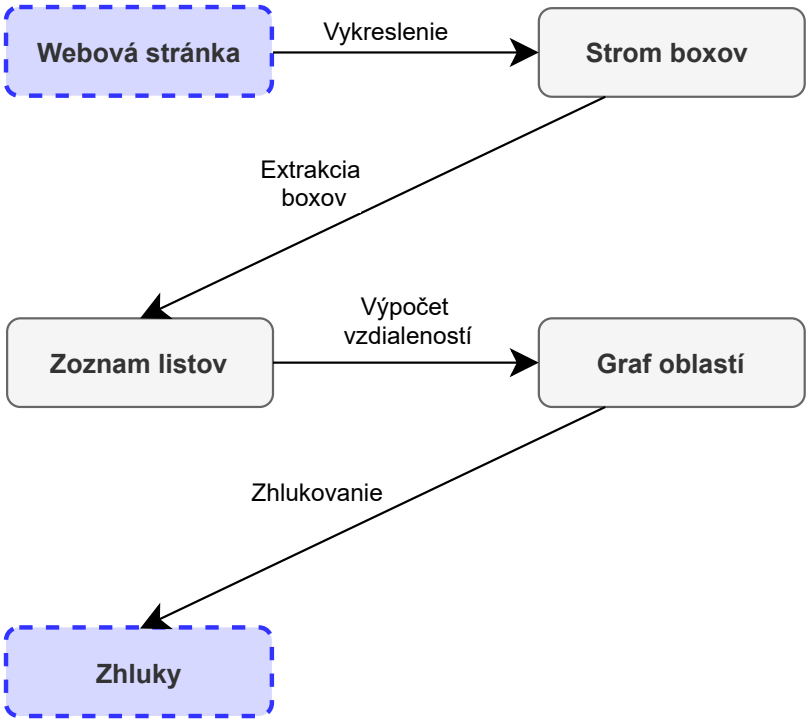
3.1 Základný princíp

Hlavným problémom väčšiny segmentačných algoritmov je ich časová náročnosť a závislosť na implementačných detailoch spracovávaných webových dokumentov. Z toho dôvodu je metóda BCS orientovaná čisto na vizuálne vlastnosti.

Mnohé metódy na výstupe produkujú zväčša hierarchickú štruktúru, ktorá rovnako komplikuje samotné spracovanie a pre väčšinu vyššie spomenutých využití sú aj tak najdôležitejšie listové uzly hierarchie. Namiesto hierarchického modelu je teda efektívnejšie pracovať priamo so zoznamom relevantných listových uzlov. Výstupom metódy je množina oblastí (zhlukov), ktoré sa nachádzajú na jednej úrovni a nepredstavujú hierarchickú štruktúru. To vychádza z myšlienky, že užívateľ nevníma stránku ako hierarchickú štruktúru, ale ako skupinu oblastí na jednej úrovni, ktoré sa odlišujú svojou sémantikou.

Jedným z hlavných cieľov metódy je vyššia rýchlosť spracovania pri segmentácii v porovnaní s inými *vision-based* algoritmami (napr. VIPS [7]). To je zabezpečené výberom len tých oblastí, ktoré sú dôležité z pohľadu užívateľa a zohľadnením najzákladnejších vizuálnych vlastností. Rýchlosť výpočtu taktiež závisí na zvolení vhodných dátových štruktúr, ktoré reprezentujú dáta v priebehu výpočtu. Algoritmus metódy BCS pri vytváraní segmentácie využíva zhlukovanie a vhodný model vizuálnej a zhlukovej podobnosti.

Samotný výpočet prebieha v 4 hlavných krokoch, ktoré sú znázornené na obrázku 3.1. Jednotlivé uzly predstavujú stavy, v ktorých sa nachádzajú práve spracovávané dáta. Hrany medzi uzlami reprezentujú akcie, ktoré sú nad dátami vykonávané. Vstupom algoritmu je webová stránka, ktorá je vykreslená vo webovom prehliadači a výstupom je množina zhlukov reprezentujúcich segmenty.



Obr. 3.1: Kroky výpočtu algoritmu BCS. Prevzaté z [38].

Prvý krok algoritmu môže byť považovaný za krok predspracovania a nie je súčasťou pôvodného algoritmu BCS. Získanie stromu boxov môže byť prevedené viacerými spôsobmi, napríklad aj algoritmami na detekciu hrán. V našej implementácii ale uvažujeme tento krok ako základný krok algoritmu, keďže strom boxov je reprezentovaný vykresľovacím stromom, z ktorého budú získavané vizuálne vlastnosti elementov. Nasledujúcim krokom spracovania je extrakcia boxov, v ktorej sú vybrané len tie oblasti, ktoré majú význam z hľadiska získavania informácií. Jedná sa hlavne o textové a obrázkové elementy. Ďalším krokom je vypočítavanie vzdialeností, ktoré reprezentujú vzájomné vzťahy medzi extrahovanými boxmi. Výstupom tohto kroku je tzv. graf oblastí. Posledným krokom je samotné zhlukovanie oblastí s využitím vypočítaných vzdialeností v predchádzajúcom kroku.

3.2 Extrakcia boxov

Vykreslenie stránky zabezpečuje transformáciu vstupného HTML dokumentu spolu s pridanými kaskádovými štýlmi do vykresľovacieho stromu, ktorý môžeme chápať ako strom boxov. Z hľadiska štruktúry odpovedá strom boxov vstupnému dokumentu a je možné ho zobraziť vo webovom prehliadači.

Strom boxov sa skladá z jednotlivých obdĺžnikov (boxov), ktoré reprezentujú elementy na vykreslenej stránke. Keďže boxy môžu byť vzájomne vnorené vytvárajú hierarchickú stromovú štruktúru, v ktorej je viditeľné zanorenie jednotlivých elementov. Typicky sa jedná o textové a obrázkové uzly, či uzly s nepriehľadným pozadím.

Samotná extrakcia boxov predstavuje prvý krok spracovania v algoritme BCS. Jej vstupom je strom boxov vytvorený behom kroku predspracovania, ktoré predstavuje spomínané vykreslenie. Výstupom je vybraná podmnožina boxov s obmedzenou množinou vizuálnych

informácií dôležitých pre ďalšie spracovanie. Relevantné boxy extrahované v tejto fáze predstavujú základné bloky, ktoré sa využijú v následnom procese zhlukovania.

Informácie, ktoré o boxoch potrebujeme získať sú potrebné pri vypočítavaní vzdialeností (podobností) na základe použitého modelu podobnosti. Jedná sa o nasledujúce vlastnosti:

- farba boxu,
- pozícia na stránke,
- veľkosť a tvar.

V tomto kroku je pri výbere relevantných boxov dôležité zahrnúť len tie boxy, ktoré sú na stránke skutočne vykreslené a viditeľné. Práve tie sú dôležité z hľadiska ďalších krokov výpočtu. Samotný výber sa prevádza pomocou rekurzívneho *pre-order* algoritmu, ktorý vo vstupnom strome boxov identifikuje požadované uzly. Ide predovšetkým o listové uzly, ktoré reprezentujú skutočný obsah vykreslenej stránky. Všetky relevantné uzly sú vyberané na základe nasledujúcich pravidiel:

1. Textové uzly, ktoré reprezentujú riadok textu alebo jeho časť sú vybrané vždy. V strome boxov sú graficky reprezentované minimálnym boxom, ktorý ohraničuje obsah textového uzla.
2. Obrázkové uzly sú vybrané vždy, automaticky. Keďže reprezentujú konkrétny obrázok poznáme ich veľkosť a umiestnenie v rámci stránky.
3. Boxy bez potomka, ktoré nespadajú do predchádzajúcich kategórií sú vynechané a v ďalších krokoch výpočtu nefigurujú.
4. Boxy s jedným potomkom, ktoré nepatria do predchádzajúcich kategórií sú považované za podstromy, v ktorých sa následne hľadajú vetvy. Ak v nich nie sú nájdené žiadne vetvy je vybraný najmenší box podstromu s nepriehľadným pozadím. Ak žiadny takýto box neexistuje, je vybraný listový uzol.

Po dokončení rekurzívneho prechodu môže dôjsť k situácii, kedy sú niektoré z boxov vnorené v iných. Takéto prípady sú detekované a ponechané sú len menšie boxy, ktoré sú vizuálne obsiahnuté v iných.

3.3 Spájanie boxov

Po získaní všetkých užitočných boxov prebieha určenie ich vzájomných vzťahov na základe vypočítaných vzdialeností a podobnosti. Prvým krokom je určenie relácie polozarovnania medzi všetkými extrahovanými boxmi, ktorá predstavuje jeden zo základných princípov algoritmu. V rámci tejto kapitoly sú presne popísané jednotlivé formálne definície využitých štruktúr a relácií, ktoré budeme používať v nadväzujúcich kapitolách.

Definícia 1 (Box) *Nech box je definovaný ako n-tica m = (left, right, top, bottom, width, height, color), kde left, right, top a bottom, sú celočíselné hodnoty, ktoré reprezentujú pozíciu boxu. Šírka boxu je potom definovaná ako rozdiel right – left a jeho výška ako bottom – top. Položka color v n-tici predstavuje reprezentatívnu farbu boxu.*

Definícia 2 (Projektované prekrytie, Polozarovnanie) *Nech m a n sú boxy. Projektované prekrytie boxov m a n je definované ako funkcia: pov(m, n) → x, y, o, kde hodnoty x a y značia projektované prekrytie boxov na príslušných osách súradnicového systému webovej stránky a hodnota o vyjadruje, že k prekrytiu nedochádza. Funkcia pov je definovaná nasledovne:*

$$pov(m,n) = \begin{cases} x & m.right \geq n.left \wedge m.left \leq n.right \\ y & m.bottom \geq n.top \wedge m.top \leq n.bottom \\ o & inak \end{cases} \tag{3.1}$$

Hovoríme, že boxy m a n sú v relácii polozarovnania (semi-alignment), ak platí $pov(m,n) \neq o$.

Po vypočítaní funkcie *pov* môžeme pre každú dvojicu boxov, ktoré sú v relácii polozarovnania určiť ich vzájomnú pozíciu.

Definícia 3 (Vzájomná pozícia boxov) *Nech $P = \{a, b, l, r, o\}$ je množina možných vzájomných pozícií medzi dvojicou boxov, pričom a, b, l, r reprezentujú postupne pozície nad, pod, vľavo, vpravo a o značí inú pozíciu. Relatívna pozícia boxu m k boxu n je definovaná ako funkcia pos: (m, n) → P, určená na základe nasledujúcich pravidiel:*

$$pos(m,n) = \begin{cases} a & m.bottom \leq n.top \wedge pov(m,n) = x \\ b & m.top \geq n.bottom \wedge pov(m,n) = x \\ l & m.right \leq n.left \wedge pov(m,n) = y \\ r & m.left \geq n.right \wedge pov(m,n) = y \\ o & inak \end{cases} \tag{3.2}$$

Pomocou uvedených definícií 1 – 3 je možné vyjadriť vzťah medzi každými dvoma ľubovoľnými boxmi na danej stránke. Relácia polozarovnania je tiež využívaná pri výpočte relatívnej vzdialenosti. Je dôležité podotknúť, že počiatok súradnicového systému $(0, 0)$ uvažujeme v ľavom hornom rohu stránky a hodnoty na osi y rastú smerom nadol, čo odpovedá súradnicovému systému webového prehliadača.

Po výpočte vzájomných pozícií medzi všetkými dvojicami boxov je nasledujúcim krokom vytvorenie množiny priamych susedov jednotlivých boxov. Ide o množinu prvkov, ktoré sa na základe pozície boxu k nemu nachádzajú najbližšie a predstavujú teda najpravdepodobnejších kandidátov na vytvorenie zhluku. Na určenie tejto množiny potrebujeme určiť absolútnu vzdialenosť medzi dvoma boxmi. Jej výpočet je vyjadrený v nasledujúcej definicii:

Definícia 4 (Absolútna vzdialenosť, Priama susednosť) *Nech B je množina boxov na webovej stránke a nech m a n sú boxy, pre ktoré platí: $m, n \in B$. Absolútna vzdialenosť medzi nimi je potom definovaná ako funkcia $abs : B \times B \to \mathbb{R}$:*

$$abs(m,n) = \begin{cases} n.top - m.bottom & pos(m,n) = a \\ m.top - n.bottom & pos(m,n) = b \\ n.left - m.right & pos(m,n) = l \\ m.left - n.right & pos(m,n) = r \\ \infty & inak \end{cases} \tag{3.3}$$

Priama susednosť boxu m je potom definovaná nasledovne:

$$\begin{aligned} N_m = \{n \mid n \in B \wedge pos(m,n) \neq o \wedge \nexists k \in B\ : \\ pos(m,k) = pos(m,n) \wedge abs(m,k) < abs(m,n)\} \end{aligned} \tag{3.4}$$

3.4 Model podobnosti

Základom každého segmentačného algoritmu je návrh a použitie vhodného modelu pre vyhodnocovanie podobností medzi jednotlivými elementmi na stránke. Algoritmus využíva zložený model podobnosti založený na dvojúrovňovej hierarchii, v ktorej rozlišuje dva druhy prvkov: boxy a zhluky. Základnú úroveň predstavujú listové uzly, ktoré reprezentujú získané boxy v kroku extrakcie. Tie sú v ďalších krokoch spracovania zoskupované do zhlukov.

V rámci modelu podobnosti rozlišujeme dva druhy podobností. Prvou z nich je základná podobnosť odvodená na základe vizuálnych vlastností medzi dvoma boxmi. Zhluková podobnosť potom vyjadruje podobnosť medzi dvoma zhlukmi alebo zhlukom a boxom. V závislosti od vypočítanej hodnoty podobnosti sa v ďalších krokoch algoritmu rozhodujeme, ktoré boxy budú vo výsledku patriť do jedného zhluku.

3.4.1 Základná podobnosť

Základná podobnosť môže byť síce vypočítaná pre ľubovoľnú dvojicu boxov, ale na základe uvedených definícií má zmysel ju počítať iba pre dvojice, ktoré sú v relácii polozarovnania. Použitie jednoduchého modelu podobnosti založeného len na základných vizuálnych vlastnostiach (relatívna vzdialenosť, tvar, farba) má význam tiež pri prevode algoritmu pre spracovanie iných typov dokumentov. Výpočet základnej podobnosti medzi dvomi boxmi predstavuje nasledujúca rovnica:

$$bsim(m,n) = \begin{cases} 0 & distance(m,n) = 0 \\ 1 & distance(m,n) = 1 \\ \dfrac{\begin{pmatrix} distance(m,n) + \\ sim\_shape(m,n) + \\ sim\_color(m,n) \end{pmatrix}}{3} & inak \end{cases} \tag{3.5}$$

Určenie *vzdialenosti* medzi jednotlivými extrahovanými boxmi predstavuje základný spôsob ako oddeliť vizuálne a sémantické bloky. Na výpočet vzdialenosti medzi dvomi boxmi je medzi nimi v prvom rade potrebné vypočítať absolútnu vzdialenosť ako už bolo spomenuté. Tá sa sa použije na výpočet relatívnej vzdialenosti, ktorá vyjadruje vzdialenosť medzi dvoma boxmi vzhľadom na ich najbližších susedov.

Definícia 5 (Relatívna vzdialenosť) *Nech B je množina boxov na webovej stránke. Pre každý box $m \in B$ a jeho množinu priamych susedov N_m je maximálna vzdialenosť suseda definovaná ako maxd(n) = abs(m, k), kde $k \in N_m \wedge \nexists l \in N_m : abs(m,l) > abs(m,k)$. Pre každý box $n \in N_m$ potom platí, že relatívna vzdialenosť $distance(m,n)$ je vypočítaná nasledovne:*

$$rel_m(m,n) = \frac{abs(m,n)}{maxd(m)} \tag{3.6}$$

$$rel_n(m,n) = \frac{abs(m,n)}{maxd(n)} \tag{3.7}$$

$$distance(m,n) = \frac{rel_m + rel_n}{2} \tag{3.8}$$

Z pozorovania pri porovnávaní veľkostí a tvarov dvoch boxov predpokladáme, že boxy, ktoré vyzerajú tvarovo podobne budú patriť do rovnakého zhluku. Existujú však prípady,

kedy nechceme zoskupovať takto podobné boxy. Pre výpočet podobnosti na základe tvaru použijeme vzťah pozostávajúci z podobnosti pomeru strán (*ratio*) a podobnosti na základe veľkosti (*size*) porovnávaných boxov.

Na výpočet podobnosti pomeru strán dvoch boxov $m, n \in B$ použijeme nasledujúce rovnice:

$$r_m = \frac{m.width}{m.height} \tag{3.9}$$

$$r_n = \frac{n.width}{n.height} \tag{3.10}$$

$$ratio(m,n) = \frac{max(r_m, r_n) - min(r_m, r_n)}{\frac{max(r_m,r_n)^2 - 1}{max(r_m,r_n)}} \tag{3.11}$$

Druhú časť výpočtu predstavuje výpočet podobnosti založenej na veľkosti. Podobnosť medzi dvoma boxmi $m, n \in B$ na základe veľkosti je vypočítaná na základe rovníc:

$$s_m = m.width * m.height \tag{3.12}$$

$$s_n = n.width * n.height \tag{3.13}$$

$$size(m,n) = 1 - \frac{min(s_m, s_n)}{max(s_m, s_n)} \tag{3.14}$$

Celková podobnosť na základe tvaru je vypočítaná ako priemer podobností pomeru a veľkosti:

$$sim\_shape(m,n) = \frac{ratio(m,n) + size(m,n)}{2} \tag{3.15}$$

Každý box môže pozostávať a obsahovať niekoľko rôznych farieb, pričom boxy, ktoré reprezentujú obrázky sa typicky skladajú z výrazne väčšieho množstva farieb ako iné boxy. Netextové boxy s nepriehľadným pozadím sa typicky líšia farbou pozadia. Textové boxy sa potom môžu líšiť farbou textu, ohraničenia alebo pozadia. Pre účely výpočtu je ale potrebné, aby bol každý box reprezentovaný jedinou farbou. U obrázkových boxov teda pracujeme s priemernou alebo dominantnou farbou, u textových s farbou textu a u netextových s farbou pozadia.

Z dôvodu, že hodnoty vzdialeností vypočítané na základe farebnej podobnosti sa musia pri porovnaní zhodovať so všetkými ostatnými zložkami je euklidovská vzdialenosť normalizovaná maximálnou diagonálnou vzdialenosťou ($\sqrt{3}$) v RGB priestore do intervalu $\langle 0, 1 \rangle$.

Výpočet podobnosti na základe farby (*color*) potom predstavuje nasledujúci vzťah:

Definícia 6 (Podobnosť na základe farby) *Nech $m, n \in B$ sú boxy a ich farby sú v modeli RGB reprezentované ako $m.color = (R_m, G_m, B_m)$, $n.color = (R_n, G_n, B_n)$. Podobnosť na základe farby je potom definovaná nasledovne:*

$$sim\_color(m,n) = \frac{\sqrt{(R_n - R_m)^2 + (G_n - G_m)^2 + (B_n - B_m)^2)}}{\sqrt{3}} \tag{3.16}$$

3.4.2 Zhluková podobnosť

Zoskupovanie boxov do zhlukov vyžaduje vyhodnotenie podobnosti dvoch zhlukov alebo zhluku a boxu. Charakteristiky zhlukov nie je možné jednoducho odvodiť od vlastností jednotlivých boxov, ktoré zhluky tvoria a to z dôvodu, že je náročné určiť prínos jednotlivých boxov pre celý zhluk. Z toho dôvodu sa využíva model založený na vnútornej podobnosti zhlukov. Vnútorná podobnosť predstavuje strednú hodnotu základnej podobnosti, ktorá sa počíta na základe boxov v zhluku. Nevyhnutným predpokladom je definícia priamej susednosti zhluku rozšírená tak, aby zahŕňala zhluky aj boxy. Model podobnosti odvodzuje priamu susednosť každého zhluku z priamych susedností všetkých boxov v danom zhluku.

Definícia 7 (Nezozhlukované boxy, Priama susednosť zhluku) *Nech B je množina boxov a C je množina zhlukov na stránke. Ďalej nech $B_c \in C$ je množina boxov, ktoré patria do zhluku c a N_m je množina priamych susedov boxu m. B_U (unclustered) je potom množina boxov, ktoré nepatria do žiadneho zhluku definovaná ako $B_U = \{b \,|\, B; \nexists B_c \in C : (b \in B_c)\}$. Pre zhluk c potom platí, že množina jeho priamych susedov je definovaná nasledovne:*

$$N_c = \{m \,|\, m \in B_U \wedge \exists n \in B_c : n \in N_m\} \cup \{B_d \,|\, D_d \in C \wedge \exists m \in B_c : n \in N_m\} \tag{3.17}$$

Výpočet podobnosti medzi zhlukom $c \in C$ a boxom $b \in B$ je vyjadrený ako podiel pomocných funkcií $card()$ a $cumul()$, pričom $card$ predstavuje koľko boxov zo zhluku c patrí do množiny priamych susedov boxu b a $cumul$ vyjadruje kumulatívnu podobnosť ako súčet hodnôt podobností medzi týmito boxmi:

$$card(c, e) = |\{m \,|\, m \in B_c \wedge e \in N_m\}| \tag{3.18}$$

$$cumul(c, b) = \sum_{\forall m \in B_c} s(m, e) \tag{3.19}$$

Podobnosť medzi zhlukom c a boxom b je potom definovaná nasledovne:

$$csim(c, b) = \frac{cumul(c, b)}{card(c, b)} \tag{3.20}$$

Po popísaní oboch zložiek modelu podobnosti (*bsim, csim*) je definovaná výsledná podobnosť medzi dvomi entitami nasledovne:

Definícia 8 (Podobnosť entít) *Nech B je množina boxov a C je množina zhlukov na stránke. Nech $e_1, e_2 \in B \cup C$ sú dve entity. Podobnosť entít s je potom definovaná ako:*

$$s(e_1, e_2) = \begin{cases} bsim(e_1, e_2) & e_1 \in B \wedge e_2 \in B \\ csim(e_1, e_2) & e_1 \in C \\ csim(e_2, e_1) & e_1 \notin C \wedge e_2 \in C \end{cases} \tag{3.21}$$

3.5 Zhlukovanie

Vstupom samotného zhlukovania je množina boxov B a hodnota zhlukovacieho prahu CT (*Clustering Threshold*) popísaná nižšie. Výstupom algoritmu je potom množina vytvorených zhlukov C. Zhlukovací algoritmus pozostáva zo štyroch základných krokov:

1. Vytvorenie zhlukovacích zŕn (*cluster seeds*)
2. Výber entít na zlúčenie – vytvorenie kandidátov na zhluk
3. Kontrola a vyriešenie prekrytia
4. Overenie a potvrdenie zhluku

Hlavná slučka algoritmu 1 pokrýva prvé dva kroky celého zhlukovacieho algoritmu – vytvorenie zhlukovacích zŕn a výber entít pre následné zlúčenie. Tieto dva kroky sú takmer rovnocenné a líšia sa len typom vstupných entít. Cieľom je nájdenie a výber najpodobnejšej dvojice boxov alebo zhlukov pre ich následné zlúčenie. Ak je aspoň jedna z entít zhluk, vytvorí sa nový kandidát na zhluk. V prípade, že sú obe entity boxy, vytvorí sa nové zhlukovacie zrno. Zrno sa však stane platným až po potvrdení, dovtedy sa považuje len za kandidátne zhlukovacie zrno.

Na riadku 6 v uvedenom algoritme 1 sa uskutočňuje výber dvoch entít na zlúčenie. S cieľom zabrániť nekonečným cyklom sa vybraný vzťah odstráni z odpovedajúcej množiny vzťahov, čo nie je v algoritme znázornené. Po výbere dvoch entít musíme pred vytvorením kandidáta skontrolovať či je medzi nimi podobnosť v prípustnom rozmedzí. V prípade dvoch boxov predstavuje kontrola podobnosti výrazne jednoduchší problém, pri dvoch zhlukoch ale musíme použiť rovnice z kapitoly 3.4.2, aby sme získali skutočnú hodnotu podobnosti vypočítanú na základe podobností jednotlivých boxov.

Algoritmus 1 Zhlukovací algoritmus [38].

```
1:  function BCS(IN: CT, IN OUT: G, OUT: C)
2:      loop
3:          if |B_U| < 2 then
4:              return
5:          end if
6:          m, n ← m, n ∈ B_U ∪ C : ∄x, y ∈ B_U ∪ C : (s(x, y) < s(m, n))
7:          if s(m, n) > CT then
8:              return
9:          end if
10:         create cluster candidate cc
11:         if ∃c ∈ C : c overlaps cc then
12:             continue
13:         end if
14:         if ∃b ∈ B : b overlaps cc then
15:             MergeOverlaps(B, cc)
16:         end if
17:         Commit(cc, G, C)
18:     end loop
19: end function
```

Po vyhľadaní najlepšej dvojice entít na zlúčenie je vykonaná kontrola (riadok 7), či hodnota podobnosti nie je väčšia než zhlukovací prah CT. CT je hodnota v intervale $\langle 0, 1\rangle$ definovaná pred začiatkom zhlukovania a jej hodnota sa v priebehu zhlukovania nemení. Okrem toho, že predstavuje vstupný parameter algoritmu, odpovedá svojou sémantikou

parametru $PDoC$ v algoritme *VIPS*. Správne nastavenie jeho hodnoty je kľúčové pre získanie presných a relevantných výsledkov. Je dôležité poznamenať, že vhodná hodnota CT sa môže líšiť v závislosti od typu a štruktúry spracovávanej stránky.

Po vytvorení kandidáta na nový zhluk sa vykonávajú overujúce testy, na základe ktorých v prípade zlyhania, nedôjde k vytvoreniu zhluku. Zhluk je vytvorený až potom, čo sa prevedú nasledujúce kontroly:

1. Kontrola, či sa kandidát neprekrýva s už existujúcim zhlukom (riadok 11),
2. Kontrola, či sa kandidát neprekrýva s niektorým boxom (riadok 14).

Ak zlyhá prvý kontrolný test, kandidát na zhluk je zrušený automaticky. Ak kandidát neprejde druhým testom, pokúsime sa kandidáta spojiť s príslušným boxom a ak výsledok ich zlúčenia nespôsobí prekrytie, tak potvrdíme vytvorenie nového zhluku, čo je reprezentované funkciou `COMMIT` a daný zhluk označíme ako platný. Pri vytváraní nového zhluku je potrebné vykonať niekoľko akcií:

1. Všetky nové boxy v kandidátnom zhluku sú označené ako členovia výsledného zhluku.
2. Vnútorná podobnosť v rámci zhluku je prepočítaná.
3. V prípade, že bol kandidát na zhluk vytvorený zlúčením dvoch zhlukov, sú tieto zhluky odstránené z množiny C.
4. Nový zhluk je pridaný do množiny zhlukov C.

Po dokončení spracovávania všetkých boxov, môžu v množine B_U ostať také boxy, ktoré nepatria do žiadneho zhluku. Tieto boxy s najväčšou pravdepodobnosťou nie sú dôležité z hľadiska získavania informácií a do výstupu a výsledných zhlukov ich nezahŕňame.

3.6 Vyhodnotenie

V rámci publikácie [38] bol algoritmus BCS implementovaný a následne otestovaný na množine stránok s rôznou zložitosťou. Kvalita a presnosť získaných výsledkov pôvodnej implementácie bola v porovnaní s metódou VIPS približne rovnaká, ak nie horšia. V niektorých prípadoch ale metóda BCS dosiahla lepšie výsledky. Pri niektorých komplexnejších stránkach sa vyskytli problémy s určením správnych kandidátov na zlúčenie do zhluku. Každopádne, BCS mala oproti VIPS úplnú prevahu, čo sa týka času potrebného na vytvorenie segmentácie.

Keďže cieľom práce je implementácia metódy BCS s využitím prehliadača, v kapitole 7 – *Vyhodnotenie a výsledky* bude nami vytvorená implementácia porovnaná a vyhodnotená na základe presnosti a rýchlosti (rámcovo) s existujúcou referenčnou implementáciou metódy BCS.

Kapitola 4

Technológie na vývoj aplikácií s využitím prehliadača

Vývoj webových aplikácií je reprezentovaný ako proces návrhu a implementácie aplikácií dostupných prostredníctvom internetu. Používatelia k týmto aplikáciám pristupujú práve pomocou prehliadača, ktorý sprostredkuje pripojenie na webový server a užívateľské rozhranie pre zobrazenie požadovanej stránky. V prehliadači je stránka vykreslená a používateľ s ňou môže ďalej interaktívne pracovať. Moderné webové prehliadače ako *Chrome* alebo *Firefox* predstavujú aplikácie s pokročilými funkciami, pomocou ktorých sme ako vývojári schopní skúmať štruktúru a rozloženie jednotlivých elementov na stránke, sledovať sieťovú aktivitu či využitie operačnej pamäte.

Keďže cieľom tejto práce je implementácia segmentačnej metódy BCS s využitím webového prehliadača, v tejto kapitole budú popísané niektoré z najpoužívanejších *frameworkov* na jeho automatizáciu. Frameworky predstavujú softvérové balíčky alebo súbory nástrojov, ktoré sa používajú na testovanie a automatizované spúšťanie prehliadača s požadovanou webovou stránkou. S ich pomocou môžeme efektívne pristupovať k vykresľovaciemu stromu prostredníctvom JavaScriptu a zisťovať z neho informácie o vizuálnych a štrukturálnych vlastnostiach jednotlivých HTML elementov.

4.1 JavaScript

Spolu s HTML a CSS je JavaScript jednou z najdôležitejších technológií vo vývoji webu. Je to dynamický skriptovací jazyk, ktorý je interpretovaný na strane klienta (prehliadača) narozdiel od jazykov, ktoré sú typické pre serverovú stranu, napr. PHP, Python alebo Ruby. To okrem iného znamená, že JavaScript a spôsob akým ho môžeme využiť závisí od možností a typu webového prehliadača.

JavaScript zabezpečuje interaktitivtu webových stránok a je dnes nevyhnutou súčasťou väčšiny webových aplikácií. Všetky bežné webové prehliadače preto disponujú výpočetným jadrom (*JavaScript engine*), ktoré interpretuje kód JavaScriptu a tým ho spúšťa na strane klienta. Príkladom môže byť *engine* SpiderMonkey vo Firefoxe alebo v8 v Google Chrome.

Zatiaľčo HTML predstavuje štrukturálnu a CSS prezentačnú vrstvu webovej stránky (štýly), JavaScript odpovedá vrstve, ktorá popisuje jej chovanie. Prostredníctvom DOM stromu je JavaScriptu umožnené pristupovať k elementom a ich atribútom. Skripty dokážu reagovať na užívateľské vstupy a udalosti, čím môžu zmeniť textový alebo vizuálny obsah elementov, prípadne meniť celkové chovanie webového prehliadača [26].

4.2 Node.js

Rastúca komplexnosť webových aplikácií a vyžadovaná podpora a efektivita súbežného spracovania veľkého množstva užívateľských požiadaviek si vyžiadala vývoj novej technológie, ktorú predstavuje Node.js. Node.js je asynchrónne behové prostredie na spúšťanie JavaScriptu mimo webového prehliadača, ktorého hlavným návrhovým cieľom bola práve možnosť vytvárať škálovateľné aplikácie za účelom spracovania veľkého množstva sieťových požiadaviek [1].

Využíva sa predovšetkým na vytváranie backendových služieb a webových API, ktoré sú súčasťou webových aplikácií na komunikáciu s databázou a zabezpečenie požadovanej aplikačnej logiky. Vzhľadom na to, že ide o open-source technológiu, rýchlo si získal obľubu, čo viedlo k vzniku veľkej komunity vývojárov, kvalitnej dokumentácie a množstva verejne dostupných návodov. Jeho výhodu tiež predstavuje dostupnosť naprieč všetkými bežnými operačnými systémami (Windows, iOS, Linux). V praxi sa využíva predovšetkým na vytváranie *real-time* a dátovo intenzívnych aplikácií [24].

V porovnaní s inými populárnymi backend frameworkami ako napr. *Django* alebo *Spring* (Java), predstavuje Node výhodu v rýchlom prototypovaní, v implementácii rovnakej funkcionality za použitia menšieho množstva kódu a tiež v lepšom uplatnení v agilnom vývoji [17, 29].

Keďže Node nie je programovacím jazykom, ale ide o behové prostredie pre spúšťanie JavaScriptu, umožňuje používať jeden programovací jazyk ako na strane klienta, tak aj na strane servera. Pri návrhu a implementácii to sprostredkuje možnosť využiť podobné vzory a v niektorých prípadoch aj rovnaké knižnice. Použitie jedného programovacieho jazyka vo frontende a backende umožňuje vytvárať čistejší a konzistentnejší kód s využitím rovnakých konvencií pomenovaní a nástrojov. Tomuto prístupu odpovedá paradigma *JavaScript everywhere* [1].

Ako serverovú backend technológiu ho využívajú mnohé technologické spoločnosti, ktorými sú napr. LinkedIn, Netflix, Uber, Trello, eBay či PayPal [5].

Node.js funguje ako neblokujúci, čo znamená, že všetky akcie pri spracovaní prevádza asynchrónne. Vďaka tomu, že jedno vlákno dokáže obsluhovať veľké množstvo požiadaviek a namiesto čakania na odpoveď sa venuje ďalšej požiadavke, je spracovanie požiadaviek veľmi efektívne. Komunikácia medzi požiadavkami a vláknom prebieha prostredníctvom udalostí. Využíva sa pri tom fronta udalostí, ktorú vlákno monitoruje a pri vzniku udalosti na ňu príslušným spôsobom reaguje. Táto vlastnosť z Node-u robí ideálnu backend službu pre aplikácie intenzívne na vstup-výstupné operácie týkajúce sa prístupu do súborového systému alebo siete. Zároveň ale nie je vhodný pre výpočetne náročné výpočty ako je spracovanie obrázkov alebo audiovizuálnych dát, ktoré vyžadujú podstatne väčší procesorový čas [24].

Architektúra Node-u využíva na spustenie JavaScriptu výpočetné jadro v8 z Chrome. Narozdiel od objektov dostupných v kontexte webového prehliadača, ktorými sú napr. `document` alebo `window`, Node umožňuje pracovať s objektmi ako je `fs` (*file system*) pre prístup k súborovému systému alebo `http` pre prácu s webovou službou a prenos dát cez HTTP protokol. Node teda zahŕňa okrem výpočetného JavaScript jadra, moduly, ktoré v porovnaní s prehliadačom rozširujú a sprostredkujú odlišné možnosti spracovania.

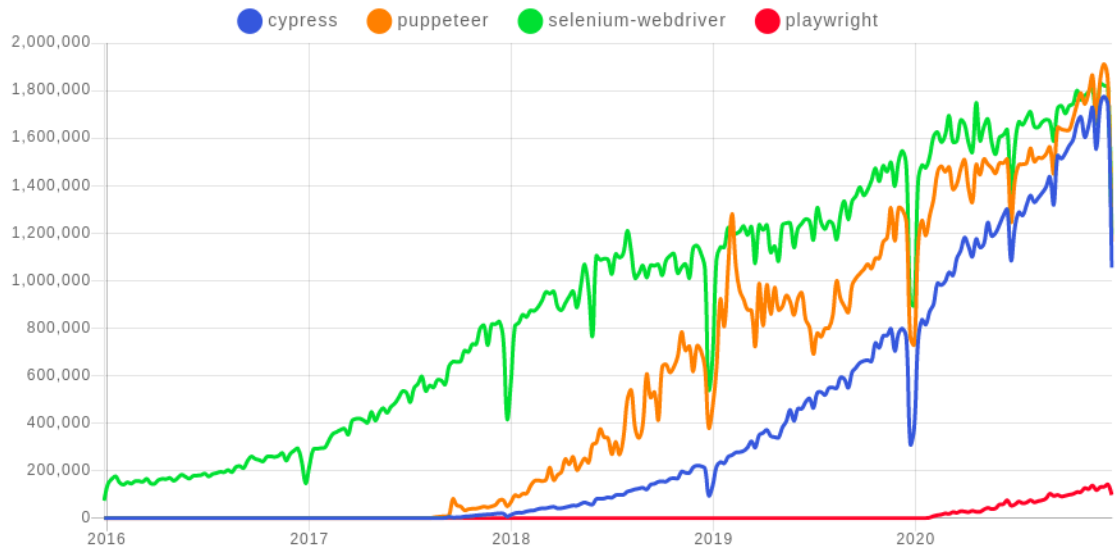
Okrem toho sa dá Node.js vhodne využiť pri automatizovanom spúšťaní prehliadača a pri testovaní webových aplikácií. Automatické spustenie prehliadača a pridávanie vlastných JavaScriptových súborov na stránku otvorenú v inštancii prehliadača je prostredníctvom frameworkov na automatizáciu prehliadača relatívne jednoduché.

4.3 Frameworky na automatizáciu prehliadača

Tieto frameworky umožňujú oveľa viac ako čistú simuláciu webového prehliadača. Okrem spustenia plnej verzie prehliadača dokážu spúšťať aj prehliadač bez hlavičky (tzv. *headless browser*), čo je v podstate webový prehliadač bez grafického užívateľského rozhrania. Ide o odľahčenú a rýchlejšiu verziu prehliadača, čo z neho robí ideálny nástroj pre automatizované testovanie a spracovávanie väčšieho množstva webových stránok, čo môžeme vhodne využiť pri testovaní na väčšej dátovej sade stránok.

Väčšina frameworkov sprostredkuje na prácu s inštanciou prehliadača vlastné API. Pomocou neho je v prehliadači možné simulovať užívateľské interakcie, zaznamenávať screenshoty, vyplňovať alebo zisťovať hodnoty zo vstupných dátových polí a ďalšie automatizované akcie.

Keďže v priebehu vývoja bolo vytvorených množstvo takýchto frameworkov, výber toho správneho si vyžaduje porozumieť ich možnostiam a limitom. Dôležité je zvoliť si taký automatizačný framework, ktorý najlepšie odpovedá účelu a potrebám jeho použitia. V nasledujúcich podkapitolách sú charakterizované momentálne najpopulárnejšie a najpoužívanejšie frameworky na automatizáciu webového prehliadača. Ich popularita môže byť vyjadrená napr. počtom stiahnutí cez *npm*, čo je znázornené na obrázku 4.1.



Obr. 4.1: Porovnanie frameworkov vzhľadom na počet stiahnutí v posledných piatich rokoch. Prevzaté z *npmtrends.com*.

4.3.1 Selenium Webdriver

Selenium Webdriver[1] predstavuje najbezpečnejšiu možnosť pri výbere automatizačného frameworku a to najmä z dôvodu, že v porovnaní s ostatnými frameworkmi bol k dispozícii medzi prvými a na jeho vývoji sa pracovalo podstatne dlhšie. Jeho používanie je zároveň odporúčané aj medzinárodnou komunitou W3C, ktorá sa podieľa na vývoji webových štandardov [31].

[1] https://www.selenium.dev/projects/

Ide o open-source projekt s veľmi aktívnou a rozsiahlou komunitou vývojárov a veľkou podporou naprieč platformami. Jednou z jeho hlavných výhod je fakt, že je súčasťou výrazne širšieho ekosystému, ktorým je Selenium. Vďaka tomu poskytuje väzby a podporuje okrem JavaScriptu aj iné jazyky, akými sú napr. Java, C#, Python, Ruby alebo PHP [31].

V priebehu vývoja bol Selenium Webdriver obohatený o širokú škálu funkcií a ako jediný z diskutovaných frameworkov podporuje všetkých 5 hlavných webových prehliadačov (Chrome, IE, Edge, Firefox a Safari). Pomocou jeho API na spustenie inštancie prehliadača je sprostredkované jednoduché nastavovanie konfigurácie. Umožňuje tiež veľmi elegantným spôsobom vytvárať kód, ktorý simuluje užívateľské akcie v prehliadači ako je písanie textu do textového poľa s istým časovým oneskorením alebo kliknutie na tlačidlo [15, 31].

Aj keď je tento framework veľmi všestranný a disponuje množstvom funkcií a možnosťami integrácie, je zameraný na predovšetkým na testovanie webových aplikácií. V porovnaní s konkurenčnými frameworkami ako je Puppeteer alebo Playwright má oveľa zložitejšie a rozsiahlejšie API, a je tiež náročnejšie ho nasadiť [15].

4.3.2 Puppeteer

Podobne ako Selenium umožňuje Puppeteer simulovať množinu užívateľských akcií, ale používaný je skôr na automatizáciu prehliadača ako na testovanie. Dokáže automatizovane vytvárať screenshoty vo forme obrázkov v požadovanom formáte a generovať PDF súbory špecifických obrazoviek. To z neho robí skvelú voľbu, ak ho chceme použiť vo vizuálnom testovaní. Jeho jednoduchosť a rýchlosť taktiež predstavuje jednu z hlavných výhod, kvôli ktorej ho ako nástroj na automatizáciu prehliadača využíva mnoho vývojárov [15].

Výhodou Puppeteeru je tiež skutočnosť, že je vysoko integrovaný s prehliadačom Chrome (a Chromiom) a jeho ladiacimi a vývojárskymi nástrojmi. To nie je prekvapením, keďže rovnako ako Chrome, bol Puppeteer vyvinutý a je spravovaný spoločnosťou Google [31].

Nevýhodou Puppeteeru je zatiaľ chýbajúca podpora pre ostatné prehliadače, ako je Edge a Safari. V súčasnosti okrem podpory Chrome, Puppeteer čiastočne podporuje Firefox, zatiaľ v beta verzii. Ak testovanie alebo iné aplikačné využitie frameworku na automatizáciu nevyžaduje podporu a nutnosť funkčnosti vo všetkých moderných prehliadačoch, Puppeteer zrejme predstavuje vhodnú možnosť a to najmä vďaka svojej efektivite a rozsiahlej vývojárskej komunite [15].

4.3.3 Cypress

Cypress je voľne dostupný, open-source frontend testovací nástroj a v porovnaní s ostatnými frameworkami ide o kompletné testovacie prostredie. Nepredstavuje univerzálne riešenie automatizovaného testovania, ale jeho hlavným cieľom je prevádzanie komplexného end-to-end testovania webovej aplikácie so zameraním užívateľskú skúsenosť [31].

Najdôležitejším aspektom, ktorým sa odlišuje od konkurencie je jeho rýchlosť, ktorú dosahuje tým, že beží priamo v prehliadači, presnejšie v rovnakom vykonávacom cykle (v hlavnom procese) ako zdrojový kód aplikácie. Z toho dôvodu Cypress nevyžaduje podporu riadiacich protokolov, ktoré sú vyžadované v medziprocesovej komunikácii [31].

Hlavnou nevýhodou tohto frameworku je zložitejšia komunikácia so serverovým backendom, čo pre využitie plného potenciálu vyžaduje inštaláciu dodatočných externých modulov. Ďalšou nevýhodou je väčšia orientácia na testovanie, chýbajúca podpora pre testovanie na viacerých kartách súčasne a skutočnosť, že v danom momente dokáže riadiť iba jedinú inštanciu prehliadača [15].

4.3.4 Playwright

Playwright je najnovším predstaveným frameworkom na automatizáciu prehliadača. Je spravovaný spoločnosťou Microsoft a vznikol v spolupráci bývalými vývojármi z Google, ktorí sa podieľali na vývoji konkurenčného Puppeteeru. Aj z toho dôvodu využíva Playwright rovnakú architektúru ako Puppeteer a tiež má veľmi podobnú (takmer identickú) syntax. Hlavným rozdielom oproti Puppeteeru je rozšírenie o plnú podporu webového prehliadača Safari a Firefoxu, a väčšia podpora a zameranie sa na testovanie, ako na čistú automatizáciu [15, 31].

Framework v porovnaní s Puppeteerom sprostredkuje získavanie elementov na základe textu namiesto CSS selektoru oveľa jednoduchším spôsobom. Ďalšou výhodou je automatické čakanie na načítanie elementov, čím je k nim sprostredkovaný prístup k elementom bez výpisu zbytočných chybových hlášok. U Puppeteera musí byť čakanie na selektor špecifikované explicitne, čo predstavuje isté obmedzenie. Plusom sú tiež spoľahlivé sieťové validácie a možnosť sieťového *mockovania* – vytvárania simulovaných objektov, ktoré kontrolovaným spôsobom napodobňujú správanie reálnych objektov. Efektivita a stabilita Playwrightu je v porovnaní s Puppeteerom približne na rovnakej úrovni. Oba možnosti teda predstavujú z diskutovaných možností správnu voľbu, ak je cieľom práve rýchlosť a jednoduchosť použitia [31].

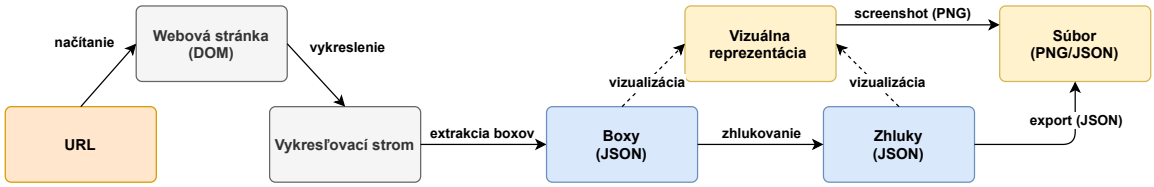
Celkovo ide o moderný automatizačný framework s dobrou stabilitou, jednoduchou možnosťou nasadenia a pohodlného prechodu z Puppeteera vďaka syntaktickým podobnostiam a rovnakým pomenovaniam väčšiny funkcií. Keďže ide o najmladší z uvedených frameworkov, nevýhodou je stále sa rozširujúce API. Oficiálna dokumentácia je v porovnaní s Puppeteerom taktiež na lepšej úrovni, čo sa týka prehľadnosti, obsahu a ukážok kódu. Komunita vývojárov zatiaľ nie je taká rozsiahla ako u iných frameworkov, čo môže spôsobovať komplikácie, ale spomínané syntaktické podobnosti s Puppeteerom umožňujú vyriešiť problémy bez väčších ťažkostí. Aj vzhľadom na nízky počet stiahnutí uvedenom na obrázku 4.1 predstavuje Playwright spolu s Puppeteerom jednu z najlepších možností, keďže hlavným cieľom ich použitia nie je testovanie, ale predovšetkým automatizácia a automatizované spustenie prehliadača s danou stránkou.

Kapitola 5

Návrh riešenia

V predchádzajúcich kapitolách tejto práce sme sa venovali hlavne teoretickým základom. V jednotlivých kapitolách boli uvedené základné informácie o prístupoch k segmentácii a boli diskutované technológie na vývoj aplikácií s využitím webového prehliadača. Najdôležitejšou časťou ale bola kapitola 3, v ktorej bol predstavený základný princíp, proces spracovania a samotný zhlukovací algoritmus metódy BCS. V tejto kapitole je popísaný návrh aplikácie ako celku, ktorého hlavným cieľom je priblížiť jednotlivé problémy a návrhové rozhodnutia. Navrhnuté riešenia sa následne uplatnia v nadväzujúcej implementácii metódy BCS. Na začiatku návrhu boli identifikované a špecifikované základné požiadavky a časti výsledného systému. Bola vymedzená ich činnosť a definované vstupy a výstupy. V rámci návrhu sú tiež priblížené využité dátové štruktúry a funkcie, ktoré sú nevyhnutné pre efektívne spracovanie a riadenú spoluprácu medzi prepojenými komponentmi aplikácie.

5.1 Základná myšlienka



Obr. 5.1: Schéma reprezentujúca spracovanie a proces segmentácie webovej stránky.

Základná myšlienka riešenia spočíva v použití automatizovaného webového prehliadača, ktorý na základe vstupnej URL adresy stránky danú webovú stránku vykreslí. Z vytvoreného vykresľovacieho stromu s aplikovanými a prepočítanými štýlmi je následne možné pristupovať k jednotlivým DOM uzlom a získavať z nich relevantné informácie. To prebieha pomocou JavaScriptu spúšťaného v prehliadači, ktorým sa extrahujú jednotlivé relevantné boxy následne použité pri zhlukovaní. Výstup aplikácie predstavujú zhluky uložené vo vhodnom formáte (JSON) pre prípadné ďalšie spracovávanie alebo vizualizáciu. Jednotlivé skupiny boxov a zhlukov je možné voliteľne vizualizovať v prehliadači alebo ich exportovať do súboru.

V každom vzťahu medzi jednotlivými prvkami navrhnutej architektúry na obrázku 5.1 vzniká rada problémov, ktoré je nutné riešiť a podrobnejšie sa im budeme venovať v nasledujúcich sekciách.

5.2 Automatizované načítanie stránky

Keďže hlavnou myšlienkou aplikácie je na základe sprostredkovanej URL adresy automaticky získať (extrahovať) množinu boxov na základe definovaných pravidiel v časti 3.2, prvý problém predstavuje práve **automatizácia**, teda automatické spustenie prehliadača s danou stránkou. Na riešenie tohto problému je vhodné využiť práve jeden z diskutovaných frameworkov. Ako už bolo spomenuté, jednotlivé frameworky umožňujú programovo otvoriť inštanciu prehliadača so špecifikovanou URL adresou stránky. Načítanie a vykreslenie stránky v inštancii prehliadača teda prebieha automaticky prostredníctvom odpovedajúceho vykresľovacieho jadra.

Druhým problémom je **nastavenie vhodných vstupných parametrov** pre inštanciu prehliadača, ktoré musia byť definované pred jej samotným spustením. U väčšiny frameworkov je nutné alebo žiaduce zadefinovať výšku a šírku otvoreného okna prehliadača (tzv. *viewport size*). V prípade ak tieto parametre nie sú definované, použité sú predvolené hodnoty, pričom šírka okna je zvyčajne príliš malá. Nastavenie týchto hodnôt môže mať výrazný vplyv na segmentáciu a jej výsledky, keďže väčšina stránok sprostredkuje rôzne usporiadanie a zalamovanie elementov na stránke práve v závislosti od šírky a výšky okna.

Ďalším parametrom, ktorý je dôležitý najmä z hľadiska extrakcie, je doba čakania na načítanie obsahu. Frameworky špecifikujú rôzne stavy pri načítavaní obsahu, pričom medzi najdôležitejšie patria: *networkidle* (čaká sa dokým prebieha sieťová aktivita medzi prehliadačom a stránkou) a *domcontentloaded* (čaká sa dokým nie je načítaný obsah DOM). Parameter ovplyvňuje predovšetkým dobu a kvalitu extrakcie, pričom pri čakaní na načítanie obsahu DOM, je doba a kvalita extrakcie pochopiteľne nižšia, keďže vlastný JavaScript stránky môže meniť jej vizuálne vlastnosti.

Posledným problémom, ktorý sa týka automatizovaného načítania stránky je **načítanie JavaScriptu do kontextu prehliadača** a zároveň udržanie hlavného bloku programu relatívne krátkeho a čitateľného. Pomocou zabudovaných funkcií frameworkov je možné vložiť do DOM stromu spracovávanej stránky externé JavaScript súbory vložením HTML značiek (*tagov*) `<script>`. Tým sa v prehliadači sprostredkuje prístup k definovaným funkciám a zároveň sa zabezpečí, že hlavný blok vykonávaný v kontexte prehliadača bude obsahovať výrazne menšie množstvo kódu.

5.3 Extrakcia boxov

Cieľom tejto fázy je identifikácia a získanie všetkých relevantných boxov na stránke na základe definovanej špecifikácie 3.2. Boxy, ktoré nás z hľadiska extrakcie informácií zaujímajú najviac, predstavujú predovšetkým textové a obrázkové uzly. Z pohľadu následného zhlukovania (segmentácie) sú dôležité tiež uzly s pozadím, ktoré obsahujú práve jedného potomka (bez vetiev). U jednotlivých boxov nás zaujíma hlavne pozícia v rámci vykreslenej stránky a ich reprezentatívna farba.

V rámci extrakcie vzniká niekoľko problémov, ktorých navrhnuté riešenia sú diskutované v nasledujúcich podkapitolách. Ide o tieto problémy:

1. efektívne získanie boxov,
2. viditeľnosť DOM uzlov,
3. použitie vhodných dátových štruktúr,
4. získanie farby obrázkových uzlov.

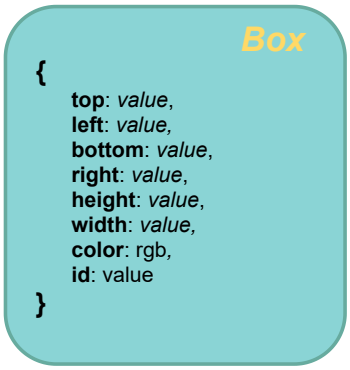
5.3.1 Nároky na efektivitu

Dôležitým aspektom, ktorý je nutné pri extrakcii boxov zvážiť je **efektivita** jej vykonania – doba trvania. Aj keď je primárnym cieľom zabezpečiť čo najefektívnejšiu segmentáciu, proces extrakcie boxov predstavuje zásadný krok predspracovania, ktorý významne ovplyvňuje celkový čas behu aplikácie.

Prvou možnosťou bolo získanie a exportovanie celého vykresľovacieho stromu s informáciami o jednotlivých uzloch a ich prepočítaných vizuálnych vlastnostiach. Už získanie samotného vykresľovacieho stromu by vyžadovalo úplný rekurzívny pre-order prechod DOM stromu. Následne by bolo nutné znovu prechádzať získaný strom v serverovej časti aplikácie, čím by proces extrakcie a výber relevantných uzlov pozostával z dvoch nadväzujúcich rekurzívnych prechodov. Z tohto dôvodu sme sa rozhodli **extrahovať boxy** už po vykreslení stránky **v kontexte prehliadača**, v ktorom je vykresľovací strom už k dispozícii. Zoznam extrahovaných boxov je následne z prehliadača serializovaný späť do serverovej časti pre následný proces zhlukovania.

5.3.2 Dátová štruktúra Box

Použitie vhodnej dátovej štruktúry na reprezentáciu boxu s **ohľadom na serializáciu** hodnôt je dôležitou súčasťou návrhu. Ako už bolo spomenuté, medzi inštanciou prehliadača a serverovou časťou je možné prenášať iba hodnoty, ktoré sú serializovateľné. Aj z toho dôvodu je reprezentácia dátovej štruktúry boxu uvedená vo formáte JSON, ktorý je ľahko čitateľný ako pre stroj tak aj pre človeka.



Obr. 5.2: Dátová štruktúra Box, znázornená vo formáte JSON.

Box ako dátová štruktúra uvedená na obrázku 5.2 obsahuje 8 položiek, dvojíc kľúč-hodnota. Pozíciu boxu v rámci webovej stránky popisujú hodnoty súradníc jej ľavého horného (`left, top`) a pravého dolného rohu (`right, bottom`) v pixloch. Dátová štruktúra ďalej obsahuje výšku (`height`) a šírku (`width`) boxu, pričom ich hodnoty je možné dopočítať ako rozdiel hodnôt vlastných súradníc na príslušnej osi. Ďalšou dátovou položkou je reprezentatívna farba boxu (`color`) uložená ako `RGB` reťazec vo formáte `rgb(n, n, n)`, kde $n \in \langle 0, 255 \rangle$. Z hľadiska následnej segmentácie je jednotlivé boxy dôležité označiť unikátnym identifikátorom (`id`).

Každý box ako dátová štruktúra odpovedá niektorému DOM uzlu (alebo v prípade zalomených textových uzlov jeho časti) na stránke. Samotný proces získania pozície (hodnôt súradníc) a farby bude popísaný v kapitole 6.

5.3.3 Viditeľnosť uzlov

V procese extrakcie je tiež potrebné rozhodnúť a identifikovať len tie boxy, ktoré sú na stránke skutočne **viditeľné**. Boxy, ktoré považujeme za neviditeľné je nutné z výslednej množiny extrahovaných boxov vynechať. To taktiež prispieva k zvýšeniu efektivity pri kontrole na prekrytie boxov, ktorá už priamo predchádza samotnému zhlukovaniu. Rozlišujeme implicitnú a explicitnú neviditeľnosť DOM uzlu (resp. boxu). Explicitná neviditeľnosť znamená, že daný objekt má explicitne uvedenú hodnotu viditeľnosti danú vlastnosťou štýlu (`visibility:hidden|collapse`, `display:none`). Implicitne neviditeľný box je potom box, ktorého aspoň jedna súradnica leží mimo definovaného rozsahu okna prehliadača (*viewportu*), pričom sa nekontroluje maximálna hodnota na osi y (hodnota `bottom`). Dôležité je poznamenať, že počiatok súradnicového systému $(0, 0)$ je v ľavom hornom rohu stránky. Textový uzol obsahujúci iba prázdne znaky je rovnako považovaný za implicitne neviditeľný a preto nie je extrahovaný.

5.3.4 Získavanie reprezentatívnej farby

Získanie reprezentatívnej farby uzlu s jednofarebným pozadím alebo textového uzlu je pomerne jednoduché a priamočiare. Farba boxu, ktorému odpovedá textový uzol je určená na základe farby textu jeho rodičovského HTML elementu. Farba uzlu s pozadím je naopak získaná prostredníctvom prístupu k relevantnej CSS vlastnosti (`background-color`).

Určenie farby, ktorá najlepšie reprezentuje obrázkový element alebo element s obrázkom na pozadí predstavuje podstatne zložitejší problém. Medzi zvažované možnosti ako určiť reprezentatívnu farbu obrázka patrí priemerná a dominantná farba. Na získanie každej z uvedených možností by bolo nutné pristúpiť jednotlivým pixlom obrázka a danú farbu z nich vypočítať. Lepšie riešenie predstavuje využitie externého modulu určeného presne k tomuto účelu, ktorý je na to optimalizovaný. Každopádne sme dospeli k záveru, že farbu obrázka lepšie zachycuje práve jeho priemerná farba, čo ukazuje aj nasledujúci obrázok 5.3.

Obr. 5.3: Porovnanie priemernej a dominantnej farby obrázka: (a) Pôvodný obrázok, (b) priemerná farba, (c) dominantná farba. Obrázok (a) prevzatý z [4].

.

Pri získavaní farby obrázka môže dôjsť k narušeniu politiky *CORS*[1] *(Cross-Origin Resource Sharing)*, ktorá špecifikuje akým spôsobom je definovaný prístup k dátam (a obrázkom) z inej domény, teda prístup cez hranice domén. Ak webový server, na ktorom beží stránka nemá povolený prístup k dátam mimo svojej domény, farbu obrázka nie je možné

[1] https://www.w3.org/wiki/CORS_Enabled

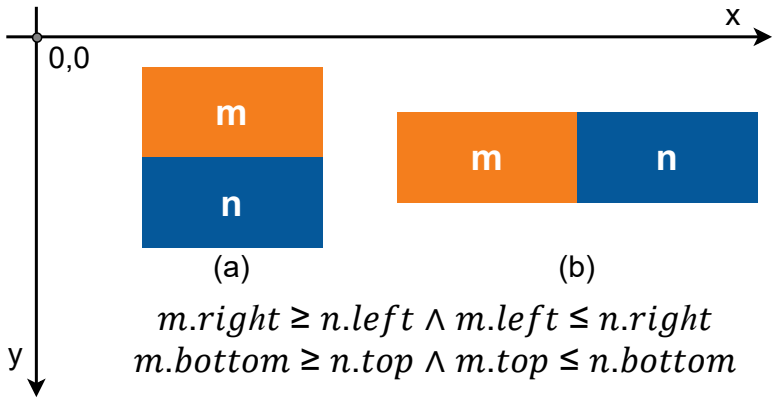
programovo získať. V takých prípadoch je tento problém riešený použitím predvolenej farby (`rgb(128,128,128)`), ktorá má na vypočítavanú podobnosť (a zhlukovanie) najmenší vplyv.

5.4 Zhlukovanie a analýza BCS

Po dokončení procesu extrakcie prebieha samotný proces zhlukovania, ktorého vstupom je zoznam extrahovaných boxov a výstupom jednotlivé segmenty. Analýzou metódy BCS a uvedených definícií v kapitole 3 sme narazili na niekoľko problémov, ktoré súvisia s nejasnou špecifikáciou a ošetrením okrajových prípadov. Tieto problémy sú rozobraté v nasledujúcej časti a pre každý problém je navrhnuté možné riešenie. V ďalších častiach sú následne diskutované návrhové rozhodnutia a riešenia najvýznamnejších krokov týkajúcich sa procesu zhlukovania.

5.4.1 Problémy metódy BCS

Výpočet priamych susedov na základe uvedených definícií počíta so susedmi, ktorí sa nachádzajú od daného boxu v 4 smeroch a ktorí sú vzhľadom na box v relácii polozarovnania. To znamená, že sa pri projekcii na osi x alebo y pretínajú v horizontálnom alebo vertikálnom smere. Zistili sme problém **v definícii polozarovnania** (3.1), ktorá nezohľadňuje vzťah kedy, sú skúmané boxy m, n rovnako široké a zdieľajú horizontálnu hranu, čo môže spôsobiť, že hodnota bude $pov(m, n)$ určená nesprávne. Rovnako to platí pre boxy s rovnakou výškou, ktoré zdieľajú vertikálnu hranu. Je to z dôvodu, že definícia neuvažuje platnosť oboch podmienok súčasne a tiež v nej nie je určená precedencia výberu hodnoty na základe splnenej podmienky. V prípade, že boxy zdieľajú niektorú z hrán po celej výške alebo šírke dôjde k situácii, kedy platia obidve podmienky súčasne, čo ilustruje obrázok 5.4. Riešenie tohto problému spočíva v kontrole platnosti obidvoch podmienok a správneho určenia hodnoty pov, čomu odpovedá detekcia spoločnej hrany.

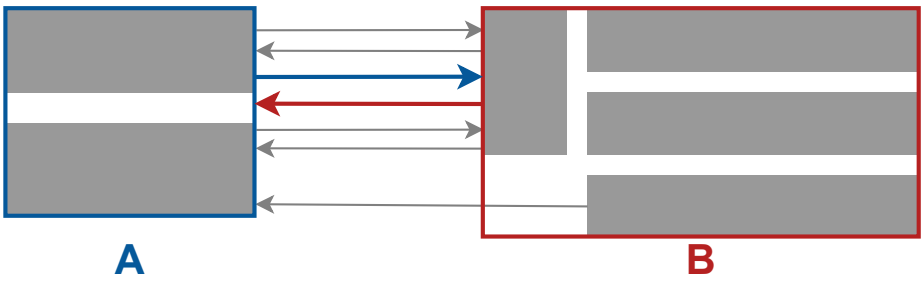


Obr. 5.4: Problém v relácii polozarovnania pri výskyte spoločnej hrany. Pre obidve dvojice boxov (a) a (b) platia súčasne obe uvedené podmienky. Je tiež dôležité si uvedomiť, že počiatok súradnicového systému sa nachádza v ľavom hornom rohu.

Ďalší problém sa vyskytuje v definícii 3.11, ktorá reprezentuje výpočet **zložky podobnosti** medzi boxmi **na základe pomeru ich strán**. Problém vzniká len v špecifickom prípade, kedy sa podobnosť počíta medzi štvorcovým a obdĺžnikovým boxom, ktorý má dlhšiu vertikálnu hranu (výška > šírka). Príklad: Uvažujme boxy m, n, ktoré sú priamymi

susedmi. Box m je štvorec 5×5 a box n je obdĺžnik 5×10. Pomery strán sú potom: $r_m = \frac{5}{5} = 1, r_n = \frac{5}{10}$. Pri výpočte *ratio* je čitateľ rovný 0, pretože $max(r_m, r_n)^2 - 1 = max(1, 0.5)^2 - 1 = 0$, čo vedie na delenie nulou. Riešenie tohto problému spočíva v odhalení a ošetrení podmienkou v prípade možného delenia nulou.

Zo základnej definície priameho susedstva medzi boxmi 4 bola zadefinovaná priama susednosť zhluku 7 medzi zhlukom a entitou (iným zhlukom alebo boxom). Žiadna z definícií ale nezahŕňa skutočnosť, že zhluk môže byť priamym susedom boxu. Na základe toho je výpočet kardinality vzťahu medzi zhlukom c a entitou e v rovnici 3.18 nesprávny v prípade, že entita e je zhluk. Je to z dôvodu, že žiadny box $m \in B_c$ podľa uvedených definícií nemá zhluk e v množine jeho priamych susedov. Riešenie tohto problému predstavuje zložitejšie prepočítavanie kardinality a kumulatívnej podobnosti medzi všetkými boxmi v jednom zhluku a boxmi druhého zhluku. Hodnota kardinality vzťahu podľa definície síce môže byť medzi tými istými zhlukmi rovnaká, čo je znázornené na obrázku 5.5, ale pri výpočte konečnej podobnosti medzi zhlukmi sa musia brať do úvahy všetky vzťahy boxov daných zhlukov relevantných v danom smere. Vypočítaná kumulatívna podobnosť medzi zhlukmi sa môže líšiť práve z dôvodu, že jednotlivé boxy si navzájom nemusia byť priamymi susedmi. Alternatívne riešenie by bolo zahrnutie zhlukov ako priamych susedov boxu, pričom by bolo nutné rozšíriť definíciu priameho susedstva boxu a riešiť iné problémy a komplikácie.



Obr. 5.5: Predpokladajme, že v priebehu segmentácie došlo k vytvoreniu zhlukov A a B. V závislosti od smeru vzťahu medzi zhlukmi A a B má vzťah podľa definície síce rovnakú kardinalitu ($card(A, B) = 2$, $card(B, A) = 2$), ale vypočítaná kumulatívna podobnosť sa môže v rôznom smere líšiť. Pozn.: Šípky znázorňujú existujúci vzťah medzi entitou a jeho priamym susedom. Vzťahy, ktoré sa netýkajú výpočtu kumulatívnej podobnosti medzi zhlukmi A a B nie sú v obrázku znázornené pre väčšiu prehľadnosť.

5.4.2 Nájdenie priamych susedov

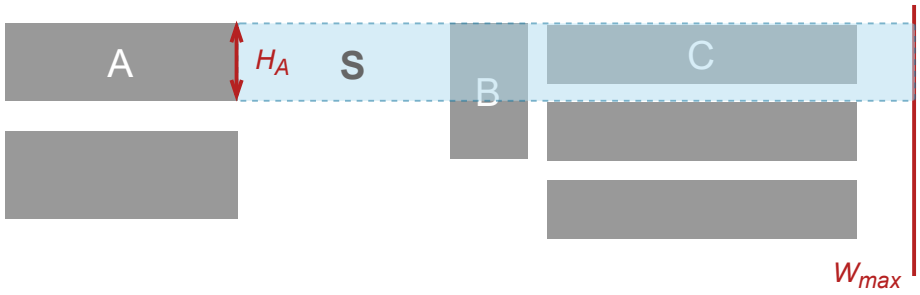
Prvým zložitejším problémom, ktorý je nutné vyriešiť a ktorý priamo prechádza samotnému procesu zhlukovania je určenie vzťahov medzi jednotlivými extrahovanými boxmi, teda výpočet množiny priamych susedov pre každý box. Vzťahy medzi boxmi pochopiteľne nemusia byť obojsmerné, čo znamená že boxy sú navzájom priamymi susedmi. Zároveň to vyplýva aj z definície priameho susedstva boxu 4, pričom najbližší sused v danom smere je box s najmenšou absolútnou vzdialenosťou (ak existuje). Z dôvodu, že vzťahy nie sú obojsmerné je nutné vypočítať množinu priamych susedov pre každý box v každom smere.

Naivným riešením je prechádzať zoznam boxov v dvoch vnorených cykloch (zložitosť $\mathcal{O}(n^2)$) a počítať absolútne vzdialenosti medzi všetkými dvojicami boxov, ktoré sú v relácii polozarovnania na osi x alebo y. Následne by bol daný box alebo boxy s najmenšou absolútnou vzdialenosťou v danom smere vybraný ako priamy sused. Keďže časová zložitosť

je kvadratická, pri zväčšujúcom sa počte boxov by to malo veľmi nepriaznivý dopad na efektívnosť segmentácie.

Oveľa lepším riešením je použitie efektívneho indexovacieho algoritmu dát v 2D priestore, práve z dôvodu, že boxy sú reprezentované ako dvojrozmerné dátové štruktúry s konečnou šírkou a výškou. Zrejme najznámejším algoritmom, ktorý zodpovedá uvedenému popisu a požadovanému účelu použitia je R-strom. Ide o stromovú dátovú štruktúru, ktorá vychádza z B-stromu a slúži na indexovanie priestorových dát vo viacrozmernom priestore. Použitie R-stromu na určenie priamych susedov boxu **napravo** by mohlo vyzerať nasledovne (znázornené tiež na obrázku 5.6):

1. Vlož všetky extrahované boxy do R-stromu
2. Vytvor selektor s výškou boxu a šírkou danou šírkou okna prehliadača (*viewportu*) s počiatkom v pravom hornom rohu daného boxu
3. Vyhľadaj všetky boxy, ktoré selektor prekrýva
4. Vypočítaj absolútnu vzdialenosť medzi boxom a všetkými vyhľadanými boxmi
5. Vyber výslednú množinu boxov s minimálnou absolútnou vzdialenosťou (množina priamych susedov boxu v smere vpravo)



Obr. 5.6: Vyhľadanie priamych susedov boxu A pomocou selektoru S s výškou H_A a šírkou $(W_{max} - A_{right})$, kde W_{max} predstavuje maximálnu šírku *viewportu*. Pomocou selektoru sa z R-stromu získa množina boxov $\{B, C\}$, s ktorými sa vypočíta absolútna vzdialenosť vzhľadom k A na základe rovnice 3.3. Box B sa následne vyberie ako jediný priamy sused boxu A v pravom smere.

Obdobným spôsobom ako pri získavaní susedov smerom vpravo je potrebné proces získania priamych susedov pre každý box opakovať práve 4-krát vo všetkých smeroch: vpravo, vľavo, nadol a nahor. Tým sa zabezpečí efektívnejšie vyhľadanie všetkých priamych susedov všetkých boxov v porovnaní s naivným prístupom.

5.4.3 Odstránenie prekrývajúcich sa boxov

Ešte pred samotným vyhľadaním priamych susedov je nutné z množiny boxov odstrániť boxy, ktoré sa prekrývajú s inými. Špeciálnym prípadom prekrytia boxov je situácia, kedy box vizuálne obsahuje iný box. V takom prípade je box, ktorý vizuálne obsahuje iný box považovaný za kontajner a je odstránený. Odhalenie prekrývajúcich sa boxov v procese extrakcie by vyžadovalo pravidelné prechádzanie celého zoznamu dosiaľ extrahovaných boxov s kontrolou na prekrytie, čo by malo negatívny dopad na efektívnosť extrakcie.

Na detekciu prekrytia medzi boxmi je rovnako vhodné využiť spomínaný R-strom. Všetky extrahované boxy budú postupne figurovať pri kontrole na prekrytie ako selektory do R-stromu. V prípade ak selektor odpovedajúci danému boxu vyhľadá okrem boxu aj ďalší box, môžeme povedať že boxy sa prekrývajú. Následne je nutné zistiť, ktorý box z prekrývajúcej sa dvojice je vizuálne obsiahnutý v druhom. V takom prípade je z dvojice odstránený práve kontajner. Ak ani jeden z boxov nie je vizuálne obsiahnutý v druhom, odstránený je box, ktorý figuroval ako selektor.

5.4.4 Prepočítavanie vzťahov

Najdôležitejším krokom pri vytváraní nového zhluku je prepočítanie množiny jeho priamych susedov a ich odpovedajúcich vzťahov. Vzťah medzi dvomi entitami je reprezentovaný vlastnou dátovou štruktúrou, ktorá obsahuje:

- referencie na entity, ktoré vzťah tvoria,
- smer,
- absolútnu vzdialenosť a
- vypočítanú podobnosť.

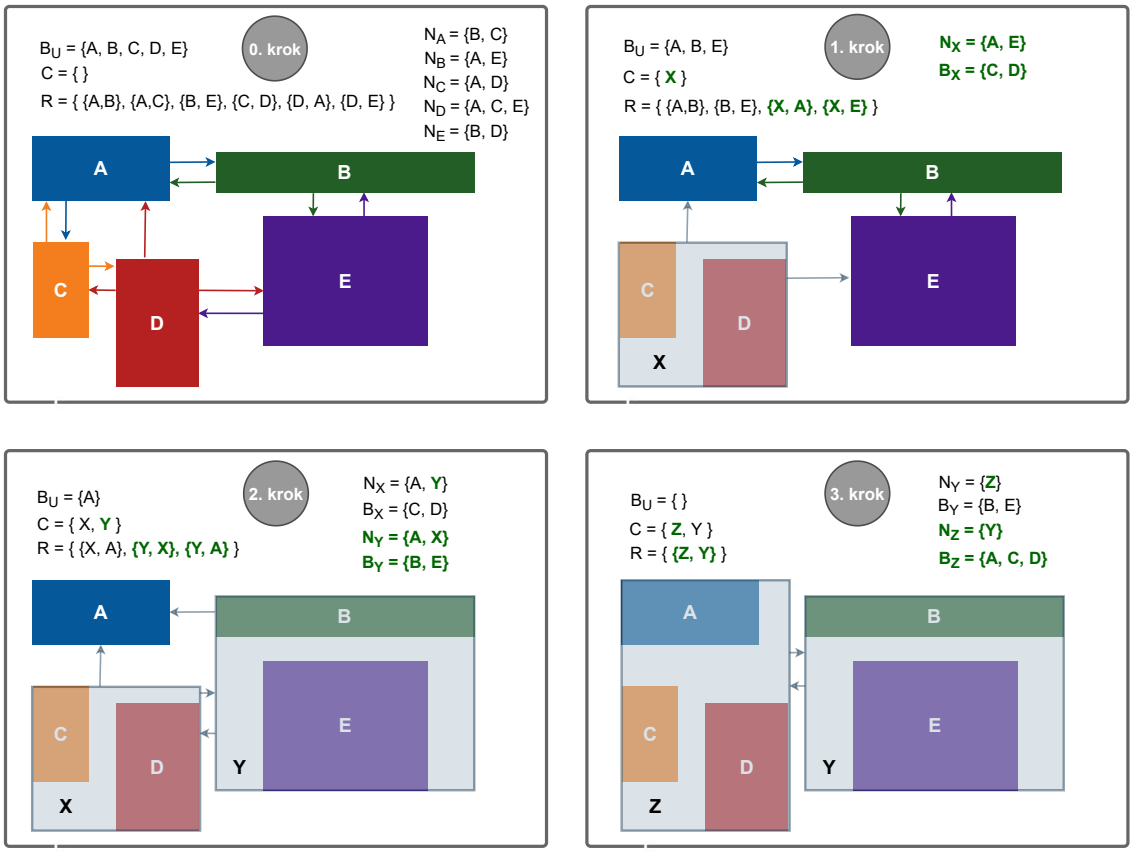
Na úplnom začiatku procesu zhlukovania sú všetky boxy, ktoré nepatria do žiadneho zhluku uložené v množine B_U. Množina zhlukov C je prázdna. Zavedená je nová množina vzťahov R, ktorá obsahuje všetky vzťahy existujúce medzi jednotlivými boxmi a ich priamymi susedmi.

Predpokladajme, že dôjde k zlúčeniu dvoch boxov, ktoré sú najviac podobné na základe podobnosti špecifikovanej modelom podobnosti definovanom v časti 3.4. Ďalej predpokladajme, že nové zhlukové zrno bude prijaté ako validný zhluk. V takom prípade musí byť vzťah na základe, ktorého boli boxy spojené odstránený z množiny vzťahov, aby sa v nasledujúcom cykle znovu neuvažoval. Následne je nutné prepočítať všetky množiny, ktoré mohli byť ovplyvnené vytvorením nového zhluku:

1. Boxy, ktoré tvoria nový zhluk sú odstránené z množiny B_U,
2. Všetky ostatné vzťahy boxov s ich priamymi susedmi sú odstránené z množiny R
3. Novému zhluku sú priradení priami susedia získaní z množín priamych susedov boxov, ktoré ho tvoria a odpovedajúce vzťahy (s prepočítanou podobnosťou na základe kardinality a kumulatívnej podobnosti) sú pridané do množiny R
4. Boxy sa stávajú členmi novo vytvoreného zhluku
5. Nový zhluk je pridaný do výslednej množiny zhlukov C

Proces vytvárania zhlukov, prepočítavania množiny vzťahov a priamych susedov tiež reprezentuje nasledujúca séria obrázkov 5.7, ktorá je uvedená pre ľahšie pochopenie celého konceptu. Šípky vychádzajúce z boxu znázorňujú jeho priamych susedov. Na začiatku (v 0. kroku) sú všetky boxy v množine B_U a množina R obsahuje všetky vzťahy, ktoré sú medzi dvojicou boxov unikátne identifikované (z toho dôvodu sú položky v R znázornené ako množiny). Množina N_A reprezentuje množinu priamych susedov boxu A. Môžeme si všimnúť, že vzťah medzi boxmi A a D je jednosmerný, pretože A má smerom nadol najbližšieho suseda iba box C.

V prvom kroku zhlukovania dôjde k zlúčeniu boxov C a D a vytvoreniu zhluku X. Vzťah medzi boxmi je automaticky odstránený z množiny R. Odstránené sú tiež všetky ostatné vzťahy boxov C a D so zvyšnými susedmi. Zhluku X sú priradené vzťahy k boxom A a E, pretože boli priamymi susedmi aspoň jedného boxu, ktorý patrí do vytvoreného zhluku. V druhom kroku dôjde k vytvoreniu zhluku Y a následnému prepočítaniu vzťahov a priamych susedov podobne ako v predchádzajúcom kroku. Množina B_X reprezentuje množinu boxov, ktoré patria do zhluku X. V poslednom kroku dôjde k zlúčeniu existujúceho zhluku X a boxu A, čím vznikne nový zhluk Z. Pôvodný zhluk X je z množiny zhlukov, vzťahov a množín priamych susedov odstránený.



Obr. 5.7: Proces zhlukovania na ukážkovom príklade uvedený v 4 krokoch. Označenie: B_U množina boxov, ktoré nie sú v žiadnom zhluku; C množina zhlukov; R množina vzťahov; N_A množina priamych susedov boxu A; B_X množina boxov, ktoré patria do zhluku X.

5.5 Vizualizácia

Na kontrolu správnosti a vizuálne porovnanie výsledkov segmentácie je vhodné výsledné zhluky/segmenty vizualizovať a exportovať do súborov. Väčšina frameworkov na automatizáciu prehliadača sprostredkuje možnosť vytvoriť *screenshot* spracovávanej webovej stránky. Dôležité je si tiež priblížiť vizuálnu reprezentáciu množiny extrahovaných boxov po odstránení tých, ktoré sa prekrývajú. To sprostredkuje väčší prehľad o tom s akými boxmi zhlukovanie pracuje a prípadnú identifikáciu problémov v situácii, kedy segmentácia konkrétnej stránky nedopadne podľa očakávaní.

Kapitola 6

Implementácia

V tejto kapitole je rozobraná implementácia metódy BCS s využitím automatizácie prehliadača. Na základe návrhu predstaveného v predchádzajúcej kapitole bola vytvorená aplikácia pozostávajúca z viacerých modulov, ktoré sa líšia vymedzenou funkcionalitou.

Na začiatku kapitoly sú vymenované jednotlivé použité technológie na klientskej i serverovej strane aplikácie. Klientskou stranou sa v tomto kontexte myslí kód, ktorý je vykonávaný v kontexte automatizovaného prehliadača. Následne je popísaný proces extrakcie boxov z webovej stránky, na ktorý nadväzuje najdôležitejšia časť implementácie, ktorou je proces zhlukovania. V poslednom rade sú opísané moduly zodpovedné za vizualizáciu segmentačných krokov a exportovanie dát do súborov.

6.1 Výber technológií

Jedným z najhlavnejších krokov, ktorému predchádza samotná implementácia je výber vhodných technológií, ktoré musia odpovedať účelu a využitiu aplikácie, a ktoré podporujú všetky požiadavky uvedené v návrhu.

Serverovú časť aplikácie sme sa rozhodli implementovať s využitím asynchrónneho behového prostredia **Node.js**. Jeho použitím je umožnené pracovať vo všetkých častiach systému s rovnakým programovacím jazykom, ktorým je **JavaScript**. Ide o spomínanú paradigmu *JavaScript everywhere*. Na základe toho je sprostredkovaná jednotná organizácia súborov a takmer identická syntax pre kód vykonávaný v serverovej a klientskej časti. To prináša výhody ako v prehľadnosti a znovupoužití kódu, tak aj v zabezpečení jednotného rozhrania pre prácu s objektmi a dátovými štruktúrami.

Ako framework na automatizáciu webového prehliadača bol zvolený **Playwright**, ktorý z možných automatizačných frameworkov v prostredí Node.js, diskutovaných v kapitole 4.3 predstavuje jednu z najideálnejších možností. Playwright je rýchlym, stabilným a spoľahlivým frameworkom, ktorý v komunite vývojárov začína naberať na popularite. Výhodou je aj podrobná dokumentácia oficiálnej stránke spolu s príkladmi. Keďže pri extrakcii nás zaujíma predovšetkým rýchlosť spracovania a získania boxov spolu s Puppeteerom predstavujú obidve možnosti vhodných kandidátov. Pôvodne sme na automatizované spustenie prehliadača používali práve Puppeteer, ale ustúpili sme od neho z dôvodu, že na stránkach s väčšou výškou ako $8192px$ dochádzalo k interným problémom týkajúcich sa získania screenshotu stránky. Vzhľadom na to, že pre proces segmentácie nie je kľúčové použitie a testovanie aplikácie naprieč rôznymi webovými prehliadačmi, nie je ich podpora explicitne vyžadovaná. Na účel extrakcie si preto vystačíme s využitím integrovaného prehliadača **Chromium**,

ktorý je spolu s Google Chrome tiež z jedným najrýchlejších, čo sa týka vykresľovania stránok. Pri použití Playwrightu je ale veľmi jednoduché v kóde zmeniť prehliadač na iný podporovaný, napr. Chrome alebo Firefox. Okrem toho Playwright umožňuje jednoduchým spôsobom ukladať screenshoty danej stránky, čo môže byť využiteľné pri ladení aplikácie a znázorňovaní vizuálnej reprezentácie extrahovaných boxov a vytvorených zhlukov.

6.2 Automatizácia prehliadača pomocou Playwright

Celý proces automatizácie prehliadača od spustenia jeho inštancie a vykonania všetkých príkazov, ktoré sú potrebné pre extrahovanie boxov až po jeho uzavretie, je pri použití Playwrightu relatívne jednoduché, čo je znázornené na nasledujúcej ukážke kódu 6.1.

Všetky volania API funkcií Playwrightu sú asynchrónne a preto ako návratovú hodnotu vracajú objekty typu `Promise`. Kvôli lepšej čitateľnosti kódu je využitý vzor `async/await` a z toho dôvodu je zároveň nutné všetky volania funkcií vykonať v hlavnom `async` bloku ako je uvedené v ukážke.

```
1  const { chromium } = require('playwright');
2
3  (async () => {
4    /* Vytvorenie inštancie prehliadača */
5    const browser = await chromium.launch();
6
7    /* Otvorenie novej stránky */
8    const page = await browser.newPage();
9
10   /* Nastavenie veľkosti okna prehliadača špecifikovanou argumentmi W a~H */
11   page.setViewportSize({ width: argv.W, height: argv.H });
12
13   /* Otvorenie stránky na základe URL danej argumentom url */
14   await page.goto(argv.url);
15
16   /* Pridanie externého JavaScriptu na danú stránku zo súborového systému */
17   await page.addScriptTag({ path: './modules/box-extraction.js'});
18
19   /* Spustenie kódu v~kontexte prehliadača a~vrátenie výsledku do premennej */
20   const extracted = await page.evaluate(async () => {
21         /* Extrakcia boxov */
22         return await extractBoxes(document.body);
23   });
24
25   /* Uloženie screenshotu stránky vo formáte PNG */
26   await page.screenshot({ path: '/out/screenshot.png', fullPage: true });
27
28   /* Zavretie inštancie prehliadača */
29   await browser.close();
30
31   /* Spustenie procesu segmentácie s~boxmi serializovanými v~premennej 'extracted' */
32   createSegmentation(extracted);
33 })();
```

Výpis 6.1: Ukážka kódu na spustenie inštancie prehliadača Chromium s využitím Playwrightu. Zoznam boxov vrátených z funkcie `page.evaluate` na riadku 20, ktorá zabezpečuje extrakciu boxov v kontexte prehliadača, je dostupný len v hlavnom `async` bloku, preto je v ňom nutné zároveň spustiť proces segmentácie s extrahovanými boxmi (riadok 32).

6.3 Extrakcia boxov

Na základe požiadavky uvedenej v návrhu bolo cieľom implementovať extrakciu čo najefektívnejšie. Boli odskúšané viaceré štruktúry optimalizované na pre-order prechod celého DOM stromu (`TreeWalker, NodeIterator`), ale pri testovaní celkovej doby trvania sa ukázalo, že jednoduchý rekurzívny pre-order prechod stromu je najrýchlejší. Dané štruktúry zavádzajú pri svojej inicializácii dodatočnú réžiu, ktorá má natoľko negatívny dopad, že sa ich použitie neoplatí. Ďalším dôvodom prečo neboli použité je aj skutočnosť, že prechod stromu potrebujeme uskutočniť iba jediný raz.

Proces extrakcie v kontexte prehliadača začína zavolaním funkcie `extractBoxes(root)` ako je znázornené v ukážke kódu 6.1 na riadku 22. Parameter funkcie `root` predstavuje koreňový uzol stromu, od ktorého sa extrahujú všetky relevantné boxy na základe špecifikovaných pravidiel uvedených v kapitole 3.2. Box ako dátová štruktúra definovaná v návrhu je v implementácii definovaná ako trieda `BoxInfo`.

6.3.1 Získanie súradníc

Pozícia daného DOM uzlu (*node*) v rámci stránky, resp. súradnice jeho ľavého horného a pravého dolného rohu je získaná pomocou štandardnej funkcie `node.getBoundingClientRect()`, ktorá vracia minimálny ohraničujúci obdĺžnik (*minimum bounding box*) daného uzla. Na získanie súradníc obrázkových uzlov a uzlov s jedným potomkom bez vetiev s pozadím sa využíva práve táto funkcia.

V prípade textových uzlov je nutné počítať so situáciou, kedy dôjde k zalomeniu jeho textového obsahu (slov) kvôli obmedzenej šírke okna. Preto je na získané pole minimálnych ohraničujúcich obdĺžnikov pomocou funkcie `getClientRects()`, ktorá je zovšeobecnením predchádzajúcej funkcie. Pre každú položku poľa je potom vytvorený samostatný box, čo znamená, že jednému textovému uzlu môže odpovedať množina boxov, pochopiteľne s rovnakou farbou a rôznou pozíciou.

Z hľadiska extrakcie je dôležité zahrnúť do výslednej množiny iba tie boxy, ktoré sú skutočne viditeľné. Na kontrolu, či niektorá zo súradníc *bounding boxu* neleží mimo *viewportu* slúži funkcia `isInViewport(node)`. V prípade, že funkcia vráti hodnotu `false`, box nie je pridaný do výslednej množiny extrahovaných boxov.

6.3.2 Získanie farby

Reprezentatívna farba uzlu sa získava z prepočítaného štýlu prostredníctvom funkcie `window.getComputedStyle(node)` po vytvorení vykresľovacieho stromu a vykreslení stránky. Farba textového uzlu je prevzatá z jeho rodičovského uzlu. Farba uzlu s jedným potomkom s jednofarebným pozadím je určená hodnotou CSS vlasnosti `background-color`.

Na získanie priemernej farby obrázkov, ktoré patria k obrázkovým uzlov a uzlom s obrázkom na pozadí je použitý externý modul `FastAverageColor`[1]. Ten umožňuje rýchle synchrónne vypočítanie priemernej farby z obrázka, ktorý je na stránke vykreslený a jeho dáta sú k pre skript k dispozícii. V prípade, že dáta nie sú k dispozícii, je nutné farbu vypočítať z URL obrázka asynchrónne funkciou `getColorAsync(imgUrl)`, čo výrazne ovplyvňuje efektivitu extrakcie (negatívne). Ďalší problém pri získavaní priemernej farby obrázka ešte môže spôsobiť politika *CORS*. V tom prípade je použitá predvolená farba.

[1] https://www.npmjs.com/package/fast-average-color

6.3.3 Najmenší box z uzlu s jedným potomkom

Na určenie validného boxu, ktorý je získaný z uzlu s jedným potomkom bez vetiev je použitá skupina funkcií `hasNoBranches, getSmallest, getParentWithBackground` a `getLastChild`. Logika získania daného boxu odpovedá popisu pravidiel uvedených v kapitole 3.2.

6.4 Zhlukovanie

Pred samotným spustením procesu zhlukovania je potrebné vytvoriť triedu (`ClusteringManager`, ozn. `CM`), ktorá bude riadiť zhlukovanie a uchovávať všetky meniace sa množiny dát v priebehu zhlukovania. Vytvorenie `CM` je v ukážke kódu 6.2 na riadku 3. V konštruktore `CM` v prvom rade prebieha reinicializácia všetkých extrahovaných boxov, ktoré boli serializované, ako objektov triedy `Box`, ktorá v porovnaní so štruktúrou `BoxInfo` naviac obsahuje:

1. mapu susedov (`neighbours`), ktorej kľúčom je susedný box (referencia na objekt `BoxInfo`) a hodnotou vzťah, ktorý medzi boxmi existuje (referencia na objekt `Relation`),
2. maximálnu susedskú vzdialenosť (`maxNeighbourDistance`) a
3. metódu `findDirectNeighbours` na výpočet priamych susedov boxu.

Znovu inicializované boxy sú následne vložené do R-stromu pre ďalšie spracovanie. Implementáciu R-stromu, ktorú sme sa rozhodli použiť je tzv. `RBush`[2], pričom ide vlastne o R*-strom (varianta R-stromu), ktorý je vysoko optimalizovaný a efektívny (podľa autorov). Následne sú boxy skontrolované na prekrytie a prekrývajúce sa boxy a kontajnery (boxy, ktoré vizuálne obsahujú iné boxy) sú odstránené práve s využitím inicializovaného R-stromu prostredníctvom funkcie `removeContainers`. Je dôležité poznamenať, že pôvodný `RBush` pracuje so súradnicovým systémom, ktorého počiatok (0,0) je v ľavom dolnom rohu. Z toho dôvodu sme museli vytvoriť novú triedu pre R-strom (trieda `RTree` rozširujúca `RBush`), ktorá pracuje s rovnakým súradnicovým systémom, aký je použitý v prehliadači.

```
1  function createSegmentation(extracted) {
2      /* Vytvorenie triedy, ktorá riadi zhlukovanie */
3      var cm = new ClusteringManager(extracted);
4
5      /* Nájdenie priamych susedov (a vzťahov) medzi všetkými boxmi */
6      cm.findAllRelations();
7
8      /* Vytvorenie zhlukov */
9      cm.createClusters();
10
11     /* Získanie dát pre export */
12     var dataForExport = cm.getDataForExport();
13
14     /* Export */
15     exportFiles(dataForExport);
16 }
```

Výpis 6.2: Zjednodušená funkcia `createSegmentation`, ktorá je invokovaná z hlavného bloku programu v ukážke kódu 6.1 na riadku 32.

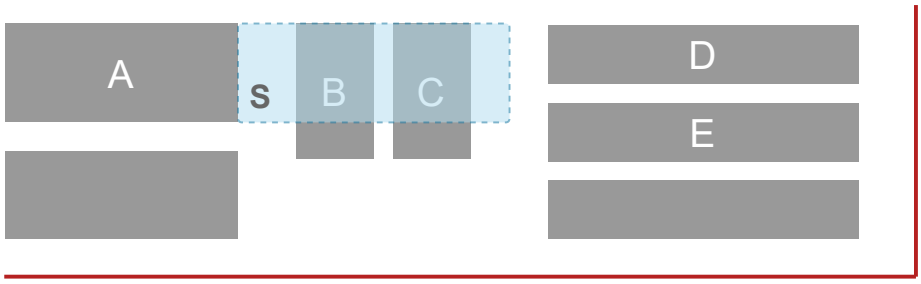
[2] https://www.npmjs.com/package/rbush

6.4.1 Nájdenie priamych susedov

Po odstránení prekrývajúcich sa boxov sú pomocou R-stromu vyhľadaní priami susedia všetkých boxov nachádzajúci od daného boxu smerom vpravo, vľavo, hore a dole. Zároveň sú vytvorené vzťahy (objekty triedy `Relation`), ktoré patria k danému susedovi.

Optimalizácia vyhľadania susedov

Už samotné navrhnuté riešenie pre vyhľadanie priamych susedov predstavuje v porovnaní s naivným prístupom istú optimalizáciu. Vyhľadanie priamych susedov je ale možné ďalej optimalizovať za predpokladu, že dotazy do R-stromu sú veľmi rýchle a hustota boxov na stránke je vysoká. Rozhodli sme sa teda riešenie uvedené v kapitole 5.4.2 ďalej optimalizovať. Na základe experimentov sme zistili, že priami susedia boxu sa v danom smere nachádzajú v jeho tesnej blízkosti. Je teda zbytočné získavať všetky boxy v danom smere a počítať medzi nimi absolútnu vzdialenosť (na základe ktorej sa určí skutočný priamy sused). Optimalizácia teda spočíva v konštantnom zväčšovaní šírky (alebo výšky) selektoru a simultánnom vyhľadávaní boxov v R-strome pod selektorom, až kým šírka (výška) selektoru nepresiahne maximálnu šírku (výšku) okna. Túto optimalizáciu lepšie ilustruje nasledujúci obrázok 6.1.



Obr. 6.1: Optimalizácia vyhľadania priamych susedov boxu A pomocou selektoru S s výškou boxu A a obmedzenou šírkou. Pod selektorom boli napravo od boxu A vyhľadané boxy B a C. Je zrejmé, že niektorý z nich bude priamym susedom A. Boxy D a E selektorom neboli vyhľadané a preto medzi nimi a boxom A nie je počítaná absolútna vzdialenosť ani základná podobnosť (časovo náročnejšia). Vyhľadanie priamych susedov napravo od boxu A tým končí.

Optimalizácia – unikátne identifikované vzťahy

Ďalšou zavedenou optimalizáciou je výpočet podobnosti iba jedného zo vzťahov, ak ide o obojsmerný vzťah medzi boxmi (vzájomní priami susedia). Optimalizácia spočíva vo vytvorení identifikátoru vzťahu, ktorý je vytvorený konkatenáciou identifikátorov boxov (napr. A a B) na základe jeho smeru. Pre úplnosť je dôležité uviesť, že ID boxu je reťazec v nasledujúcom formáte `(t:val,l:val,b:val,r:val,c:rgb)`, určený už v procese extrakcie. Ak je daný vzťah smerom vpravo alebo nadol, ID je definované ako $ID_A + ID_B$. V opačnom prípade to je $ID_B + ID_A$. Ak sú boxy A a B vedľa seba na osi x a sú si navzájom priamymi susedmi, ich vzťah je potom unikátne identifikovaný. Výsledná podobnosť všetkých identifikovaných vzťahov medzi boxmi sa počíta až po určení všetkých priamych susedov všetkých boxov vo funkcii `findAllRelations` (v 6.2 na riadku 6). V niektorých prípadoch došlo k redukcii počtu výpočtov základnej podobnosti (*base similarity*) až o 50%.

6.4.2 Proces vytvárania zhlukov

Po vypočítaní všetkých vzťahov a podobností medzi boxmi je možné začať boxy zhlukovať. Hlavný cyklus procesu zhlukovania, ktorý odpovedá zhlukovaciemu algoritmu BCS 1 v zjednodušenej verzii vyzerá nasledovne:

```
1   /* Cyklus cez všetky platné vzťahy */
2   while(relations.size > 0) {
3
4       /* Nájdi dvojicu najviac podobných entít na zlúčenie */
5       var bestRel = getBestRelation();
6
7       /* Odstráň vzťah z množiny vzťahov */
8       relations.delete(bestRel.id);
9
10      /* Kontrola zhlukovacieho prahu */
11      if(bestRel.similarity > CT) break;
12
13      /* Vytvorenie nového kandidáta na zhluk */
14      var clusterCandidate = new Cluster(bestRel.entityA, bestRel.entityB);
15
16      /* Kontrola na prekrytie kandidáta s inými entitami */
17      if(overlaps(clusterCandidate, bestRel)) continue;
18
19      /* Aktualizácia množín zhlukov, boxov a vzťahov  */
20      removeEntities(clusterCandidate);
21      updateRelations(clusterCandidate);
22
23      /* Potvrdenie zhluku a pridanie do množiny zhlukov */
24      clusters.set(clusterCandidate.id, clusterCandidate);
25  }
```

Výpis 6.3: Zjednodušený hlavný cyklus zhlukovacieho procesu, ktorý je súčasťou funkcie `createClusters`.

Na začiatku cyklu je pomocou funkcie `getBestRelation` získaný vzťah z množiny vzťahov, ktorý predstavuje najlepšiu (najviac podobnú) dvojicu na zlúčenie na základe podobnosti. Vzťah je v zápätí odstránený z množiny vzťahov. Následne prebieha kontrola, či hodnota podobnosti nepresiahla zhlukovací prah CT. Ak áno, proces zhlukovania končí. Z entít, ktoré sú súčasťou vzťahu je vytvorený nový kandidátny zhluk. Ak neprekrýva iný zhluk môže byť potvrdený. Potrebná je ešte kontrola na prekrytie s niektorým boxom. Ak kandidátny zhluk prekrýva iba boxy, pokúsime sa ich pridať do kandidátneho zhluku. Ak sa následne kandidátny zhluk neprekrýva s nijakou entitou, môže byť skutočne potvrdený ako platný zhluk. Kontrolu prekrytia v ukážke 6.3 reprezentuje funkcia `overlaps` na riadku 17.

Samotnému potvrdeniu zhluku a jeho pridaniu do množiny zhlukov, predchádza odstránenie všetkých entít (boxov a zhlukov), ktoré sú súčasťou nového zhluku. Rovnako je potrebné odstrániť všetky vzťahy, ktorých boli entity súčasťou a nutné je tiež pridať nové vzťahy medzi existujúcimi zhlukmi a novo vytvoreným zhlukom. Tento prepočet vzťahov a priamych susedov zabezpečuje funkcia `updateRelations` z kontextu `CM` spolu s funkciou `updateAllNeighbours` zavolanou nad kandidátnym zhlukom. Je tiež veľmi dôležité spomenúť fakt, že množiny priamych susedov boxov sa v priebehu zhlukovania **nemenia**, z množiny vzťahov sa odstraňujú len ich referencie. Referencia na vzťah medzi boxom a jeho susedom ostáva nezmenená, aby bolo stále možné dopočítať skutočnú hodnotu zhlukovej podobnosti pomocou funkcie `calcClusterSimilarity` (definovanej v triede `Relation`) me-

dzi novo vytváranými zhlukmi v priebehu zhlukovania. Pre úplnosť uvedieme, že hodnota vypočítaná funkciou `calcClusterSimilarity` je vyjadrená ako podiel vypočítanej kumulatívnej podobnosti (`calcCumulSimilarity`) a kardinality (`calcCardinality`).

6.5 Spracovanie vstupných argumentov

Základné vstupné argumenty aplikácie `bcs.js`, ktoré sa týkajú priamo procesu extrakcie a zhlukovania predstavujú nasledujúce argumenty (v kóde programovo spracované pomocou balíčka `yargs`):

```
<URL>                          URL stránky, ktorá bude segmentovaná
--CT, --clustering-treshold    Zhlukovací prah
--W, --width                   Šírka okna prehliadača
--H, --height                  Výška okna prehliadača
--IGIM, --ignore-images        Extrakcia bez výpočtu priemernej farby
...
```

Najdôležitejším (a jediným povinným) argumentom programu je špecifikovaná URL adresa stránky, ktorej segmentáciu chceme získať (`<URL>`). Ďalším dôležitým argumentom pre segmentáciu je hodnota zhlukovacieho prahu (`--CT`), ktorá musí byť z intervalu $(0, 1\rangle$. Predvolená hodnota `CT` je nastavená na hodnotu 0.5.

Argument `--IGIM` je významný práve z hľadiska extrakcie a jej rýchlosti, pretože umožňuje zamedziť výpočtu priemernej farby obrázkov. V prípade ak je tento argument špecifikovaný, pre všetky obrázky sa použije predvolená farba `rgb(128,128,128)` a priemerná farba sa vôbec nepočíta, čo výrazne urýchľuje proces extrakcie. Zároveň to môže istým spôsobom ovplyvniť výslednú segmentáciu, ale pri experimentoch sa ukázalo, že farebná zložka nemá na podobnosť (a výslednú segmentáciu) významný vplyv a dôležitá je predovšetkým relatívna vzdialenosť.

Pomocou argumentov `-W` a `-H` je možné špecifikovať šírku a výšku okna spusteného prehliadača (*viewportu*). Hodnoty, môžu ovplyvniť extrakciu aj výslednú segmentáciu vzhľadom na rýchlosť i presnosť, čo ukazuje aj nasledujúci obrázok 6.2.

Obr. 6.2: V závislosti od šírky *viewportu* môže dôjsť napr. k skrytiu elementov na stránke. Na príklade stránky so šírkou *viewportu* 1200px by nedošlo extrakcii textových uzlov v navigácii, pretože nie sú viditeľné. Segmentácie stránky s rôznou šírkou by teda boli odlišné.

V rámci implementácie sme sa rozhodli vytvoriť dve rôzne implementácie segmentácie. Prvá, ktorú označujeme ako základná (`basic`) bola vytvorená tak, aby čo najviac zodpovedala teoretickému popisu metódy BCS (najmä čo sa týka zhlukovania) popísanej v kapitole 3. V rozšírenej implementácii sa vo výpočte podobnosti naviac zohľadňuje tzv. skóre priliehavosti (`alignmentScore`). Princíp jeho výpočtu a použitia bol zistený analýzou kódu a prevzatý z referenčnej implementácie[3]. Jedným z hlavných rozdielov pri vytváraní zhlukov je, že v rozšírenej implementácii je povolené vytvorenie zhluku v špeciálnom prípade prekrytia, kedy kandidátny zhluk vizuálne obsahuje iný zhluk. V základnej implementácii je to považované za klasické prekrytie zhluku a kandidátny zhluk je v takom prípade zahodený. Na kontrolu prekrytia pri vytváraní kandidátneho zhluku slúži funkcia `overlaps`.

6.5.1 Základná implementácia

V základnej implementácii je možné špecifikovať len základné argumenty, z ktorých najdôležitejšie boli popísané na začiatku tejto kapitoly. Okrem už spomenutých argumentov je v základnej implementácii možné použiť: výpis ladiacich informácií (`-D`), výpis informácií spracovania (`-I`), zobrazenie nápovedy (`-h`).

6.5.2 Rozšírená implementácia

Pri použití rozšírenej implementácie (`--extended`) je možné použiť dodatočné argumenty, ktoré oproti základnej implementácii umožňujú: nastaviteľnosť váhového vektoru vo výpočte podobnosti pre jednotlivé zložky podobnosti (`--WVEC`), agresívne zhlukovanie – pridanie všetkých prekrývajúcich zhlukov do kandidátneho zhluku (`-A`), zamedzenie vytvorenia zhluku na základe nedostatočnej hustoty pokrytia boxmi (`--DT`). Všetky dodatočné argumenty sú **experimentálne** a môžu viesť k neočakávaným výsledkom segmentácie. V niektorých prípadoch a pri špecifickom nastavení hodnôt umožňujú však zlepšiť presnosť segmentácie.

6.6 Vizualizácia segmentačného kroku

V rámci implementácie bola tiež vytvorená vizualizácia kroku segmentácie (modul `box-vizualizer`), ktorá veľmi podrobným spôsobom znázorňuje ako sa vyvíja proces vytvárania zhlukov. Krok segmentácie chápeme ako jednu iteráciu cyklu v ukážke kódu 6.3. Vizualizácia je interaktívna a prebieha v inštancii prehliadača otvorenej pomocou Playwrightu. Samotnej vizualizácii predchádza získanie dát daného segmentačného kroku z kontextu `CM` prostredníctvom funkcie `getDataForVizualization`. Boxy a zhluky sú vo vizualizácii reprezentované ako `<div>` elementy.

V prípade ak chceme krok segmentácie vizualizovať, aplikácia musí byť spustená s argumentom `--VS N`, kde `N` ako celé číslo reprezentuje poradie (iteráciu) požadovaného segmentačného kroku. Ak `N` presiahne skutočný počet iterácií, vizualizovaný je konečný výsledok segmentácie. Keďže je vizualizácia interaktívna, sprostredkuje možnosť skúmať a kontrolovať správnosť prepočítavania vzťahov, podobností a vytváraných zhlukov prechádzaním myšou nad boxmi alebo zhlukmi. Vizualizácia kroku je spustená funkciou `vizualizeStep`, pričom po zavretí prehliadača je proces segmentácie prerušený.

Na sérii obrázkov 6.3 je znázornená vizualizácia 3. kroku segmentácie vytvorenej jednoduchej stránky (`5greydivs.html`) pre názornú ukážku. Na obrázku ① je pôvodná stránka

[3] https://github.com/janzeleny/bcs/

vykreslená v prehliadači. Obrázok ② znázorňuje vizualizáciu 3. segmentačného kroku stránky ihneď po otvorení prehliadača. Farba pozadia zhlukov je priehľadná na 50% kvôli viditeľnosti boxov pod nimi. Potvrdené zhluky, ktoré sa nachádzajú v množine zhlukov sú znázornené svetlomodrou farbou. Entity, ktoré sú súčasťou najlepšieho vybraného vzťahu v danej iterácií, sú označené červenou farbou. Prechodom myši nad niektorý z boxov (zhlukov) sa farba boxu (zhluku) zmení na žltú a jeho priami susedia sú označení ružovou farbou. Zároveň je na susedovi v ľavom hornom rohu zobrazená vypočítaná hodnota podobnosti a platnosť vzťahu vyjadrená hodnotou true alebo false (teda či sa daný vzťah ešte nachádza v množine vzťahov R). Uvedenému popisu odpovedajú obrázky ③, ④ a ⑤. Na obrázku ⑤ si môžeme všimnúť, že obidva vzťahy sú neplatné (false), pretože priami susedia boxu sa už nachádzajú v zhlukoch. Vzťah je preto platný iba v smere od daných zhlukov (③, ④). Na poslednom obrázku ⑥ je znázornený výsledok segmentácie pri použití `--CT` 0.4.

Obr. 6.3: Séria obrázkov znázorňujúca vizualizáciu 3. segmentačného kroku na jednoduchej stránke, ktorá obsahuje 5 `<div>` elementov so sivým pozadím. Popis jednotlivých obrázkov je uvedený v odseku nad obrázkom kvôli prehľadnosti.

6.7 Exportovanie dát

Aby mohla byť presnosť a kvalita vytvorenej segmentácie danej stránky vizuálne (subjektívne) a objektívne (na základe metrík) zhodnotená, je dôležité, aby ju bolo možné exportovať do súboru. Rovnako dôležité je aj zobrazenie množiny extrahovaných boxov, ktoré sú vstupom pre zhlukovanie. Je to z dôvodu, že výsledná segmentácia a jej presnosť úzko súvisí s kvalitou a podrobnosťou extrakcie. Na objektívne porovnanie výsledkov segmentácií pri použití rôznych zhlukovacích prahov je nutné vytvorené zhluky (segmenty) a extrahované boxy exportovať do súborov vo formáte `JSON`. Následne je s nimi možné ďalej pracovať.

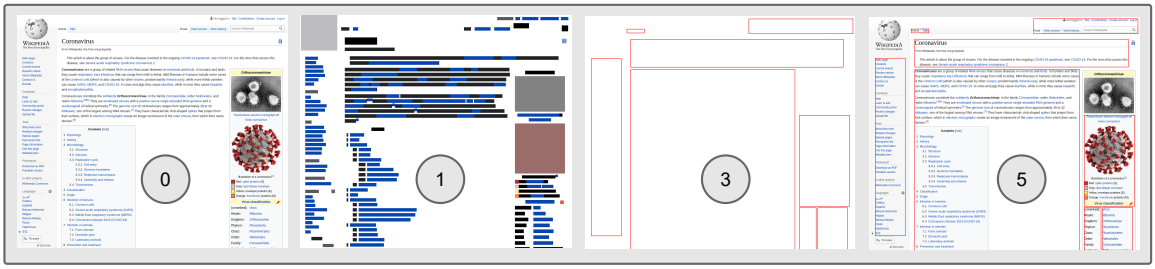
Ak chceme exportovať dáta do súboru, aplikáciu spustíme s argumentom `-E` nasledovaným reťazcom obsahujúcich celé čísla (1 až 8), ktoré odpovedajú jednotlivým možnostiam, resp. konkrétnym požadovaným súborom v danom formáte. Možnosti pre export sú nasledujúce (príklad exportovaných súborov je znázornený na obrázku 6.4):

```
0 - žiadny export
1 - export boxov vo formáte PNG
2 - export boxov vo formáte JSON
3 - export zhlukov vo formáte PNG
4 - export zhlukov vo formáte JSON
5 - export zhlukov zobrazených nad screenshotom stránky vo formáte PNG
6 - export všetkých možností 1 až 5
7 - export všetkých segmentačných krokov vo formáte PNG
8 - export všetkých DOM uzlov ako boxov vo formáte JSON
```

Príkladom použitia argumentu `-E` môže byť napr. `-E 134`, `-E 6`, `-E 6111`, `-E 7`. Spolu s argumentom `-E` môže byť špecifikovaná cesta k výstupnému priečinku (`-O`) a v názve exportovaných súborov môže byť zahrnutá URL adresa segmentovanej stránky (`--IURL`). Pre úplnosť spomenieme, že uloženie screenshotu webovej stránky je automatické. Ak screenshot uložiť nechceme, program spustíme s argumentom `-S false`.

Pre možnosti exportu sú definované nasledujúce pravidlá: **1.** Ak reťazec čísel obsahuje 0, neexportuje sa žiadny súbor. **2.** Ak reťazec čísel obsahuje 6, exportujú sa súbory odpovedajúce možnostiam 1 až 5. **3.** Export všetkých segmentačných krokov (7) môže byť použitý iba výlučne / samostatne (kvôli časovej náročnosti).

Na exportovanie dát do súborov bol vytvorený modul `exporter`, ktorý obsahuje všetky relevantné funkcie pre export dát vo formátoch `JSON` a `PNG`. Na riadku 15 v úseku kódu 6.2 je volanie funkcie `exportFiles` práve zo spomínaného modulu.



Obr. 6.4: Príklad exportovaných súborov (screenshotov) vo formáte `PNG`, ktoré číselne odpovedajú možnostiam pre export (1,3,5). Prvý obrázok s číslom 0 reprezentuje screenshot pôvodnej stránky.

Kapitola 7

Vyhodnotenie a výsledky

V tejto kapitole sa zaoberáme vyhodnotením presnosti a kvality segmentácie vytvorenej prostredníctvom implementácie BCS popísanej v predchádzajúcej kapitole na množine webových stránok (dataset). Samotné vyhodnotenie sme vytvorili na základe objektívnych metrík, ktoré presným a dôverným spôsobom zachytávajú podobnosť vytvorených zhlukov (segmentov) – rôznych segmentácií stránky. Na začiatku kapitoly je popísaná vytvorená dátová sada stránok. Súčasťou kapitoly je aj popis vytvorenej jednoduchej aplikácie na anotáciu a porovnanie segmentácií vzhľadom na očakávanú segmentáciu (*ground truth*). Najdôležitejšou časťou kapitoly je porovnanie segmentácií s referenčnou implementáciou na základe spomínaných metrík. V závere kapitoly sú diskutované vplyvy parametrov na segmentáciu, obmedzenia a limity vytvoreného riešenia a možnosti na zlepšenie.

7.1 Dátová sada

Výber vhodnej dátovej sady webových stránok predstavoval dôležitý krok, ktorý predchádzal samotnému procesu vyhodnotenia a porovnania. Na internete existuje nespočetné množstvo stránok, takže výber reprezentatívnej množiny stránok nebol jednoduchou úlohou. Pri vytváraní datasetu sme sa snažili vybrať reprezentatívnu množinu stránok s rôznou zložitosťou v závislosti od usporiadania a typov elementov na stránke, obsahu, štrukturálnej zložitosti HTML kódu a celkovej komplexnosti vizuálnej štruktúry vykreslenej stránky.

Vytvorený dataset (`dataset.html`), ktorý je súčasťou odovzdaného archívu obsahuje spolu 30 webových stránok získaných z rôznych webových lokalít. Z hľadiska štruktúry a zamerania ich môžeme rozdeliť (približne) do nasledujúcich kategórií:

- Eshopy (4): booking.com, datacomp.sk, datart.sk, gymbeam.sk
- Wikistránky (4): citizendium.org, infoplease.com, wikiskripta.eu, wikipedia.org
- Články (8): bardejov.sk, detska-rec.sk, korona.gov.sk, logo-centrum.sk, old.korona.gov.sk, sal.sk, slovensko.sk
- Iné (14): bistro.sk, bratislava.sk, brno.cz, cssbox.sf.net, cylex.sk, farmabusovgaboltov.sk, fit.vut.cz, karatebardejov.webnode.sk, lubotin.sk, netflix.com, vutbr.cz

Kategórie niektorých stránok sa môžu prelínať alebo stránka nemusí patriť ani do jednej z uvedených. Práve tie stránky sú priradené do kategórie **Iné**. U jednotlivých kategórií je uvedený celkový počet stránok, pričom v zozname sú uvedené len webové lokality, čo

znamená, že z niektorého webu bolo využitých viac stránok (napr. z *fit.vut.cz*). Približné rozdelenie stránok do kategórií je uvedené len pre bližšiu predstavu obsahu datasetu a v celkovom vyhodnotení uvedené kategórie nie sú zohľadnené.

Webové stránky v dátovej sade boli (subjektívne) rozdelené do dvoch kategórií na základe ich zložitosti (toto rozdelenie bolo zohľadnené vo výslednom porovnaní):

- Jednoduché stránky (15),
- Zložitejšie stránky (15).

7.2 Metriky

Na kvantitatívne (objektívne) vyhodnotenie podobnosti dvoch segmentácií sme sa rozhodli použiť nasledujúce metriky predstavené v [18]: presnosť (*precision*) P_{B^3}, spätná presnosť (*recall*) R_{B^3} a rozšírené F_1-skóre ako harmonický priemer P_{B^3} a R_{B^3}, F_{B^3}.

Na určenie vhodných metrík, ktoré umožňujú zachytiť podobnosť dvoch segmentácií je potrebné vnímať segmentáciu ako výsledok zhlukovania (čo v našom prípade presne odpovedá). Uvedené metriky teda vychádzajú práve z teoretického základu určenia podobnosti medzi zhlukmi. Aby bolo možné dané metriky vypočítať, na stránke musia byť identifikované atomické elementy, ktoré by bolo možné zoskupovať do zhlukov odpovedajúcich jednotlivým segmentom [18].

V publikácii sa spomínajú ako vhodní kandidáti na atomické prvky: pixely, uzly DOM a jednotlivé znaky (text). Na účel porovnania segmentácií sme sa rozhodli využiť ako atomické prvky: **všetky uzly** (rovnako ako v publikácii) a naviac **listové uzly** odpovedajúce extrahovaným boxom. Podobnosť segmentácií je určená práve vzhľadom na jeden typ atomických prvkov.

Nasledujúce rovnice [18] (7.1 – 7.3) reprezentujú výpočet uvedených metrík medzi dvoma segmentáciami stránky S a S^*.

$$P_{B^3}(S, S^*) = \frac{1}{|E^S|} \sum_{e \in E^S} \left(\frac{1}{|E_e^S|} \sum_{e' \in E_e^S} \left(\frac{min(|S_e \cap S_{e'}|, S_e^* \cap S_{e'}^*|)}{|S_e \cap S_e'|} \right) \right) \tag{7.1}$$

$$R_{B^3}(S, S^*) = P_{B^3}(S^*, S) \tag{7.2}$$

$$F_{B^3}(S, S^*) = \frac{2 \cdot P_{B^3}(S, S^*) \cdot R_{B^3}(S, S^*)}{P_{B^3}(S, S^*) + R_{B^3}(S, S^*)} \tag{7.3}$$

Jednotlivé premenné v uvedených rovniciach majú nasledujúci význam (podľa [18]) : $E = \{e_1, \ldots, e_n\}$ je množina atomických elementov na stránke; $S = \{s_1, \ldots, s_n\}$ je množina segmentov (segmentácia), pričom $s_i \subseteq E$; S^* je segmentácia porovnávaná s S; $E^S \subseteq E$ je množina elementov, ktoré patria do aspoň jedného segmentu S; $E_e^S \subset E$ je množina elementov, ktoré sú spolu s e v aspoň jednom segmente S; $S_e \subseteq S$ je množina segmentov, ktoré obsahujú element e.

Uvedené metriky sú priamočiarou adaptáciou rozšírených metrík *BCubed* z teórie zhlukov. Na výpočet priemernej hodnoty F_{B^3} na všetkých stránkach z dátovej sady, bola dodatočne definovaná metrika $F_{B^3}^*$, ktorá je harmonickým priemerom priemerných hodnôt P_{B^3} a R_{B^3} [19].

Je dôležité poznamenať, že P_{B^3} sa znižuje, ak algoritmicky vytvorené segmenty presahujú poza očakávané (*ground truth*) segmenty a R_{B^3} sa znižuje v opačnom prípade. Metriky teda dôverne zachytávajú podobnosť segmentácií a riešia problém s presegmentovaním (ak všetky elementy patria do jediného segmentu) a podsegmentovaním (ak každý element tvorí samostatný segment) [19].

7.3 Anotácia očakávaných segmentov

Na to, aby sme mohli porovnať dve rôzne segmentácie stránky na základe uvedených metrík potrebujeme v prvom rade vytvoriť očakávanú (*ground truth*, GT) segmentáciu. Táto segmentácia reprezentuje ako by mala podľa nás vyzerať očakávaná výsledná (a správna) segmentácia. Na základe toho, sme sa rozhodli vytvoriť jednoduchú aplikáciu, pomocou ktorej je možné vytvárať (anotovať) očakávané segmentácie interaktívnym spôsobom v prehliadači.

7.3.1 Aplikácia na anotáciu a výpočet metrík

Aplikácia je programovo (z terminálu) otváraná v inštancii prehliadača Chromium pomocou Playwrightu. Okrem vytvárania GT segmentov umožňuje načítať existujúce segmentácie zo súborového systému a vizuálne a kvantitatívne (na základe vypočítaných metrík) ich porovnať s vytvorenou GT segmentáciou. Na svoje fungovanie využíva súbory exportované aplikáciou `bcs` a k využitiu plného potenciálu vyžaduje spustenie s nasledujúcimi argumentmi:

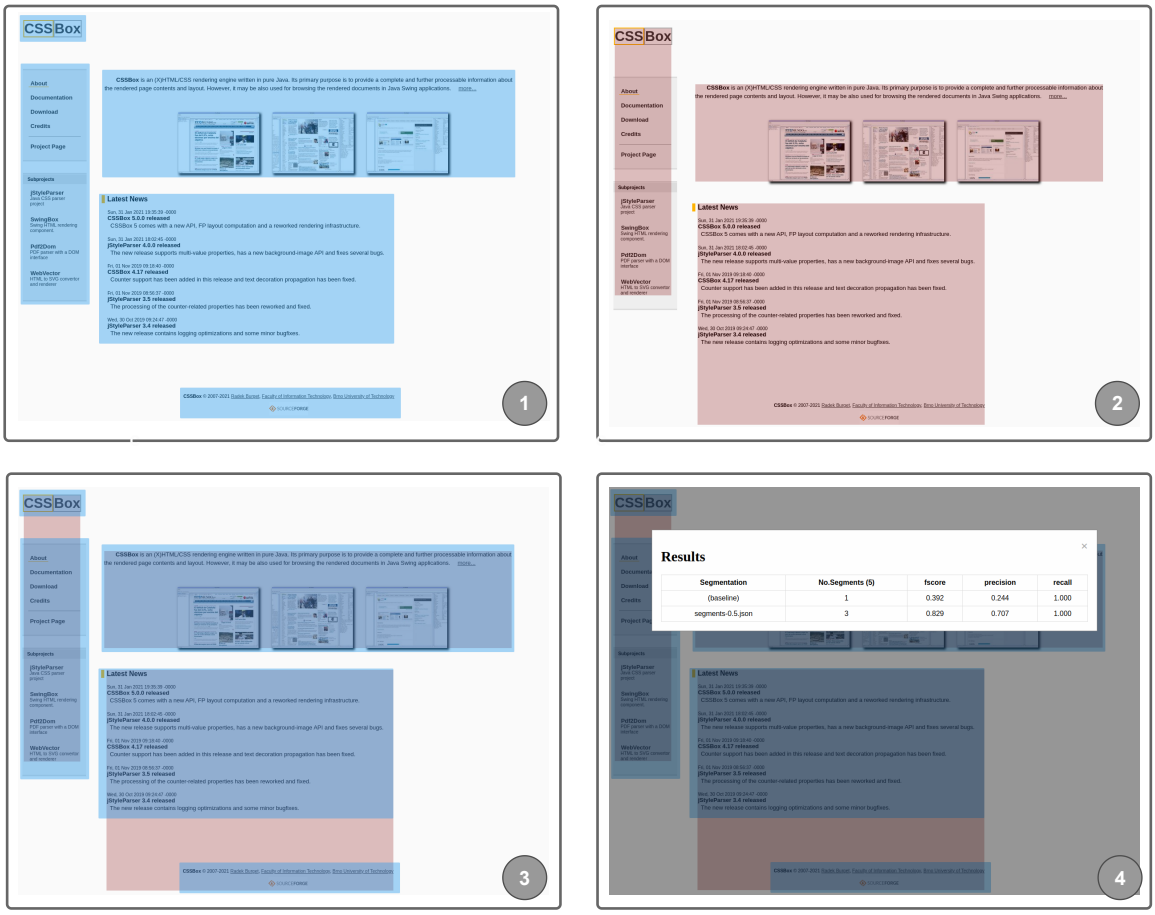
```
 <URL>                    URL stránky, pre ktorú chceme vytvoriť anotáciu
--WPNG, --webpage-png-filepath  Screenshot stránky (webpage.png)
--BPNG, --boxes-png-filepath    Screenshot boxov (boxes.png)
--BJSON, --boxes-json-filepath  Boxy vo formáte JSON (boxes.json)
--S1, --segmentation1           Segmentácia vo formáte JSON/XML
...                             (segments.json|xml)
```

Aby sme mohli pre danú stránku vytvoriť anotáciu musí byť špecifikovaná jej URL adresa a cesta k screenshotu stránky (`WPNG`). Nad screenshotom stránky, ktorý je na pozadí, je potom možné vytvárať GT segmenty a manipulovať s nimi (myšou a tlačidlami klávesnice). Pre budúce použitie je možné vytvorenú anotáciu exportovať do súboru.

Na výpočet metrík je aplikáciu naopak nutné spustiť s argumentom `BJSON`, ktorý špecifikuje množinu atomických elementov, a s minimálne jednou segmentáciou napr. `S1` (naraz je možné porovnať až 3 segmentácie – S1, S2, S3). V prípade, ak GT anotácia danej stránky už bola exportovaná, je načítaná automaticky a zároveň sú automaticky výpočítané a zobrazené aj metriky. Ak anotácia pre danú stránku neexistuje (nie je v priečinku `/output/`), je nutné ju vytvoriť a výpočet metrík spustiť odpovedajúcim tlačidlom na klávesnici.

Aby sme mohli porovnať výsledky segmentácie s referenčnou implementáciou BCS, aplikácia umožňuje načítať segmentáciu aj vo formáte XML, ktorú produkuje referenčná implementácia. Dôležité je poznamenať, že presnosť segmentácie je určená vzhľadom na *baseline* segmentáciu, ktorej odpovedá práve jeden segment (celá stránka), tak ako bolo uvedené v publikácii [19]. Len pre úplnosť spomenieme, že výpočet metrík bol implementovaný na základe rovníc 7.1 – 7.3 a na zisťovanie, ktorý segment obsahuje ktorý element (výpočet množín uvedených v rovniciach) bol použitý R-strom. Ďalšie implementačné detaily nie

sú spomenuté kvôli stručnosti. Na sérii obrázkov 7.1 je pre ukážku znázornená spustená aplikácia na anotáciu a výpočet metrík s odpovedajúcimi popismi.



Obr. 7.1: Screenshoty aplikácie na anotáciu GT segmentov a výpočet metrík, ktoré vyjadrujú podobnosť medzi GT anotáciou a algoritmicky vytvorenou segmentáciou. ① Očakávané GT segmenty. ② Algoritmicky vytvorená segmentácia s CT 0.5. ③ Zobrazenie oboch segmentácií na vizuálne porovnanie. ④ Zobrazenie výsledných metrík vypočítaných vzhľadom na GT anotáciu medzi *baseline* segmentáciou a algoritmicky vytvorenou segmentáciou.

7.4 Porovnanie s referenčnou implementáciou

Na overenie funkčnosti sme sa rozhodli porovnať vytvorené implementácie metódy BCS s referenčnou implementáciou BCS[1], ktorá je súčasťou nástroja FitLayout[2]. Na vyhodnotenie presnosti a kvality jednotlivých vytvorených segmentácií sme využili práve metriky definované v kapitole 7.2.

Aby bolo možné metriky vypočítať, bolo nutné vytvoriť GT anotácie pre všetky stránky z dátovej sady. Vytváranie anotácií prebiehalo so snahou dosiahnuť čo najväčšiu vizuálnu a/alebo sémantickú konzistenciu (s preferenciou oboch zároveň) jednotlivých segmentov. Na niektorých stránkach bol zámerne vytvorený väčší počet segmentov, čomu odpovedá

[1] https://github.com/janzeleny/bcs/
[2] https://github.com/FitLayout/FitLayout

nižšia úroveň podrobnosti (granularita). Bolo to z dôvodu, že väčší počet GT segmentov negatívne ovplyvňuje presnosť *baseline* segmentácie. Ak anotácia obsahuje relatívne málo segmentov, presnosť *baseline-u* je relatívne vysoká, čo môže skresľovať celkové výsledky. Cieľom teda bolo stiahnuť presnosť *baseline* pod určitú hranicu, aby výsledné porovnanie implementácií malo význam.

Kvôli možnosti porovnať výsledky segmentácií aj vzhľadom na zložitosť stránky boli spolu vytvorené 3 tabuľky, pričom tabuľka 7.1 odpovedá celej dátovej sade (30 stránok) a tabuľky 7.2, 7.3 odpovedajú množinám jednoduchých a zložitejších stránok. Tabuľky obsahujú okrem vypočítaných metrík aj priemerný počet vytvorených segmentov, pričom priemerný počet segmentov GT anotácie je **8.5** a rámcové porovnanie času zhlukovania, resp. času potrebného na vytvorenie segmentácie.

Z uvedených tabuliek (7.1 – 7.3) vyplýva, že v konečnom vyhodnotení sme porovnali 3 implementácie a *baseline*. Referenčnej implementácii v tabuľkách odpovedá stĺpec *FitLayout BCS*. Dvom implementovaným variantám BCS, ktoré boli bližšie popísané v kapitole 6 potom odpovedajú stĺpce *Základná* a *Rozšírená implementácia*. Všetky implementácie boli spustené na každej stránke dátovej sady 8-krát s hodnotami CT od 0.1 do 0.8 s krokom 0.1. Ukázalo sa, že vyššia hodnota CT ako 0.8 u žiadnej implementácie už presnosť segmentácie nezvyšuje. Zo získaných výsledkov boli následne do porovnania vybrané segmentácie s najvyššou hodnotou F_{B^3}, teda tie ktoré z hľadiska zhlukovej podobnosti najviac odpovedali očakávanej segmentácii (GT anotácii).

Elementy	**Metrika**	**Baseline**	**FitLayout BCS**	**Základná impl.**	**Rozšírená impl.**
listové uzly	Segmenty	1	13	9	9.5
	F_{B^3}	0.341	0.737	0.739	**0.771**
	P_{B^3}	0.213	0.763	0.718	**0.774**
	R_{B^3}	1.000	0.736	**0.794**	0.790
	$F^*_{B^3}$	0.262	0.750	0.728	**0.772**
uzly	Segmenty	1	17	9.5	9
	F_{B^3}	0.367	**0.761**	0.748	**0.762**
	P_{B^3}	0.235	**0.833**	0.738	0.764
	R_{B^3}	0.990	0.715	0.783	**0.787**
	$F^*_{B^3}$	0.287	**0.795**	0.743	0.763
	Čas výpočtu	-	74.533	**56.462**	66.067

Tabuľka 7.1: Porovnanie implementácií na všetkých stránkach dátovej sady. Výsledky ukazujú, že všetky porovnané implementácie u oboch typov atomických elementov dosahujú takmer rovnaké F_{B^3} skóre. Priemerný počet segmentov (8.5) je najbližšie k počtu segmentov základnej implementácie (9). Priemerné časy zhlukovania uvedené v *ms* sú rámcovo taktiež veľmi podobné.

Interpretáciou získaných výsledkov porovnania všetkých uvažovaných implementácií môžeme dospieť k záveru, že obidve vytvorené implementácie (*základná, rozšírená*) presnosťou segmentácie vyjadrenou prostredníctvom metrík v priemere odpovedajú referenčnej implementácii.

Výraznejšie rozdiely hodnôt F_{B^3} skóre môžeme pozorovať predovšetkým u listových uzlov v tabuľke 7.3, ktorá obsahuje výsledky množiny zložitejších stránok. Je to zrejme z dôvodu iného prístupu pri extrakcii boxov a s tým spojeného určenia iného počtu a množiny boxov ako vstupu pre proces zhlukovania. Pre úplnosť spomenieme, že na extrakciu

boxov v referenčnej implementácii je použitý framework Puppeteer. Ale vzhľadom na to, že v oboch prípadoch je použitá rovnaká inštancia prehliadača (Chromium), použitie iného frameworku by malo mať na výsledky minimálny vplyv.

Elementy	Metrika	Baseline	FitLayout BCS	Základná impl.	Rozšírená impl.
listové uzly	Segmenty	1	10	7	8
	F_{B^3}	0.389	0.780	0.761	**0.814**
	P_{B^3}	0.250	0.800	0.743	**0.828**
	R_{B^3}	1.000	0.794	0.815	**0.823**
	$F^*_{B^3}$	0.305	0.790	0.752	**0.821**
uzly	Segmenty	1	13	7	11.5
	F_{B^3}	0.440	**0.801**	0.789	0.723
	P_{B^3}	0.289	**0.866**	0.773	0.707
	R_{B^3}	0.990	0.756	**0.825**	0.759
	$F^*_{B^3}$	0.349	**0.832**	0.781	0.715
	Čas výpočtu	-	71.721	**41.398**	48.088

Tabuľka 7.2: Porovnanie implementácií na množine **jednoduchých** stránok. Výsledky ukazujú, že rozšírená implementácia dosahuje mierne vyššie hodnoty F_{B^3} skóre pri porovnaní na základe listových uzlov. Naopak F_{B^3} skóre referenčnej implementácie je vyššie pri porovnaní na základe všetkých DOM uzlov. Zhlukovanie u oboch vytvorených implementácií v porovnaní s referenčnou bolo v priemere 1.5× rýchlejšie.

Elementy	Metrika	Baseline	FitLayout BCS	Základná impl.	Rozšírená impl.
listové uzly	Segmenty	1	16	11.5	6.5
	F_{B^3}	0.300	0.677	0.723	**0.807**
	P_{B^3}	0.183	0.729	0.701	**0.800**
	R_{B^3}	1.000	0.648	0.775	**0.829**
	$F^*_{B^3}$	0.227	0.702	0.712	**0.803**
uzly	Segmenty	1	22	12.5	11.5
	F_{B^3}	0.291	**0.710**	0.706	**0.710**
	P_{B^3}	0.178	**0.799**	0.707	0.716
	R_{B^3}	0.992	0.659	0.736	**0.749**
	$F^*_{B^3}$	0.221	**0.752**	0.706	0.713
	Čas výpočtu	-	78.212	**76.162**	89.577

Tabuľka 7.3: Porovnanie implementácií na množine **zložitejších** stránok. Rozšírená implementácia dosahuje lepšie výsledky v hornej časti. V dolnej časti sú naopak výsledky medzi všetkými implementáciami takmer totožné. Doba trvania sa na zložitejších stránkach u oboch vytvorených implementácii zvyšuje v závislosti od počtu boxov a vzťahov medzi nimi.

Ďalším dôvodom, prečo dochádzalo k rozdielom oproti referenčnej implementácii môže byť fakt, že pri výbere najlepšej segmentácie (s najvyšším F_{B^3} skóre) sa nebral ohľad na počet vytvorených segmentov. To si môžeme všimnúť napr. v tabuľke 7.1, že u výsledkov v spodnej časti sa líši priemerný počet segmentov oproti referenčnej implementácii takmer dvojnásobne.

Jedným z ďalších možných dôvodov spôsobujúcich rozdiely medzi výsledkami môže byť použitie špecifického váhového vektoru $[0.2, 0.3, 0.5]$ v referenčnej implementácii vo výpočte základnej podobnosti (rovnica 3.5):

$$0.2 * distance + 0.3 * sim\_shape + 0.5 * sim\_color$$

Váhový vektor v prvom rade ovplyvňuje veľkosť zhlukovacieho prahu, pričom približne rovnaké výsledky segmentácií vytvorenej našou a referenčnou implementáciou môžeme pozorovať pri hodnotách CT 0.5 (základná, rozšírená) a 0.3 (referenčná). Z hodnôt váhového vektoru tiež vyplýva, že vyššia váha sa prikladá farebnej zložke podobnosti a najnižšia váha relatívnej vzdialenosti medzi boxmi. To pochopiteľne ovplyvňuje výber najlepšieho kandidáta na zlúčenie a tiež poradie vytvárania zhlukov.

Posledným identifikovaným faktorom, ktorý by podľa nás mohol spôsobovať rozdielne výsledky je citlivosť metódy BCS na zhlukovacie prahy. Kvôli jednoduchosti a väčšej objektivite pri porovnaní implementácií sme použili zhlukovacie prahy od 0.1 zväčšujúce sa po desatinách. Použitie hodnôt zhlukovacieho prahu s vyššou presnosťou (napr. 0.25), ale môže viesť k výrazne odlišnej segmentácii, čo sa týka hodnôt vypočítaných metrík.

7.5 Obmedzenia a limity

Na základe celkových výsledkov a uvedeného porovnania s referenčnou implementáciou môžeme tvrdiť, že implementácia metódy BCS bola úspešná. Implementácia bola vytvorená s cieľom, aby čo najviac odpovedala teoretickému popisu metódy v kapitole 3. V rámci analýzy metódy BCS bolo identifikovaných niekoľko problémov súvisiacich so samotnými definíciami. Aj keď pre jednotlivé problémy boli navrhnuté riešenia, implementácia má stále isté obmedzenia a limity, s ktorými musíme počítať. Isté limity zrejme existujú v extrakcii boxov, zhlukovaní a zároveň aj v konečnom vyhodnotení a porovnaní s referenčnou implementáciou.

Extrakcia boxov

Obmedzenie extrakcie boxov spočíva v tom, že je získavaná iba množina boxov na základe špecifikovaných pravidiel uvedených v 3.2. Na niektorých stránkach s neobvyklou HTML štruktúrou môže dochádzať k situáciám, kedy je zo stránky extrahovaný príliš malý či nedostatočný počet boxov roztrúsených po celej stránke, ktoré spolu vizuálne ani sémanticky nesúvisia. Niekedy sú práve takéto boxy zhlukované spolu, čo vedie k neočakávaným výsledkom segmentácie.

Zhlukovanie

Obmedzením, ktoré priamo predchádza procesu zhlukovania je odstránenie boxov, ktoré sa prekrývajú s inými a tých, ktoré vizuálne obsahujú iné boxy. Práve boxy (kontajnery), ktoré vizuálne obsahujú ďalšie boxy väčšinou na stránke tvoria vizuálne konzistentný blok, ktorý sa nám pri anotácii očakávaných segmentov zdá rozumné označiť ako jeden segment. Kontajnery častokrát nemajú význam z hľadiska extrakcie informácií, lebo väčšinou ide o obrázky na pozadí. Nie je preto triviálne rozhodnúť, ktoré kontajnery by malo význam pred zhlukovaním zachovať. Na základe špecifikácie BCS musia byť odstránené všetky kontajnery, čo v konečnom dôsledku predstavuje isté obmedzenie.

Ďalším obmedzením, ktoré súvisí so zhlukovaním je už spomínaná citlivosť na zhlukovacie prahy. V niektorých prípadoch môže presnosť výslednej segmentácie závisieť od jediného spojenia dvoch konkrétnych zhlukov. Predpokladajme, že vypočítaná podobnosť medzi zhlukmi je 0.5005 a očakávali by sme, že ich spojením vznikne segmentácia s najvyššou presnosťou. Následne pri použití CT s hodnotou 0.5, zhluky spojené nebudú a presnosť bude nižšia. Ak ale použijeme CT s hodnotou 0.6 môže dôjsť k spojeniam ďalších zhlukov, čo taktiež povedie k zníženiu výslednej presnosti.

Posledné obmedzenie spojené s vytváraním zhlukov predstavuje výpočet zhlukovej podobnosti. V istých špecifických situáciách môže byť uprednostnený vzťah medzi zhlukmi, aj keď by sme očakávali, že by mal byť ako najlepší kandidát vybraný práve vzťah medzi dvomi boxmi, ktoré evidentne patria k sebe (sú vizuálne konzistentné). Je to z dôvodu, že výpočet kumulatívnej podobnosti závisí od všetkých vzájomných vzťahov boxov v oboch zhlukoch. V prípade, že počet daných vzťahov je relatívne vysoký, výpočet zhlukovej podobnosti, ktorá je vyjadrená ako podiel kumulatívnej podobnosti a kardinality spôsobí, že hodnota podobnosti medzi zhlukmi bude nižšia a zároveň je aj ťažko predvídateľná. To môže viesť k vytvoreniu neočakávaných zhlukov. Podobne ako u obmedzenia extrakcie boxov toto obmedzenie súvisí s princípom výpočtu podobnosti medzi zhlukmi podľa uvedených definícií v kapitole 3.4.2.

7.5.1 Vyhodnotenie a porovnanie

Obmedzením vytvoreného vyhodnotenia a porovnania je vytvorenie jedinej GT anotácie pre každú stránku, pričom anotácia je zrejme dosť subjektívna. Toto obmedzenie síce nepredstavuje problém pri vyhodnotení metrík, ale vytvorenie viacerých anotácií rôznymi užívateľmi a nájdenie zhody medzi nimi by mohlo vo vyhodnotení priniesť zaujímavejšie a vierohodnejšie výsledky.

Limitom vyhodnotenia je aj obmedzená a relatívne malá dátová sada, aj keď bola vytvorená ako reprezentatívna vzorka webových stránok s rôznou zložitosťou a štruktúrou. Podobne ako v predchádzajúcom prípade, pri testovaní a vyhodnotení metrík na väčšej dátovej sade by sme mohli získať vecnejšie výsledky za cenu vytvárania väčšieho množstva anotácií a rozsiahlejšieho počítania metrík.

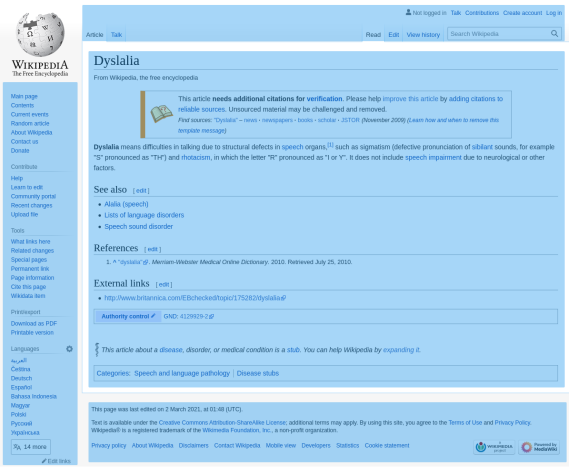
7.6 Ukážka vizuálneho porovnania

V tejto kapitole je na obrázku 7.3 znázornené vizuálne porovnanie vytvorených segmentácií referenčnou a nami vytvorenou (rozšírenou) implementáciou na ukážkovej stránke[3]. Jednotlivé segmentácie boli vybrané z množiny vytvorených segmentácii s rôznymi hodnotami zhlukovacích prahov. Na porovnanie boli vybrané segmentácie s najvyšším F_{B^3} skóre a tiež s ohľadom na najväčšiu vizuálnu podobnosť v porovnaní s očakávanou (*ground truth*) segmentáciou, ktorá je znázornená na obrázku 7.2.

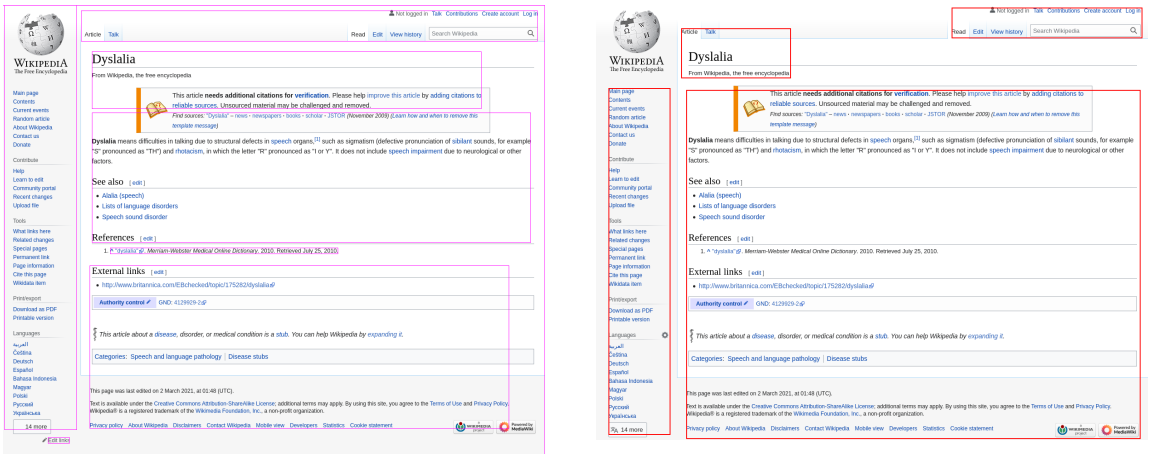
Ako môžeme vidieť na vizuálnom porovnaní segmentácií na obrázku 7.3, referenčná implementácia dokázala na danej stránke lepšie identifikovať segmenty, ktoré odpovedajú navigácii na stránke (najľavejší a navrchnejší segment na obrázku (a)). Rozšírená implementácia správne identifikovala navigačný panel naľavo, ale časť navigácie v hornej časti stránky nesprávne zlúčila s nadpisom článku. Hlavný blok obsahu stránky bol na tomto konkrétnom príklade lepšie identifikovaný našou implementáciou. Z neznámych príčin, ktoré neboli

[3] https://en.wikipedia.org/wiki/Dyslalia

predmetom skúmania, nedošlo v referenčnej implementácii k spojeniu jednotlivých segmentov v hlavnom bloku obsahu, čo značne ovplyvnilo aj vypočítanú hodnotu F_{B^3} skóre a teda aj celkovú presnosť. Ani jedna z implementácií pri danom zhlukovacom prahu nedokázala odčleniť odkazy s obrázkami v spodnej časti stránky, ktorým na obrázku 7.2 odpovedá najspodnejší samostatný segment.



Obr. 7.2: Očakávaná (*ground truth*) segmentácia stránky vybranej na vizuálne porovnanie vytvorená pomocou aplikácie na anotáciu.



(a) (b)

Obr. 7.3: Porovnanie najlepších segmentácií stránky vytvorených referenčnou (a) a rozšírenou (b) implementáciou metódy BCS. U oboch implementácii bol použitý zhlukovací prah 0.4. Hodnoty F_{B^3} skóre boli nasledujúce: 0.719 u referenčnej implementácie a u rozšírenej 0.836, čo zrejme odpovedá aj vizuálnemu porovnaniu segmentácii.

V priebehu implementácie bolo vzájomne vizuálne porovnaných a vyhodnotených (s referenčnou implementáciou) veľké množstvo stránok (okrem stránok z dátovej sady). V konečnom dôsledku môžeme povedať, že na niektorých stránkach lepšie fungovala referenčná implementácia a na niektorých zasa naša, a to buď jej základná alebo rozšírená varianta.

Kapitola 8

Záver

Hlavným zámerom tejto diplomovej práce bola implementácia metódy segmentácie webových stránok vo webovom prehliadači. Ako metódu, ktorú sme sa rozhodli implementovať bola zvolená metóda BCS (*Box Clustering Segmentation*) založená na zhlukovaní boxov, resp. relevantných listových DOM uzlov. Jej výstupom je množina zhlukov, ktoré by mali z hľadiska segmentácie čo najvernejšie odpovedať vizuálne alebo sémanticky konzistentným blokom. Dôležitým aspektom segmentácie stránky v prehliadači je využitie vhodného frameworku na jeho automatizáciu, pomocou ktorého je možné spustiť inštanciu prehliadača s otvorenou stránkou na základe sprostredkovanej URL adresy.

V rámci tejto práce boli vytvorené dve implementácie metódy BCS. Základná implementácia bola vytvorená s cieľom, aby čo najviac zodpovedala teoretickému popisu metódy. V rozšírenej implementácii sa vo výpočte podobnosti naviac zohľadňuje tzv. skóre priliehavosti (*alignment score*) za účelom zlepšenia presnosti a kvality segmentácie. Jedným z hlavných rozdielov pri vytváraní zhlukov medzi vytvorenými implementáciami je, že v rozšírenej implementácii je povolené prekrytie zhluku v špeciálnom prípade prekrytia, kedy kandidátny zhluk vizuálne obsahuje iný zhluk. V rámci rozšírenej implementácie je možné použiť aj ďalšie dodatočné argumenty, ktoré v prípade ak sú použité so správnymi (špecifickými) hodnotami, môžu viesť k zvýšeniu presnosti. Príkladom môže byť použitie váhového vektoru vo výpočte podobnosti pre jej jednotlivé zložky.

Dôležitou súčasťou implementácie je aj interaktívna vizualizácia vzťahov medzi extrahovanými boxmi, ktorá sprostredkuje možnosť podrobne skúmať priebeh a vývoj segmentácie v jednotlivých segmentačných krokoch. Z hľadiska následného vizuálneho porovnania, a objektívneho kvantitatívneho porovnania na základe metrík, bolo implementovaných viacero možností pre export dát vo formáte JSON a PNG. Aby bolo možné vytvorené implementácie porovnať s referenčnou implementáciou bola taktiež vytvorená aplikácia na anotáciu očakávaných výsledných segmentov (*ground truth segmentation*) s možnosťou jednoduchého vizuálneho porovnania.

Výsledkom práce je funkčná implementácia metódy BCS s použitím automatizovaného prehliadača, implementovaná v jazyku *JavaScript* so súčasným využitím technológie *Node.js* na serverovej strane. Implementácia bola vytvorená s ohľadom na jej efektivitu a z toho dôvodu boli zavedené viaceré optimalizácie. Funkčnosť implementácie bola overená na množine webových stránok z vytvorenej dátovej sady. Obidve vytvorené implementácie boli porovnané s referenčnou implementáciou na základe metrík [18], ktoré vecne a dôverne zachytávajú podobnosť medzi dvoma segmentáciami. Je dôležité poznamenať, že segmentácie boli porovnávané vzhľadom na vytvorené anotácie očakávaných segmentov. Dosiahnuté výsledky ukázali, že obidve vytvorené implementácie presnosťou približne odpovedajú refe-

renčnej implementácii. Na základe porovnania môžeme považovať vytvorenú implementáciu za funkčnú a použiteľnú.

Samotná metóda BCS a definície, na ktorých je založená majú isté obmedzenia a limity, ktoré negatívne ovplyvňujú výslednú segmentáciu. Aj z toho dôvodu boli v priebehu vývoja referenčnej implementácie zavedené dodatočné metriky a zmeny vo výpočte podobnosti. V niektorých prípadoch je vytvorená segmentácia výrazne ovplyvnená i obmedzenou množinou extrahovaných boxov, na základe ktorej je takmer nemožné vytvoriť segmentáciu podobnú tej očakávanej. Aj keď na niektorých stránkach segmentácia nedopadne podľa očakávaní, dosiahnuté výsledky považujeme za veľmi uspokojivé. Vzhľadom na to môžeme prehlásiť, že všetky vytýčené ciele tejto práce boli úspešne splnené.

Literatúra

[1] *About Node.js* [online]. OpenJS Foundation, 2011 [cit. 2020-12-29]. Dostupné z: https://nodejs.org/en/about/.

[2] Akpinar, E. a Yesilada, Y. Vision Based Page Segmentation Algorithm: Extended and Perceived Success. In: *Current Trends in Web Engineering*. Júl 2013, sv. 8295, s. 238–252. DOI: 10.1007/978-3-319-04244-2_22. ISBN 978-3-319-04243-5.

[3] Alcic, S. a Conrad, S. Page Segmentation by Web Content Clustering. In: *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*. New York, NY, USA: Association for Computing Machinery, 2011. WIMS '11. DOI: 10.1145/1988688.1988717. ISBN 9781450301480.

[4] *Are those really SARS-CoV-2 virions in the kidney?* [online]. Parma, Italy: Era-Edta, 2020 [cit. 2021-04-20]. Dostupné z: https://www.era-edta.org/en/are-those-really-sars-cov-2-virions-in-the-kidney/.

[5] Brewster, C. *15 Companies That Use Node.Js in 2020 Successfully* [online]. Trio, 2020 [cit. 2020-12-28]. Dostupné z: https://trio.dev/blog/companies-use-node-js.

[6] Burget, R. Visual Area Classification for Article Identification in Web Documents. In: *2010 Workshops on Database and Expert Systems Applications*. 2010, s. 171–175. DOI: 10.1109/DEXA.2010.49. ISBN 978-1-4244-8049-4.

[7] Cai, D., Yu, S., Wen, J.-R. a Ma, W.-Y. Extracting Content Structure for Web Pages Based on Visual Representation. In: *Proceedings of the 5th Asia-Pacific Web Conference on Web Technologies and Applications*. Berlin, Heidelberg: Springer-Verlag, 2003, s. 406–417. APWeb'03. ISBN 3540023542.

[8] Chen, Y., Ma, W.-Y. a Zhang, H.-J. Detecting Web Page Structure for Adaptive Viewing on Small Form Factor Devices. In: *Proceedings of the 12th International Conference on World Wide Web*. New York, NY, USA: Association for Computing Machinery, 2003, s. 225–233. WWW '03. DOI: 10.1145/775152.775184. ISBN 1581136803.

[9] Cormier, M., Moffatt, K., Cohen, R. a Mann, R. Purely Vision-Based Segmentation of Web Pages for Assistive Technology. *Computer Vision and Image Understanding*. USA: Elsevier Science Inc. júl 2016, zv. 148, C, s. 46–66. ISSN 1077-3142.

[10] Debnath, S., Mitra, P. a Giles, C. L. Identifying Content Blocks from Web Documents. In: *Proceedings of the 15th International Conference on Foundations of*

Intelligent Systems. Berlin, Heidelberg: Springer-Verlag, 2005, s. 285–293. ISMIS'05. DOI: 10.1007/11425274_30. ISBN 3540258787.

[11] Eldirdiery, H. F. a Ahmed, A. H. Web Document Segmentation for Better Extraction of Information: A Review. *International Journal of Computer Applications*. Január 2015, zv. 110, s. 24–28. DOI: 10.5120/19297-0734.

[12] Feng, H., Zhang, W., Wu, H. a Wang, C. Web Page Segmentation and Its Application for Web Information Crawling. In: *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*. 2016, s. 598–605. DOI: 10.1109/ictai.2016.0097. ISBN 978-1-5090-4459-7.

[13] Goodman, D. *Dynamic HTML: The Definitive Reference*. 3. vyd. O'Reilly Media, 2007. ISBN 978-0596527402.

[14] Grigorik, I. *Primer on Web Performance* [online]. O'Reilly Media, 2013 [cit. 2020-11-30]. Dostupné z: https://hpbn.co/primer-on-web-performance/.

[15] Gruenbaum, B. *Puppeteer, Selenium, Playwright, Cypress – how to choose?* [online]. Testim, 2020 [cit. 2020-01-25]. Dostupné z: https://www.testim.io/blog/puppeteer-selenium-playwright-cypress-how-to-choose/.

[16] Hong, J. L., Siew, E.-G. a Egerton, S. Information Extraction for Search Engines Using Fast Heuristic Techniques. *Data & Knowledge Engineering*. NLD: Elsevier Science Publishers B. V. február 2010, zv. 69, č. 2, s. 169–196. DOI: 10.1016/j.datak.2009.10.002. ISSN 0169-023X.

[17] Kaneriya, T. *Node.js vs Django: Key differences, popularity, use cases and more* [online]. Simform, 2021 [cit. 2020-01-25]. Dostupné z: https://www.simform.com/nodejs-vs-django/.

[18] Kiesel, J., Kneist, F., Meyer, L., Komlossy, K., Stein, B. et al. Web Page Segmentation Revisited: Evaluation Framework and Dataset. In: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*. New York, NY, USA: Association for Computing Machinery, 2020, s. 3047–3054. CIKM '20. DOI: 10.1145/3340531.3412782. ISBN 9781450368599.

[19] Kiesel, J., Meyer, L., Kneist, F., Stein, B. a Potthast, M. An Empirical Comparison of Web Page Segmentation Algorithms. In: Hiemstra, D., Moens, M.-F., Mothe, J., Perego, R., Potthast, M. et al., ed. *Advances in Information Retrieval. 43rd European Conference on IR Research (ECIR 2021)*. Berlin Heidelberg New York: Springer, Marec 2021, sv. 12657, s. 62–74. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-72240-1_5. ISBN ISBN 978-3-030-72240-1.

[20] Kohlschütter, C. a Nejdl, W. A Densitometric Approach to Web Page Segmentation. In: *Proceedings of the 17th ACM Conference on Information and Knowledge Management*. New York, NY, USA: Association for Computing Machinery, 2008, s. 1173–1182. CIKM '08. DOI: 10.1145/1458082.1458237. ISBN 9781595939913.

[21] Kreuzer, R., Hage, J. a Feelders, A. A Quantitative Comparison of Semantic Web Page Segmentation Approaches. In: Cimiano, P., Frasincar, F., Houben,

G.-J. a Schwabe, D., ed. *Engineering the Web in the Big Data Era*. Springer International Publishing, Jún 2015, s. 374–391. DOI: 10.1007/978-3-319-19890-3_24. ISBN 978-3-319-19890-3.

[22] Lengál, T. *Segmentace webových stránek s využitím shlukování*. Brno, CZ, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/19293/.

[23] Mohsen, A., Pedram, M. a Rahmani, A. Main Content Extraction from Detailed Web Pages. *International Journal of Computer Applications*. August 2010, zv. 4. DOI: 10.5120/869-1219.

[24] Pasquali, S. a Faaborg, K. *Mastering Node.js - Second Edition: Build robust and scalable real-time server-side web applications efficiently*. 2. vyd. Packt, december 2017. ISBN 978-1785888960.

[25] Prasad, J. a Paepcke, A. Coreex: Content Extraction from Online News Articles. In: *Proceedings of the 17th ACM Conference on Information and Knowledge Management*. New York, NY, USA: Association for Computing Machinery, 2008, s. 1391–1392. CIKM '08. DOI: 10.1145/1458082.1458295. ISBN 9781595939913.

[26] Robbins, J. N. *Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics*. O'Reilly Media, máj 2018. ISBN 978-1491960202.

[27] Sanoja, A. a Gançarski, S. Web Page Segmentation Evaluation. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. New York, NY, USA: Association for Computing Machinery, 2015, s. 753–760. SAC '15. DOI: 10.1145/2695664.2695786. ISBN 9781450331968.

[28] Sanoja, A. a Gançarski, S. Block-o-Matic: A web page segmentation framework. In: *2014 International Conference on Multimedia Computing and Systems (ICMCS)*. 2014, s. 595–600. DOI: 10.1109/ICMCS.2014.6911249. ISBN 978-1-4799-3824-7.

[29] Solanki, J. *Comparing Nodejs vs Java: Your Backend Tech Stacks Explained* [online]. Simform, 2021 [cit. 2020-01-25]. Dostupné z: https://www.simform.com/nodejs-vs-java/.

[30] Song, R., Liu, H., Wen, J.-R. a Ma, W.-Y. Learning Important Models for Web Page Blocks Based on Layout and Content Analysis. *SIGKDD Explor. Newsl.* New York, NY, USA: Association for Computing Machinery. december 2004, zv. 6, č. 2, s. 14–23. DOI: 10.1145/1046456.1046459. ISSN 1931-0145.

[31] Tayar, G. *Comparing JavaScript Browser Automation Frameworks: Selenium Versus Webdriver.io Versus Puppeteer* [online]. Applitools, 2018 [cit. 2020-12-29]. Dostupné z: https://applitools.com/blog/comparing-javascript-browser-automation-frameworks/.

[32] Uma, R. a Latha, B. Noise elimination from web pages for efficacious information retrieval. *Cluster Computing*. November 2019, zv. 22. DOI: 10.1007/s10586-018-2366-x.

[33] Vadrevu, S., Gelgi, F. a Davulcu, H. Semantic Partitioning of Web Pages. In: Ngu, A. H. H., Kitsuregawa, M., Neuhold, E. J., Chung, J.-Y. a Sheng, Q. Z., ed. *Web Information Systems Engineering – WISE 2005*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, s. 107–118. ISBN 978-3-540-32286-3.

[34] Vargas, A. S. *Web page segmentation, evaluation and applications* [online]. Paris VI, 2015. Dizertačná práca. Université Pierre et Marie Curie. English. Dostupné z: https://tel.archives-ouvertes.fr/tel-01128002/document.

[35] Win, C. S. a Thwin, M. M. S. Informative Content Extraction By Using Eifce Effective Informative Content Extractor. *International Journal of Scientific & Technology Research*. Jún 2013, zv. 2, s. 136–144. ISSN 2277-8616.

[36] Yesilada, Y. *Web Page Segmentation: A Review*. figshare, Apr 2014. DOI: 10.6084/m9.figshare.979322.v1. Dostupné z: https://figshare.com/articles/journal_contribution/Web_Page_Segmentation_A_Review/979322/1.

[37] Yi, L., Liu, B. a Li, X. Eliminating Noisy Information in Web Pages for Data Mining. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2003, s. 296–305. KDD '03. DOI: 10.1145/956750.956785. ISBN 1581137370.

[38] Zeleny, J., Burget, R. a Zendulka, J. Box clustering segmentation: A new method for vision-based web page preprocessing. *Information Processing and Management*. 2017, zv. 53, č. 3, s. 735 – 750. DOI: 10.1016/j.ipm.2017.02.002. ISSN 0306-4573.

Príloha A

Obsah CD

- **/xzubri00.pdf** – súbor obsahujúci text tejto diplomovej práce
- **/tex/** – adresár obsahujúci zdrojový kód pre vytvorenie textu práce
- **/bcs-nodejs/** – adresár obsahujúci zdrojové kódy vytvorenej aplikácie
- `/README` – súbor obsahujúci podrobnejšie informácie o obsahu CD