VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

METODY SEGMENTACE WEBOVÝCH STRÁNEK

WEB PAGE SEGMENTATION METHODS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MARTIN GRNÁČ**
AUTHOR

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

BRNO 2023

Abstrakt

Táto práca sa zaoberá segmentačnými metódami. Rozoberá ich na teoretickej úrovni, popisuje ich vlastnosti, výhody a nevýhody. V rámci práce bola nakoniec zvolená segmentačná metóda Block-o-Matic, ktorá bola implementovaná do rámca FitLayout. Po implementácií bola zhodnotená a porovnaná s jej referenčnou implementáciou.

Abstract

This thesis focuses on segmentation methods. It discusses them at a theoretical level, describes their properties, advantages, and disadvantages. Within the scope of this work, the segmentation method Block-o-Matic was ultimately chosen and implemented within the FitLayout framework. After the implementation, it was evaluated and compared to its reference implementation.

Kľúčové slová

segmentácia, web, heuristika, FitLayout, algoritmy

Keywords

segmentation, web, heuristic, FitLayout, algorithm

Citácia

GRNÁČ, Martin. *Metody segmentace webových stránek*. Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Radek Burget, Ph.D.

Metody segmentace webových stránek

Prehlásenie

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

.....
Martin Grnáč
17. mája 2023

Poďakovanie

Chcel by som sa poďakovať doc. Ing. Radkovi Burgetovi, Ph.D. za odborné vedenie práce a pomoc.

Obsah

Zoznam obrázkov

Kapitola 1

Abstrakt

Webové stránky sú čoraz zložitejšie z dôvodu pokroku webových technologií, reklamného a interaktívneho obsahu, novým možnostiam vo webdizajne a nástrojom pre generovanie webového obsahu a webových stránok - WYSIWYG editory. Ich analýza, t.j. automatická identifikácia a klasifikácia rôznych prvkov vo webových stránkach, ako je hlavný obsah, navigačné menu, obrázky, reklamy a pod. sa stáva zložitým. Aby bolo možné úspešne analyzovať obsah na stránke je nutnosťou rozdeliť a identifikovať jedlotlivé prvky webovej stránky. Segmentácia predstavuje proces rozdelenia webovej stránky na vizuálne a sémanticky koherentné segmenty. Táto práca sa zaoberá problematikou segmentácie webového obsahu. Snaží sa nájisť vhodnú segmentačnú metódu, ktorá bude implementovaná do rámca FitLayout. Následne po tomto procese, zhodnotiť túto implementáciu a porovnať jej výsledky s dostupnou referenčnou implementáciou.

Kapitola 2

Úvod

Webové stránky tvoria značnú časť dnešného internetu a sú stavebným pilierom systému *World Wide Web*, známeho aj ako *Web*. Web si môžeme predstaviť ako rozsiahly informačný systém, ktorého úlohou je poskytovať prístup k webovým stránkam a ďalším webovým zdrojom cez internet. Užívateľ má v pláne si v blízkej dobe zaobstarať nejaký produkt, napr. kávu. Aby sa dozvedel potrebné informácie, otvára *prehliadač* a zadáva doň požiadavku vo forme *webovej adresy* svojho obľúbeného eshopu. Požiadavka je ďalej smerovaná na *webový server*, resp. na jeden z viacerých webových serverov, ktorý spadá pod daný eshop. Server požiadavku spracuje a odosiela do užívateľovho prehliadača webovú stránku. Prehliadač stránku vykreslí a užívateľ pokračuje ďalej v prehliadaní, výbere a nákupe.

S nástupom vyhľadávačov ako je napr. *Google Search* sa prístup k webovému obsahu zefektívnil. Ak si užívateľ chce spraviť širší prehľad a neviazať sa len na svoj obľúbený eshop, stačí ak do vyhľadávača zadá pár kľúčových slov. Následne sú užívateľovi vrátené výsledky stránok, ktoré vyhľadávací algoritmus vyhodnotil ako relevantné vzhľadom na úžívateľovu požiadavku, resp. kľúčové slová. U dnešných vyhľadávačov je štandardom poskytovanie aj iných výsledkov ako len zoznam relevantných stránok. Napríklad ak zadáme do vyhľadávača slová „najvyššia budova sveta“ (môžeme použiť vyššie spomínaný Google Search), dostaneme výsledky vo forme obrázkov, správ, obchodov, videí alebo lokacií na mape. Ak sa zameriame na vyhľadané obrázky, zisťujeme, že veľkú časť z nich tvoria fotky mrakodrapu Burj Khalifa (najvyššia budova sveta v dobe písania práce).

Tu je vidieť, že sa vyhľadávače snažia prehľadávať stránku, detegovať hlavný obsah a extrahovať z tohto obsahu informácie a zdroje, ktoré sa ďalej spracovávajú, analyzujú a sú použité na zlepšenie vyhľadávania a poskytovanie relevantnejších výsledkov. Ak sa zameriame na priemernú webovú stránku z hľadiska vizuálu, tak si všimneme že obsah, ktorý spolu súvisí, je zoskupovaný do vizuálnych zhlukov. Celú stránku potom tvorí niekoľko takýchto zhlukov. Tieto zhluky dokážeme medzi sebou rozlišovať na základe vizuálnych hraníc ako sú napr. prázdne miesto, veľkosť písma alebo farba pozadia. Okrem hlavného obsahu sa v stránke vyskytujú aj prvky ktoré nie sú pre extrakciu informácií relevantné. Sú to napríklad prvky ako *hlavička*, *navigačné menu*, *päta* alebo *reklama*. Ak chceme vyextrahovať časti s obsahom a vyfiltrovať vyššie spomínané „rušivé“ časti (reklamy, navigačné menu), musíme vedieť tieto časti v stránke identifikovať.

Jedným zo spôsobov ako toto docieliť je *segmentácia*. Segmentáciu môžeme definovať ako proces rozdeľovania webovej stránky na vizuálne a sémanticky koherentné časti, ktoré označujeme ako *bloky* [18]. Táto práca sa zameriava na proces automatickej segmentácie, pretože existujú aj pokusy o manuálnu segmentáciu. Pri manuálnej segmentácií sa väčšinou využívajú nástroje pre manuálne anotovanie blokov na webovej stránke. Narozdiel od auto-

matického procesu je ale manuálna segmentácia nevýhodná z hľadiska pracnosti a časovej náročnosti spracovania. Taktiež nesmieme zabúdať aj na chybovosť ľudského faktoru.

Cieľom tejto práce je preskúmať a naštudovať segmentačné metódy a oboznámiť sa s nástrojom Fitlayout, najmä s jeho rozhraním a vnútornými štruktúrami. Z týchto metód potom zvoliť vhodného kandidáta, ktorý sa do systému Fitlayout implementuje. Po úspešnej implementácií sa otestuje funkčnosť a efektívnosť tejto metódy v prostredí Webu.

Práca obsahuje 8 kapitol z toho prvé dve obsahujú abstrakt a krátky úvod do problematiky segmentácie webových stránok. Kapitola tretia poskytuje teoretické poznatky o využití segmentačných metód, prehĺad týchto metód a popisuje taktiež metriky na ich vyhodnocovanie. Štvrtá kapitola popisuje rámec FitLayout a jeho rozhranie. V piatej kapitole je popísaný algoritmus Block-o-Matic, ktorý bol vybraný pre integráciu do systému FitLayout. Šiesta kapitola popisuje implementáciu tohto algoritmu v rozhraní FitLayoutu. Siedma kapitola testuje a porovnáva implementovanú metódu s jej referenčnou implementáciou, ktorá je dostupná v jazyku JavaScript. Posledná ôsma kapitola zhŕňa a zhodnocuje výsledky toho ako moc sa implementácia v rozhraní FitLayout podobá na jej referenčnú implementáciu.

Kapitola 3

Webové stránky a ich segmentácia

Táto kapitola je zameraná na teóriu v oblasti segmentovania webových dokumentov a v úvode rozoberá aj krátky pohľad do procesu jej vykresľovania. Poskytuje teoretický prehľad o využití segmentácie v rôznych oblastiach od prispôsobovania obrazoviek pre mobilné telefony až po extrakciu informácií. Kapitole rovnako popisuje prístupy a stratégie segmentačných algoritmov a tiež rozoberá metriky, ktorými sa dá úspešnosť segmentačnej metódy vyhodnocovať

3.1 Webová stránka a jej vykresľovanie

Webová stránka je dokument prístupný z prostredia Webu. V dnešnej dobe, kedy je rozšíreným štandardom pre prezentovanie webovej stránky kombinácia jazykov HTML5, CSS a JavaScript sa už málokedy stretávame s webovou stránkou ktorá by nebola vizuálne formátovaná a neobsahovala žiadne dynamické prvky ako je reklama alebo animácie. Pre lepšie chápanie segmentácie a segmentačných algoritmov, si rozoberieme proces vykresľovania webovej stránky, najmä aj z toho dôvodu že téma tejto práce sa zaoberá implementovaním vizuálnej segmentácie do rámca FitLayout. To znamená že algoritmus sa bude sústreďovať výhradne na formátovanie, rozloženie a celkovú vizuálnu prezentáciu stŕanky.

- **Analýza HTML**: Webový prehliadač analyzuje HTML kód webovej stránky, aby porozumel jej štruktúre a vytvoril reprezentáciu štruktúry *Document Object Model* (DOM). DOM predstavuje hierarchickú štruktúru HTML elementov vo webovej stránke.
- **Analýza a aplikácia CSS štýlov**: Prehliadač analyzuje CSS kód priradený k webovej stránke a aplikuje štýly na príslušné elementy DOM-u. Tento krok zahŕňa vyriešenie selektorov, výpočet špecifík a výpočet konečných štýlov pre každý element.
- **Výpočet rozloženia**: Prehliadač určuje pozíciu a veľkosť každého elementu na webovej stránke. Rozloženie *anglicky Layout* sa vypočíta na základe aspektov ako je CSS box model elementu, pozície a rozloženia okolitého obsahu v stránke.
- **Kreslenie**: Prehliadač vytvára vizuálnu reprezentáciu webovej stránky kreslením každého elementu zo štruktúry DOM na obrazovku. Tento krok zahŕňa rasterizáciu získaného rozloženia stránky do formy pixelov a aplikáciu formátovania ako je farba textu, ohraničenie elementu alebo farba pozadia.

- **Kompozícia**: Ak webová stránka obsahuje prekryvajúce sa elementy, prehliadač vykonáva kompozíciu pre ich zobrazenie v správnom poradí a aplikuje zmiešavanie farieb (blending). Tento krok zaisťuje správne zoraďovanie (z-ordering) a priehľadnosť elementov.
- **Vykonávanie JavaScript kódu**: Ak webová stránka obsahuje JavaScriptový kód, prehliadač ho vykonáva, aby pridal interaktivitu a dynamické správanie. JavaScript môže manipulovať s DOM-om, modifikovať štýly, spracovávať interakcie od užívateľov a získavať údaje zo serverov.
- **Aktualizácia vykresľovania**: Vždy, keď dochádza k zmenám v štruktúre DOM, v CSS štýloch alebo v rozložení stránky, prehliadač vyvolá aktualizácie vykresľovania, aby hneď reflektoval tieto zmeny na obrazovku. Tento proces zaisťuje, že zobrazená webová stránka zostáva synchronizovaná.

3.2 Čo je to segmentácia

Segmentáciu môžeme definovať ako proces rozdeľovania celej webovej stránky na menšie oblasti, alebo segmenty, na základe ich vizuálnych a štrukturálnych charakteristík. Požadovaná vlastnosť regiónu je, že musí byť vizuálne a sémanticky koherentný [18]. V kontexte segmentácie sa tieto oblasti, resp. segmenty nazývajú bloky. Vo webovej stránke môžu byť bloky identifikované podľa rôznych vlastností, ako sú HTML značky, vizuálne vlastnosti (ako farba, rozloženie, veľkosť a štýl písma), textový obsah alebo kombinácia týchto vlastností.

Vizuálne a sémanticky koherentné segmenty sú oblasti alebo časti webovej stránky, ktoré zoskupujú prvky stránky s podobným vzhľadom a významom. Prvky vo vizuálne koherentnom bloku majú podobné vizuálne vlastnosti, ako sú farba písma, veľkosť písma, farba pozadia alebo rozloženie prvkov vrámci bloku, zatiaľ čo prvky v sémanticky koherentnom bloku majú podobný obsah alebo funkciu, ako sú časti s textom, obrázky alebo navigačné menu.

Napríklad na webe spravodajského portálu tvorí hlavičková sekcia, ktorá obsahuje logo daného portálu a navigačné menu, sémanticky koherentný blok, zatiaľ čo hlavná časť obsahu s novinovými článkami je ďalším sémanticky koherentným blokom. V rámci hlavnej časti obsahu môže byť každý novinový článok vizuálne koherentným blokom, pretože zvyčajne majú podobné rozloženie a formátovanie.

Segmentácia sa aplikuje pri analýze stránky a zvyčajne sa začleňuje ako podúloha alebo medzikrok pri riešení nejakej konkrétnej úlohy a jej cieľom je identifikovať a extrahovať bloky z webovej stránky, ktoré sú relevantné pre konkrétnu úlohu, ako je napríklad extrakcia obsahu a informácií, analýza rozloženia alebo prispôsobenie rozloženia webovej stránky pre konkrétny typ obrazoviek. Využitie segmentácie v rôznych úlohách a oblastiach je podrobnejšie rozobraté v následujúcej podkapitole.

3.3 Motivácia a oblasti využitia segmentácie

V tejto podkapitole si rozoberieme motiváciu používania segmentácie v rôznych oblastiach od extrakcie dát, hlasové prehliadače až po detekciu phishingových mailov.

Prispôsobovanie webu pre malé obrazovky

Dostupnosť webu pre malé obrazovky prenosných zariadení, ako sú napr. tablety a chytré mobilné telefóny, je v dnešnej dobe štandardom. Avšak viac ako dekádu dozadu zaberali majoritnú časť trhu s prenosnými zadriadeniami klasické mobilné telefóny alebo zariadenia PDA (personal digital asistant). V tomto období boli webové stránky prioritne dizajnované pre veľké obrazovky osobných počítačov. Prehliadať kompletnú webovú stránku bolo na týchto zariadeniach teda nemožné kvôli viacerým technickým obmedzeniam. Pamäť týchto zariadení neposkytovala dostatočnú kapacitu na to aby dokázala načítať kompletnú stránku. Navigácia po stránke bola komplikovaná, pretože väčšina klasických mobilných telefónov sa ovládala cez tlačidlovú klávesnicu a dotykové obrazovky neboli veľmi rozšírené. Samotná obrazovka tiež predstavovala problém, pretože kvôli nízkemu rozlíšeniu a obmedzenej farebnej schéme by nebolo možné zobraziť stránku v čitateľnej forme.

Zmienené problémy predstavujú časť z rady dalších problémov, ktoré boli motiváciou pre vznik rôznych riešení v oblasti prispôsobovania webov pre malé obrazovky prenosných zariadení. Niektoré riešenia [22] sa snažili túto problematiku riešiť tak, že vezmú na vstup klasickú webovú stránku, zjednodušia prezentáciu obsahu odfiltrovaním nadbytočných elementov, ktoré nesúvisia s obsahom alebo sú náročné na stiahnutie, a následne celú stránku rozdelia na niekoľko menších podstránok. Podstránky sú vpodstate bloky získané segmentáciou vstupnej webovej stránky, ktoré sú potom vykreslené na obrazovke mobilu samostatne. Ukážka tohto procesu je na obrázku 3.1.

Detegovanie phishingových stránok

V oblasti kybernetickej bezpečnosti sa našlo pre segmentáciu využitie pri prevencií pred phishingom. Jedná sa o známu a rozšírenú podvodnú techniku, ktorá využíva elektronickú komunikáciu pre získanie citlivých údajov (napr. heslo, číslo kreditnej karty) od obeti. Útočník vytvorí falošnú webovú stránku nejakej dôveryhodnej entity a odkaz na ňu odošle obeti cez elektronickú komunikáciu (elektronická pošta, SMS). V praxi túto entitu najčastejšie zastupuje banka, webová služba alebo sociálna sieť. Pri tvorbe falošnéj stránky sa útočník hlavne spolieha na vizálnu prezentáciu a snaží sa celý vizuál stránky napodobniť tak, aby bol čo najmenej rozoznateľný od originálnej stránky. Tu majú priestor pre využitie najmä segmentačné metódy, ktoré segmentujú stránku na základe vizuálnych vlastností.

Jeden zo spôsobov prevencie pred týmto typom útoku je systém [10], ktorý vykonáva detekciu na základe vizuálnej analýzy stránky. Pri analýze sa porovnáva pravá stránka s jej phishingovou kópiou. Obe stránky sa rozdelia pomocou segmentácie na bloky. Podobnosť stránok sa určuje na základe troch metrík: podobnosť na úrovni lokánych vizuálnych vlastností blokov (medzi stránkami sa hľadajú rovnaké páry blokov s podobnými vizuálnymi vlastnosťami), podobnosť na úrovni rozloženia blokov v stránke a celková podobnosť štýlu oboch stránok. V rámci všetkých metrík sa medzi pravou a podozrivou stránkou zistuje koeficient podobnosti, ktorý určuje mieru podozrenia na phishingovú stránku (vyššia hodnota koeficientu znamená vyššiu pravdepodobnosť phishingovej stránky). Systém po analýze prihliada na hodnoty koeficientu a na základe stanovenej prahovej hodnoty sa rozhoduje, či sa jedná o phishingovú stránku alebo nie.

Detekcia duplicitných stránok

Ak úžívateľ pri vyhľadávaní zadá dotaz, očákáva, že mu budú požadované výsledky vrátené v rozumnom čase. Tento problém rieši proces indexovania, počas ktorého prehliadač

(a) Získanie hlavného obsahu stránky *(anglicky content block)* a navigácie

(a) Link block1 page (b) Link block2 page (c) *Campus Tour* content block

(b) Reprezentovanie celej stránky na malej obrazovke po blokoch, ktoré sú prepojené odkazmi

Obr. 3.1: Transformácia webovej stránky pomocou segmentácie do reprezentácie vhodnej pre malú obrazovku mobilného telefónu. Zdroj: [22]

zbiera, analyzuje a ukladá informácie o webových stránkach. Stránky sú tak po zadaní vyhľadávacieho dotazu užívateľom získané rýchlo a efektívne. V rámci webu je bežným javom výskyt viacerých kopií toho istého obsahu. Toto je spôsobené napr. zrkadlením stránok na viac serverov kvôli vyrovaniu záťaže. Odhaduje sa že až 40% stránok na webe sú duplicítne stránky. Duplicita ale predstavuje pre vyhľadávače problém, pretože sa defakto niekoľkokrát opätovne indexuje stránka s rovnakým obsahom.

Jeden z konceptov, ktoré využíva väčšina algoritmov na detekciu duplicítnych stránok, sú tzv. *shingles* [14]. V stručnosti sa jedná o unikátny sled n po sebe idúcich znakov alebo slov v dokumente. Shingles sa pri detekcií duplicity využívajú na zistenie miery podobnosti obsahu dvoch dokumentov. Zo všetkých shingles sa vytvára podpis pre celý dokument. Prebieha to tak, že sa vypočítava hash pre každý shingle z množiny a najmenšia hodnota hashu sa uloží do podpisu. Tento proces prebehne N iteracií. Potom v prípade ak sú dokumenty duplicítne, resp. podobné tak ich podpisy obsahujú rovnaké shingles.

Ak sa zameriame na webové dokumenty tak okrem hlavného obsahu sa na stránke vyskytuje už spomínaný „šum“ v podobe navigačného menu, užívateľských komentárov alebo reklamy. Pri detekcií duplicity algoritmy pracujú s celým obsahom stránky a tak tento šum môže ovplyvniť tvorbu podpisov natoľko, že môžu byť niektoré stránky s rovnakým obsahom ale iným šumom označené za rozdielne. Preto je vhodné pre zlepšenie detekcie

zapojiť segmentáciu pri predspracovaní stránky a identifikovať tak šum a hlavný obsah, aby sa minimalizovala tvorba šumom skreslených podpisov.

Extrakcia informácií

Webové stránky sú rozsiahlym zdrojom informácií pre mnoho aplikacií, a to najmä v oblasti vyhladávačov a doporučovacích systémov. Neštruktúrovaná forma webových stránok, ktoré obsahujú kombináciu textu, obrázkov a ďalšieho multimedálneho obsahu, predstavuje výzvu pre automatické vyhladávanie relevantných informácií.

Tejto problematike sa venuje práve proces extrakcie informácií a jeho hlavným cielom je premeniť neštruktúrované alebo pološtrktúrované dáta na štruktúrované dáta, ktoré sú ďalej analyzované alebo uložené do databázy. Segmentácia pomáha zvyšovať presnosť extrakcie informacií, čož ďaľej ovplyvňuje výsledky algoritmov použitých pri dolovaní dát. Webové stránky zahŕňajú veľké množstvo dát ale nie všetky sú relevantné, alebo zaujímavé pre ukladanie a ďalšiu analýzu. [3] rozdelí stránku pomocou navrhnutej segmentačej metódy, identifikuje role týchto segmentov a následne sa zameriava na tie, ktoré sú vhodné na ťažbu dát.

Archivovanie

Web je prostredie ktoré podlieha neustálym zmenám. Webové stránky časom aktualizujú svoj obsah, funkcionalitu, technológie alebo dizajn. Ak by sme chceli zistiť aké témy boli aktuálne na nejakom našom obľúbenom spravodajskom portáli pár rokov dozadu alebo by sme chceli sledovať ako sa časom menil dizajn stránky tak by sme prevdepodobne využili digitálnu archivačnú službu *Wayback Machine* [1]. Táto služba udržuje rozsiahlu kolekciu webových stránok a webových lokalít zachovaných v čase a umožňuje nad touto kolekciou vykonávať temporálne vyhľadávanie. Užívateľ si dokáže pomocou kombinácie URL adresy a konrétneho dátumu zobraziť archivovanú verziu webovej lokality alebo webovej stránky z určitého obdobia. Obecný princíp fungovania webových archívov je taký, že pomocou špecializovaných botov (tzv. spider, web crawler) neustále prehľadáva Web. Títo boti prehľadávajú odkazy vo webových stránkach a dochádza tak k postupnému objavovaniu nových webových stránok. Pri objavení novej webovej stránky sa z nej vytvorí záznam a to tak, že sa získajú všetky zdroje potrebné k jej vykresleniu v prehliadači. Kľúčové zdroje zahŕňajú súbory typu HTML, CSS, JavaScript a obrázky. Archív zdroje ukladá do databázy a dokáže z nich reprodukovať stránku do pôvodného formátu. Aby bol archív stále aktuálny a aby obsahoval najnovšie záznamy je nutné aby boti navštevovali webové stránky opätovne. Neustále ukladanie záznamov pri každej novej zmene na webovej stránke je ale z hľadiska obmedzených výpočetných zdrojov nereálne. Archívy sa preto snažia odhadnúť s akou frekvenciou majú webovú stránku navštevovať. Túto frekvenciu sa archív snaží odhadnúť podľa počtu zmien na webovej stránke za určité časové obdobie. Tu ale môže dochádzať k redundantnému sťahovaniu stránok z dôvodu častých zmien nepodstatného obsahu ako je napríklad neustála aktualizácia reklám alebo inzercie.

[2] navrhuje riešenie tohto problému a to tak, že sa zameriava na detekciu relevantných zmien na stránke, čím sa snaží eliminovať časté sťahovanie stránok s nepríliš významnými zmenami. Pri detekcií zmien sa riešenie zameriava na pozícu blokov v rámci stránky, typy operácií nad elementami (presun, odstánenie alebo vloženie elementu) a na počet operácií/zmien vykonaných v rámci samostatného bloku. K rozdeleniu stránky na bloky je tu aplikovaná vizuálna segmentácia a segmentačný algoritmus *Visual-page segmentation* (skrátene VIPS)[5], ktorý je kvôli potrebám detekcie modifikovaný a extrahuje pre každý blok

viac informácií. Celá štruktúra, ktorú modifikovaný VIPS algoritmus vytvorí, je ukladaná do špeciálneho XML dokumentu *Vi-XML*. Riešenie k výslednej detekcií používa navrhnutý algoritmus Vi-DIFF, ktorý deteguje relevantné zmeny medzi dvoma Vi-XML dokumentami.

Prehliadače s hlasovým rozhraním

Konvenčné prehliadače obvykle neposkytujú rozhranie pre zrakovo postihnuté osoby, ktoré by umožňovalo efektívne prehliadanie Webu. Časté riešenie v tomto prípade poskytujú čítače obrazovky. Takéto riešenie ale nemusi poskytovať zrakovo postihnutej osobe komfort pri prehliadaní a to z dôvodu, že čítače obrazovky čítajú obsah stránky sekvenčne. To znamená, že sa pri prehliadaní prečíta celá stránka a aj s informáciami ktoré užívateľa nezaujímajú. Prehliadanie sa tak zbytočne predlžuje. V rámci práce [13] bol vytvorený hlasový prehliadač CSurf, ktorý berie do úvahy kontext prehladávania. To znamená, že ak úžívateľ pri prehliadavaní klikne na odkaz, tak je okolitý kontext daného odkazu analyzovaný, a za pomoci strojového učenia, použitý k identifikácií relevantných informácií na následujúcej stránke. Užívateľ je pri prechode z jednej stŕanky na druhú nasmerovaný priamo k segmentu s požadovaným obsahom a prehliadač číta užívateľovi len relevantné informácie. Architektúra CSurf-u tvorí rozšírenie nad hlasovým prehliadačom HearSay a vrámci tohto rozšírenia pridáva modul pre analýzu kontextu. Aby bolo možné určiť hranice okolitého kontextu odkazu,tak je súčasťou tohto modulu aj segmentácia. Táto časť je pomenovaná ako geometrické zhlukovanie a prebieha pomocou algoritmu FindBlocks. FindBlocks pracuje s uzlami (rámcami) štruktúry frame tree, ktorú poskytuje renderovadlo prehliadača Mozilla a ktorá v sebe drží obsah celej webovej stránky. Cieľom tohto algoritmu je získať množinu maximálnych sémantických blokov. Maximálny sémantický blok tu predstavuje najväčší možný zhluk obsahujúci sémanticky súvisiace informácie čo v kontexte segmentácie poznáme ako klasický blok.

Ďalšie oblasti

Celkovo má segmentácia webových stránok mnoho aplikácií v rôznych oblastiach, a aj keď sa táto kapitola zameriava na niektoré z dôležitých oblastí, kde je segmentácia dôležitou podúlohou, stále existuje mnoho ďalších oblastí, ako napríklad detekcia užívateľského záujmu na stránke [12], vyhodnocovanie vizuálnej kvality stránky [21] alebo automatická anotácia HTML dokumentov [16] v tejto podkapitole neboli rozobraté z dôvodu rozsiahlosti tejto problematiky.

3.4 Klasifikácia segmentačných metód

V následujúce podkapitoly obsahujú klasifikáciu segmentačných metód a rozdelenie základných stratégií prístupu algritmov k pri segmentácií stránky. V týchto podkapitolách som vychádzal z informácií v [18].

3.4.1 Základné metódy

Metódy z tejto časti sú klasifikované ako základné metódy a to z toho dôvodu, že pracujú len s jednou charakteristikou webovej stránky. Tieto charakteristiky sú text, obraz stránky alebo HTML dokument. Na základe týchto charakteristík sú metódy rozdelené do troch kategórií. Sú to kategórie Text based, DOM based a Image based, ktoré sú detailnejšie rozobraté v podsekciách nižšie.

Text-based

Textovo orientované segmentačné metódy pracujú s textovým obsahom stránky a využívajú analýzu textového obsahu webovej stránky za účelom identifikovania jej štruktúry a významných častí. Pri analýze sa využívajú štatistické metódy, ako je hustota textu a hustota odkazov vo webovej stránke.

Tento typ segmentačných metód môže extrahovať text buď zo zdrojového HTML kódu, alebo po vykreslení stránky zo štrukúry DOM. Zástupcami týchto metód sú napríklad [20] a [8]. [20] Pracuje so štrukúrou DOM, prechádza každý jej uzol a pre každý uzol zisťuje štatistické informaćie o počte textových znakov a HTML značiek, ktoré sa nachádzajú v danom uzle.

Obe metódy pristupujú k meraniu hustoty textu dvoma rozličnými spôsobmi. Metóda [20] definuje Hustotu textu uzla ako pomer počtu všetkých znakov k počtu všetkých tagov v DOM podstrome, zatiaľ čo v metóde Block Fusion [8] je hustota textu zisťovaná pomocou zalamovania textu cez konštantnú šírku riadku.

[20] naviac definuje aj kompozitnú hustotu textu ktorý, je pridaný z toho dôvodu, že metóda berie v stránke časti s hypertextovými prepojeniami ako šum. Kompozitná hustota textu teda ešte zapája štatistické informácie o počte všetkých textových znakov v hypertextových prepojeniach v podstrome DOM-u pre daný uzol a informácie o počte všetkých hypertextových značiek v podstrome DOM-u pre daný uzol. Táto segmentačná metóda zakladá na tom, že časti s hlavným obsahom budú mať v stránke najvyššiu hodnotu hustoty textu. Metóda taktiež spolieha na to, že pridanie kompozitnej hustoty textu umožňuje rozlišovať medzi dôležitými uzlami, ktoré obsahujú veľa textu a málo hypertextových odkazov (vysoká hodnota hustoty) a s menej podstatnými uzlami, ktoré obsahujú veľa hypertextovych prepojení a málo textu (nízka hodnota hustoty).

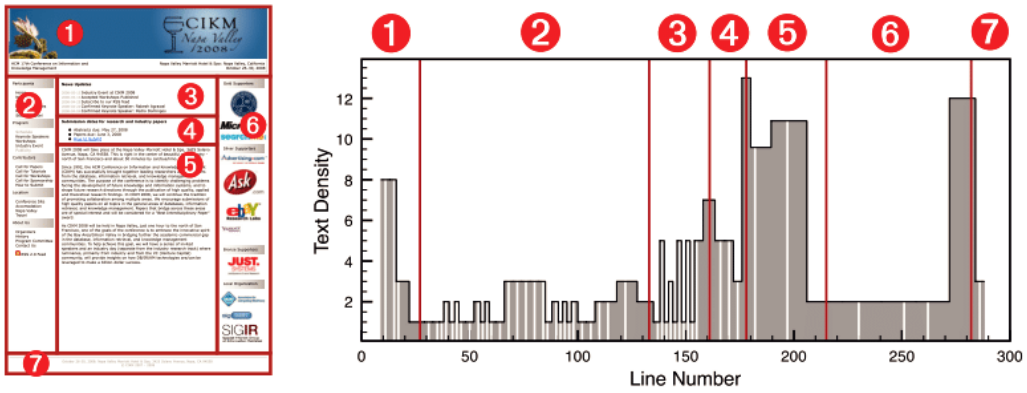
Metóda Block Fusion spočíva v detekcií medzier medzi textovými častami v histogramovej reprezentácii hustoty textu, ktorá je získana z HTML kódu. V histograme sa potom zameriava na susedné bloky s významnými skokmi, resp. zmenami v hustote textu. Susedné bloky sa postupne spájajú, čím vznikajú väčsie bloky, ktoré potom určujú hlavný obsah na stránke, tak ako je to znázornené na obrázku 3.2. Na určenie, kedy spojiť dva susedné bloky, metóda zisťuje rozdiel v hustotách textu susediacich blokov a na podľa nastavenej prahovej hodnoty rozhodne či sa dané bloky spoja.

Tento typ metód sa zameriava len na text v stránke a ignoruje ostatné charakteristiky ako sú obrázky, formátovaný text a farba pozadia, preto existujú stránky na ktorých tieto metódy zlyhávajú.

DOM-based

DOM-based prístup je technikou segmentácie webových stránok, ktorá analyzuje webovú stránku na základe zdrojového kódu stránky. Tento prístup rozdeľuje a segmentuje stránku na základe jej preddefinovanej syntaktickej štruktúry, ktorá je definovaná HTML značkami.

Niektoré metódy z tejto kategórie pristupujú k segmentácií tak, že sa snažia vypočítať vzdialenosť medzi značkami v štruktúre HTML súboru. [7] definuje elementy, ktoré označuje ako nositeľov obsahu. Jedná sa napríklad o značky `<A>`, `<IMG>` alebo o čisto textové dáta. Následne sa medzi dvoma obsahovými elementmi zisťuje vzdialenosť na základe počtu a hĺbky zanorenia ostaných značiek, ktoré sa medzi týmito dvoma obsahovými elementmi nachádzajú. Segmentácia prebieha delením počiatočného bloku, ktorý sa iteratívne separuje na pozíciach, kde hodnota vzdialenosti prekročí dynamicky odhadnutú prahovú hodnotu.



Obr. 3.2: Detekcia blokov v histograme hustoty textu pomocou algoritmu Block Fusion[8] (vpravo) a vyobrazenie zdetegovaných blokov vo vizuálnej prezentácií stránky (vľavo). Zdroj: [18]

K vyriešeniu možných problémov pri segmentácií metóda zapája aj sadu heuristických pravidiel, ktoré sa orientujú podla jednotlivých HTML značiek.

Metóda [9] bola navrhnutá v rámci pokusu o zlepšenie kvality vyhľadávania na relevantných informácií na Webe. Stránka sa segmentuje do tzv. mikroinformačných jednotiek. Metóda vytvára z HTML značiek stromovú štruktúru, ktorá sa podobá na štruktúru DOM. Pre segmentovanie sú potom aplikované dve heuristiky, ktoré agregujú uzly stromu do segmentov. Spájajú sa nadpisy s podnadpismi a spájajú sa susedné odstavce textu.

Výhodou tohto prístupu je, že analýza prebieha veľmi rýchlo, ale samotné značky na stŕanke nemusia poskytnúť dostatočné informácie k uspokojujúcej segmentácií. Jeden z dôvodov je to, že sa nezohľadňuje vizuálna prezentácia obsahu, a tak sa môže stať že dva bloky, ktoré spolu susedia po vizuálnej stŕanke budú v štruktúre HTML dokumentu od seba ďaleko a tak nedôjde k ich spojeniu.

Image-based

Tento typ metód k segmentácií nepotrebuje zdrojový kód stránky ani štruktúru DOM. Metódy z tejto kategórie segmentujú stránku na základe snímky, ktorá obsahuje obrazový záznam celej stránky po vykreslení. Pre detekciu objektov z obrazových dát existuje viacero efektívnych techník, ktoré sú nasadzované aj v prostredí "prirodzených" obrazových dát, ako sú kamerové záznamy alebo fotky. V prípade snímky webovej stránky je táto detekcia jednoduchšia. Narozdiel od "prirodzených" obrazových dát obsahuje webová stránka objekty s výraznými hranami a farebnými prechodmi, a tieto objekty sú pre algoritmy spracovania obrazu najlepšie zdetegovateľné.

Jeden zo zástupcov týchto metód navrhuje segmentáciu s pomocou analýzy hrán [17]. Pomocou detekcie dlhých vertikálnych a horizontálnych hrán sú na stránke nájdené obdĺžniky, ktoré vyznačujú jednotlivé vizuálne prvky v rozložení stránky. Potom sú vybrané tie obdĺžniky, ktoré sa nenachádzajú v žiadnom inom obdĺžniku na obrázku, resp. stránke. Po tejto fáze sa algoritmus pokúsi ešte dodatočne spojiť niektoré vhodné oblasti do väčsieho celku. V prvom priechode algoritmu sa zdetegujú najväčsie a pravdepodobne najhlavnejšie časti do akých je stránka rozložená. Tento proces pokračuje rekurzívne v obdĺžnikoch, ktoré boli nájdené v prvom priechode. Týmto spôsobom je možné sa dostať až na úroveň jednotlivých prvkov, ako textové oblasti, obrázky, videá, tlačidlá atď. Metóda nedokáže

genericky zdetekovať všetky prvky na stránke, pretože prvky ako radio button alebo check box nemajú hranatý tvar. Na tento prípady metóda deteguje zvlášť.

Výhoda tejto kategórie metód je, že sa dá použiť nielen na webové stránky ale aj na klasické dokumenty, ktoré boli do digitálneho obrazu prevedené naskenovaním. Na druhú stranu, nedostatkom týchto metód je nevyužívanie štruktúry DOM alebo aspoň HTML dokumentu, pretože sa z nich dá lepšie zistiť sémantický význam prvku. V prípade textu napríklad vieme pomocou značky určiť či sa jedná o nadpis alebo klasický textový blok.

3.4.2 Hybridné metódy

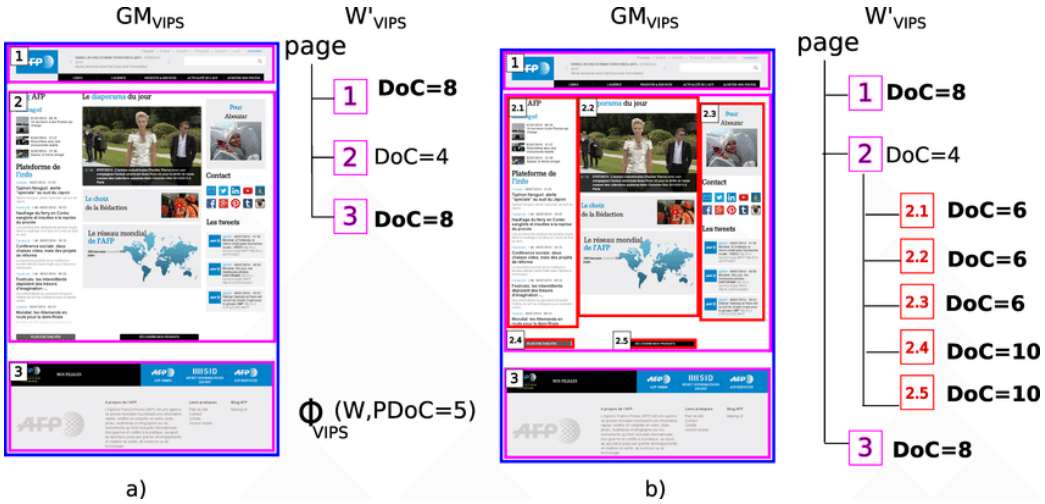
Metódy z tejto časti sú označované ako hybridné pretože sa nezameriavajú len na jednu charakteristiku webovej stránky ale pri segmentácií sledujú viac charakteristík. Tento prístup umožňuje nahliadnuť na stránku z viacero dimenzií naraz a to môže dopomôcť k lepšej segmentácií. V následujúcich podkapitolách sú rozabraté dve kategórie hybridných metód.

Vison-based

Metódy z tejto kategórie rozdeľujú stránku na základe vizuálnej prezentácie jej obsahu. Počas segmentácie sa sústredia na priestorové a vizuálne indikátory, ako napríklad farba textu, veľkosť textu, geometrické vlastnosti vykreslených elementov alebo vzdialenosť medzi dvoma elementmi. Tieto metódy sa zakladajú na ľudskom vnímaní webovej stránky. Snažia sa simulovať to, ako stránku vníma bežný úžívateľ a ako kategorizuje a rozdeľuje jednotlivé prvky stránky. Metódy počas segmentácie vychádzajú z dát, ktoré si extrahujú pri priechode štruktúrou DOM po kompletnom vykreslení stránky.

Jeden z najznámejších a najpoužívanejších metód z tejto kategórie je algoritmus VIPS [5]. Na začiatku algoritmus predpokladá, že celý dokument je jedna veľká oblasť. Potom prechádza štruktúru DOM po úrovniach a analyzuje každý uzol, aby zistil, či ho možno považovať za podoblasť. Tieto podoblasti sú označované ako vizuálne bloky. Pre extrakciu vizuálnych blokov sa používa 13 heuristík. Pri každom získanom bloku sa počíta parameter DoC - stupeň koherencie *(anglicky Degree of coherence)*, podľa ktorého sa neskôr určuje, či je potrebné daný vizuálny blok znovu rozdeliť. Tento parameter vlastne určuje granularitu výslednej segmentácie. Po zistení množiny vizuálnych blokov algoritmus vypočíta množinu separátorov. Separátory sú obdĺžniky, ktoré sa nedotýkajú žiadneho z vizuálnych blokov. Separátor predstavuje prázdnu oblasť v dokumente. Každý separátor má priradenú váhu, ktorá je odvodená od vizuálnej odlišnosti medzi vizuálnymi blokmi, ktoré separátor rozdeľuje. Susedné oblasti, ktoré majú separátor, ktorého váha je menšia ako predom určený práh, sa zlúčia. Vzniknú tak nové vizuálne bloky, pre ktoré sa opätovne vypočíta DoC a rozhodne sa či sa bude v celom procese delenia blokov pokračovať v ďalšej iterácií. V prípade ak je už nie je čo deliť, algoritmus vracia hierarchickú stromovú štruktúru vizuálnych blokov, tak ako je to znázornené na 3.3.

Ďalšou z metód z tejto kategórie, ktorá sa pomaly dostáva do popredia je metóda Block-o-Matic (skrátene BoM) [19]. Segmentácia prezentovaná v troch štruktúrach a to štruktúra DOM, obsahová štruktúra a logická štruktúra. Každá z nich predstavuje odlišný pohľad na segmentáciu. Konečná segmentácia je prezentovaná v logickej štruktúre. O tejto metóde detajlnejšie pojednáva ďalšia kapitola.



Obr. 3.3: Segmentácia stránky pomocou algoritmu VIPS do hierarchickej štruktúry vizuálnych blokov. Zdroj: [19]

Graph-based

Predchádzajúce metódy sa zväčša spoliehali na rôzne heristiky alebo sadu heuristických pravidiel. Táto kategória algoritmov sa snaží do problematiky segmentácie zapojiť formálne prístupy ako je využitie grafových algoritmov a štruktúr. Častý postup je že sa pred samotnou segmentáciou mapuje štruktúra DOM na neorientovaný graf s hranami, ktoré majú pridelené váhy. Segmentácia potom prebieha tak, že na základe váh medzi uzlami algoritmus jednotlivé dvojce uzlov zhlukuje do nového uzlu. Tento proces je iteratívny a končí až keď sa graf dostane do stavu, kedy mu prednastavené obmedzujúce podmienky nedovolia ďalej uzly zhlukovať. Typickým príkladom obmedzujúcej podmienky je napríklad prahová hodnota pre váhu hrany.

Konkrétnou reprezetáciou metód z tejto kategórie je [11]. Algoritmus po vykreslení stránky získa štruktúru DOM. Z tejto štruktúry sa zostaví neorientovaný graf s váhami hrán. Každý uzol v grafe predstavuje vizuálny blok. Vizuálny blok je každý uzol v stromovej štruktúre DOM, ktorý má za bezprostredných následovníkov uzly s textom, hypertextovým prepojením alebo s obrázkami. Hrany medzi uzlami v grafe, resp. vizuálnymi blokmi sú vypočítané tak, že sa pre daný vizuálny blok nájdu priestorovo najbližšie vizuálne bloky. Tie sa potom prepoja hranami s daným uzlom. Váhy hrán sú získané na základe podobnosti ciest v stromovej štruktúre DOM medzi dvoma uzlami, ktoré predstavujú vizuálne bloky. Týmto sa algoritmus snaží docieliť to, že dvojica uzlov, ktoré sa nachádzajú v rovnakom podstrome budú mať na hranách menšiu váhu hrany než dvojica uzlov nachádzajúcich sa v dvoch rôznych vetvách stromu DOM. Nad takto zostaveným grafom prebieha spájanie dvoch vizuálnych blokov na základe váhy ich hrany. Pre celý graf sa nastaví prahová hodnota a ak už žiadna hrana v grafe nepresahuje túto hodnotu, tak proces segmentácie končí.

3.5 Stratégie segmentačných metód

Sú dva spôsoby akým segmentačný algoritmus extrahuje bloky z webovej stránky. Extrakcia blokov prebieha buď smerom zhora-nadol alebo zdola-nahor. Existujú ah segmentačné algoritmy, ktoré používajú obe stratégie.

3.5.1 Stratégia zhora-nadol

V stratégii zhora-dole sa najprv celá webová stránka označí ako blok a potom sa rozdelí na niekoľko nových veľkých blokov. Tieto bloky väčšinou definujú hlavné rozloženie stŕanky, ako je hlavička, päta a hlavný obsah. Potom sa nové bloky iteratívne delia na menšie oblasti až kým sa nedosiahne požadovaná úroveň granularity. Jedným z predstaviteľov tejto stratégie segmentácie je už spomínaný algoritmus VIPS [5]. Ten najprv nájde veľké vizuálne bloky, ktorým priradí hodnotu DoC. Po nájdení separátorov a vytvorení hierarchickej štruktúry z vizuálnych blokov sa na základe podmienky $DoC > PDoC$ kontroluje každý blok a ak blok spĺňa túto podmienku, tak prebehne opätovné delenie.

3.5.2 Stratégia zdola-nahor

Pri stratégii zdola-nahor segmentácie sa webová stránka najprv rozdelí na malé bloky, ako sú slová, riadky a odseky. Potom sa susedné bloky zlúčia do väčších blokov na základe podobnosti obsahu, rozloženia a štýlu, tak ako je to znázornené na obrázku 3.4. Túto stratégiu používajú algoritmy, ktoré transformujú reprezentáciu stŕanky na grafovú reprezentáciu, Algoritmus spomínaný v predchádzajúcej sekcií [11] využíva túto stratégiu a po rozdelení stŕanky na elementárne vizuálne bloky, spája tieto bloky na základe váhy hrany medzi dvoma uzlami, resp, vizuálnymi blokmi.

3.6 Vyhodnocovanie segmentačných metód

Aby bolo možné zistiť, aký spoľahlivo dokáže segmentačná metóda rozdeliť stránku na požadované časti, je nutné mať nejaký spôsob, aký sa dá daná segmentačná metóda zhodnotiť a prípadne porovnať s ďalšími segmentačnými metódami. Hodnotiť segmentačné metódy môžeme hodnotiť dvoma spôsobmi [18]:

- **Analytické hodnotenie** analyzuje postup a jednotlivé kroky segmentačného algoritmu, kedy sa sústreďuje na princípy jeho návrhu, jeho vlastnosti a spôsob akým pristupuje k segmentácií (vizuálna, textová apod.).
- **Empirické hodnotenie** posudzuje výsledky segmentácie aplikovaním segmentačnej metódy nad množinou dát. Potom je kvalita segmentácie pre jednotlivé výsledky segmentácie ohodnotená ľudským faktorom. Druhým spôsobom empirického hodnotenia je to, že množina dát obsahuje referenčnú segmentáciu (*anglicky ground truth*). Táto množina dát je už vopred osegmentovaná skupinou ľudí. Pri empirickom vyhodnocovaní sa používajú metriky. Jednou z najpoužívanejších metrík je *precision*, *recall* a *F1* skóre.

3.6.1 Metriky Precision, Recall a F1 skóre

Metriky ako precision, recall a F1 skóre sa bežne používajú pri metódach z oblasti získavania informácií a strojového učenia pri zisťovaní ich efektivity a výkonu. Medzi úlohy, ktoré sa hodnotia týmito metrikami sú napríklad klasifikácia textu a klasifikácia obrazov.

Metrika precision

Precision je mierou presnosti správne extrahovaných blokov. Vypočíta sa ako podiel prieniku všetkých extrahovaných blokov s referenčnými blokmi(*ground truth* 3.5) ku všetkým

extrahovaným blokom. *Precision* poskytuje informáciu o tom ako veľmi sa výsledky segmentácie prekrývajú s blokmi, ktoré boli anotované ľudským faktorom. Inými slovami, koľko z nájdených blokov je správnych.

$$Precision = \frac{|\{correct\ blocks\} \cap \{found\ blocks\}|}{|\{found\ blocks\}|} \tag{3.1}$$

Metrika recall

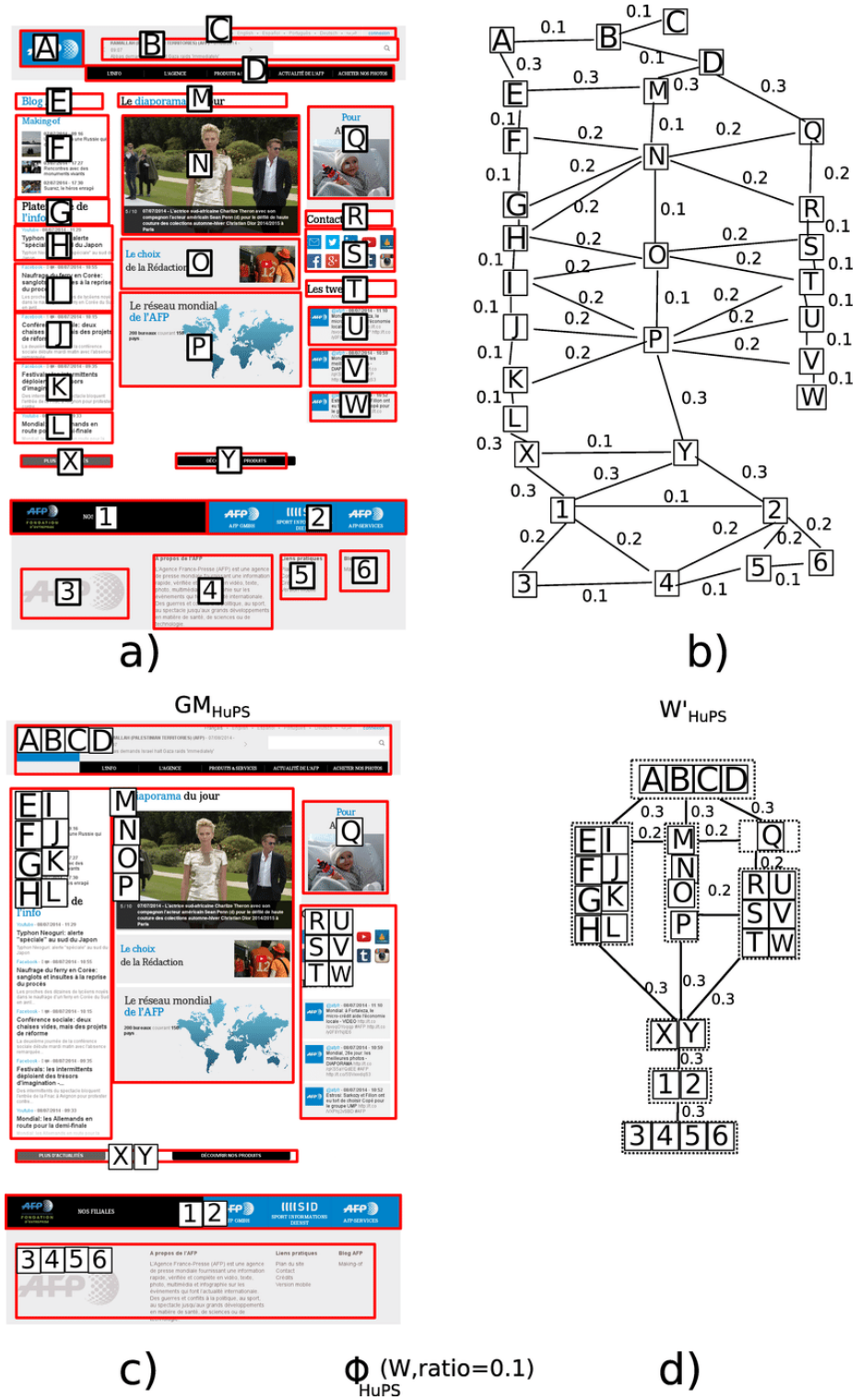
Recall, predstavuje senzitivitu správne extrahovaných blokov. Hodnotí schopnosť metódy nájsť všetky bloky z manálnej referenčnej segmentácie (*ground truth*). Vypočíta sa ako podiel prieniku referenčných blokov a extrahovaných blokov ku referenčným blokom. Metrika *ecall* ukazuje, koľko referenčných blokov je správne identifikovaných.

$$Recall = \frac{|\{correct\ blocks\} \cap \{found\ blocks\}|}{|\{correct\ blocks\}|} \tag{3.2}$$

Metrika F1 skóre

F1 skóre kombinuje hodnoty metrík *precision* a *recall* a vyjadruje harmonický priemer medzi nimi.

$$F_1 = \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{3.3}$$



Obr. 3.4: Ukážka prístupu zdola hore, ktorý využíva segmentačný algoritmus z [8]. Zdroj: [18]

(a) Segmentácia získaná anotovaním častí stránky, ktorú manuálne vykonali návštevníci webovej stránky

(b) Referenčná segmentácia, ktorá bola získaná zjednotením všetkých manuálnych segmentácií z webovej stránky z obrázku vyššie.

Obr. 3.5: Proces vytvárania referenčnej (*ground truth*) segmentácie s ktorou sa porovnáva segmentačný algoritmus

Kapitola 4

Rámec FitLayout

Aplikačný rámec FitLayout [15], slúži ako nástroj pre spracovanie a analýzu webových stránok a PDF dokumentov. Poskytuje robustné aplikačné rozhranie v jazyku Java. Rozhranie obsahuje sadu tried, metód a štruktúr, ktoré umožňujú reprezentovať a spracovávať webovú stránku alebo PDF dokument v prostredí Javy. Rámec je tiež rozšírený o aplikačné rozhranie REST, ktoré umožňuje používanie tohto rámca aj v iných jazykoch. K dispozícií je aj rozhranie príkazoveho riadku.

Táto kapitola popisuje architektúru a jednotlivé aspekty tohto aplikačného rámca, pretože v jeho rozhraní bude implementovaný segmentačný algoritmus *Block-o-Matic* [19].

4.1 Artefakty

Rámec FitLayout pracuje s artefaktmi. Sú to serializovateľné výstupy jednotlivých fáz analýzy vstupnej webovej stránky. S artefaktmi je možné pracovať interne použitím aplikačnérho rozhrania v Jave alebo ho spracovať ako externý súbor, ktorý vychádza z nástroja po prejdení procesu analýzy. Artefakt môže byť vytvorený buď kompletne od základu, napríklad tým že sa vykreslí stránka na základe poskytnutej URL alebo môže byť vytvorený odvodením od rodičovského artefaktu využitím niektorej zo služieb analýzy ako napríklad aplikovaním segmentácie na už vykreslenú stránku. V nástroji FitLayout sú artefakty vytvárané prostredníctvom tzv. artifact service. Momentálny stav nástroja poskytuje tieto artefakty:

- **Stránka** v sebe uchováva popis vizuálnej štruktúry dokumentu. Ako už bolo spomenuté, tento artefakt môže vzniknúť úplne od základu vykreslením novej stránky alebo spracovaním iného vykresleného dokumentu. Vykreslený dokument je artefaktom reprezentovaný ako strom boxov. Každý box v strome je vygenerovaný vykresľovacím jadrom zo štruktúry DOM danej webovej stránky a jeho vizuálne vlastnosti sú definované na základe formátovacieho modelu CSS.
- **Strom oblastí** v sebe uchováva výsledok segmentácie dokumentu. Artefakt v sebe uchováva strom vizuálnych oblastí. Každá vizuálna oblasť vyznačuje vizuálny blok, ktorý bol v dokumente detekovaný. Vzniká aplikáciou jednej z dostupných segmentačných metód v rámci na vykreslenú stránku, alebo po aplikácií segmentačnej metódy na ďalší strom oblastí. Strom vizuálnych blokov je hierarchický, ale kedže niektoré segmentačné metódy vytvárajú plochú, nehierarchickú segmentáciu (ako napríklad

metóda BCS), tak sa v týchto prípadoch vytvára strom s dvomi úrovňami, v ktorom je v prvej úrovni umelý koreňový uzol a v druhej úrovni sú už len listové uzly.

- **Strom logických oblastí** reprezentuje výsledok logického interpretovania stromu oblastí. Obsahuje stromovú štruktúru zostavenú z logických oblastí. Logická oblasť zoskupuje skupinu vizuálnych oblastí, ktoré spolu tvoria sémantickú entitu.
- **Množina textových blokov** reprezentuje množinu všetkých textových blokov extrahovaných z vizuálnych oblastí v stránke pomocu metód ako je napríklad rozpoznávanie pomenovaných entít (NER). Každý textový blok predstavuje obdĺžnikovú oblasť, ktorá obsahuje reťazec textu.

4.2 Aplikačné rozhranie v jazyku Java

Aplikačné rozhranie v jazyku Java je najrozsiahlejšie zo všetkých dostupných rozhraní, ktoré rámec FitLayout poskytuje. Pomocou tohto rozhrania sa dá pracovať s artefaktmi, využívať ich metódy a služby. Je možné si tiež vytvoriť vlastnú implementáciu rozhraní. V rámci tohto rozhrania si môžme napríklad zavolať službu jednoho z dostupných vykresľovacích jadier. Rámec momentálne poskytuje tri služby vykresľovacích jadier z toho dve `CSSBoxTreeProvider` a `PuppeteerTreeProvider` získavajú vykresľenú stránku. Tretia služba `PDFBoxTreeProvider` získava PDF dokumenty. Ukážka kódu pre získanie stránky pomocou vykresľovacieho jadra Puppeteer je nižšie.

```
URL url = new URL("http://...");
var renderer = new CSSBoxTreeProvider(url, 1200, 800);
renderer.setIncludeScreenshot(false);
Page page = renderer.getPage();
```

V prípade segmentačných metód je v rámci momentálne možné použiť tri implementované služby a to `BasicSegmProvider`, `VipsProvider` a `BCSProvider`. Nižši v kóde je názorne ukázanie použitie segmentačného algoritmu `VIPS` nad získaným artefaktom, ktorý reprezentuje vykreslenú stránku.

```
var vips = new VipsProvider();
vips.setPDoC(9);
AreaTree atree = vips.createAreaTree(page);
```

4.3 Rozhranie príkazového riadku

Rámec FitLayout umožňuje pomocou rozhrania príkazového riadku analýzu dokumentov priamo z prostredia terminálu ako samostatne sustiteľný program. Rozhranie príkazového riadka taktiež umožňuje zapojiť rámec FitLayout v multiplatformovom prostredí. Je tak spracovávať a analyzovať webové stránky a dokumenty s použitím iných jazykov ako je Java. Avšak rozhranie príkazového riadku neposkytuje tak rozsiahle rozhranie ako vyššie popísané aplikačné rozhranie. Dokumentácia rozhrania popisuje sadu následujúcich príkazov:

- `RENDER` *Vykreslí stránku a uloží jej reprezentáciu do artefaktu*
- `SEGMENT` *Vykoná zvolenú segmentačnú metódu nad artefaktom a výsledok segmentácie uloží do artefaktu*

- `EXPORT` *Exportuje posledný vytvorený artefakt (záleží od reťazenia príkazov RENDER A SEGMENT). Artefakt je možné uložiť vo formátoch XML, Turtle, HTML, PNG nebo PNGi.*
- `USE` *Otvorí lokálny alebo vzdialený repozitár pre uloženie alebo načítanie artefaktu*
- `LIST` *Ukáže všetky artefakty v repozitári*
- `LOAD` *Načíta artefakt na základe poskytnutého identifikátoru IRI*
- `STORE` *Ukladá exportovaný artefakt do repozitára*

Jednotlivé príkazy je možné za seba skladať. Keď chceme napríklad získať len artefakt, ktorý reprezentuje vykreslenú stránku z URL `http://cssbox.sf.net` a následne tento artefakt uložiť do XML dokumentu, musíme volať príkazy programu v prostredí príkazového riadku následovne:

```
java -jar FitLayout.jar \
       RENDER -b puppeteer http://cssbox.sf.net \
       EXPORT -f xml
```

Ak by sme chceli aplikovať v prostredí príkazového riadku aj analýzu stránky využitím segmentačnej metódy VIPS s parametrom požadovanej granularity pDoC a uložiť výsledky tejto analýzy do formátu XML, tak zreťazíme za seba príkazy pre vykreslenie stránky `RENDER` a príkaz pre vykonanie segmentácie `SEGMENT`. Pri segmentácií volíme metódu pomocou parametra `-m`.

```
java -jar FitLayout.jar \
       RENDER -b puppeteer http://cssbox.sf.net \
       SEGMENT -m vips -O pDoC=9 \
       EXPORT -f xml
```

4.4 Segmentačné metódy

Sada segmentačných algoritmov poskytuje v rámci nástroja FitLayout možnosť analyzovať a identifikovať oblasti z webových stránok a PDF dokumentov. V rámci FitLayout sú momentálne implementované tri segmentačné metódy. Sú to:

- *VIPS - Vision-based page segmentation*, algoritmus vizuálnej segmentácie.
- *BCS - Block Clustering Segmentation*, algoritmus zhlukovania blokov.
- *Basic page segmentation*, algoritmus zoskupovania vizuálnych oblastí, tiež pomenovaný ako *Visual area grouping*

Rámec obsahuje sadu metód a datových štruktúr s ktorými pracujú rôzne kategórie segmentačných algoritmov, takže je možné jednoducho integrovať do systému ďalšie algoritmy. V aplikačnom rozhraní sa implementovaným segmentačným algoritmom predáva na vstup artefakt typu *Stŕanka*. Po prebehnutí segmentácie algoritmus vytvára a predáva ďalej artefakt typu *Strom oblastí*.

4.5 Aplikačné rozhranie REST

V prípade ake chceme rámec FitLayout používať ako webovú službu je pre tento účel dostupné aj aplikačné rozhranie REST. Rozhranie je implementované v mikroslužbe *FitLayoutWeb* a je možné ho nasadiť na ľubovoľný aplikačný server, ktorý umožňuje beh mikroslužieb ako je napríklad *Glassfish* alebo *Payara*.

4.6 Vykresľovacie jadrá

Úlohou vykresľovacieho jadra je previesť vstupnú webovú stránku alebo PDF dokument na internú reprezentáciu (artefakt) s ktorou rámec ďalej pracuje. FitLayout momentálne umožňuje používať dve vykresľovacie jadrá a to *CSS Box* [4] a *Puppeteer* [6].

4.6.1 CSS Box

Vykreslovacie jadro CSSBox predstavuje odľahčenú alternatívu k bežným vykresľovacím jadrám, ktoré používajú klasické webové prehliadače ako je napríklad WebKit alebo Gecko. Narozdiel od týchto vykresľovacích jadier, ktoré sú štandardne implementované v jazyku C++ je toto vykreslovacie jadro je napísané výhradne v jazyku Java. Jeho hlavným účelom je poskytnúť množinu kompletných a ďalej spracovateľných informácií o obsahu a rozložení vykreslenej stránky. Jadro spracúvava stránku vo formáte (X)HTML ktorá obsahuje vizuálne formátovanie pomocou jazyka CSS [4]. Z hľadiska implementačneho jazyka je toto vykresľovacie jadro dobré pri používaní v platformovo nezávislých projektoch. CSS Box dokáže taktiež vykresliť PDF dokumenty, čo u druhého vykresľovacieho jadra Puppeteer nie je podporované. Nevýhodou jadra je absencia podpory jazyka JavaScript, ktorý je v dnešných dynamických webových stránkach štandardom a jadro taktiež zlyháva pri vykresľovaní vizuálne komplexnejších webových stránok.

4.6.2 Puppeteer

Puppeteer [6] je nástroj na automatizáciu prehliadača, vyvinutý Googlem. Je to knižnica pre Node.js, ktorá umožňuje ovládať a manipulovať s prehliadačom Chrome alebo Chromium prostredníctvom programovacieho jazyka JavaScript. Vykresľovanie prebieha priamo v prehliadači a vykresľená stránka sa získava pomocou komunikačného protokolu, ktorý prehliadač poskytuje. V rámci FitLayout predstavuje alternatívu k vykresľovaciemu jadru CSS Box.

Puppeteer poskytuje možnosti automatizácie rôznych interakcií v prehliadači, ako je napríklad otváranie stránok, vyhľadávanie, klikanie na prvky, vypĺňovanie formulárov, generovanie screenshotov a PDF súborov a mnoho ďalších. Taktiež umožňuje sledovať a simulovať užívateľské správanie na webových stránkach, testovanie a skriptovanie rôznych scenárov.

Puppeteer je široko využívaný v oblasti automatizácie testovania, tvorby súborov s údajmi zo stránok (web scraping), tvorby náhľadov stránok (pre-rendering), monitorovania a analýzy výkonu webových aplikácií a ďalších oblastí vývoja a analýzy webových technológií.

Jeho hlavnou výhodou je, že umožňuje jednoduché získavanie vykresľenej stránky cez komunikačný protokol a možnosť vykresľovania stránok s podporou JavaScriptu a s komplexnejším vzhľadom. Nevýhoda je nutnosť inštalovať rozsiahlu knižnicu so závislosťami, ktoré zaťažujú úložisko.

Kapitola 5

Segmentačný algoritmus Block-o-Matic

Segmentačný algoritmus BoM – Block-o-Matic [19] je kategorizovaný ako hybridný algoritmus a to pretože jeho návrh je inšpirovaný metódami automatizovaného spracovania dokumentov a metódami vizuálnej segmentácie obsahu. BoM má niekoľko verzií. V rámci tejto práce je popísaná len jedna verzia a to z dokumentu [19]. Algoritmus berie na svoj vstup strom HTML elementov uložených v rozhraní DOM.

5.1 Segmentačný model

Segmentačný proces webovej stránky je rozdelený do troch častí: analýza štrukúry DOM, porozumenie a rekonštrukcia stránky. Ako vstup sa berie DOM. Ten je predspracovaný a jednotlivé elementy sú označované v závislosti na to či sú nositeľmi nejakého obsahu alebo či zastrešujú ďalšie elementy ktoré nosia obsah. Výsledkom tohto predspracovania je obsahová štruktúra. Takto predspracovaný DOM pokračuje do ďalšej fázy segmentácie – do analýzy. V rácmi analýzy sa DOM spracováva algoritmom *c2g*. Výstupom tohto algoritmu je geometrická štruktúra objektov webovej stránky W_{geom} . Posledná fáza berie na vstup geometrickú štruktúru ktorú mapuje na logickú štruktúru W_{log} stránky pomocou algoritmu *g2l*. Algoritmus *g2l* je parametrizovaný a jej výstup je závislý na parametroch *pDC* a *pAC*, ktorých význam bude vysvetlený nižšie v texte. Rekonštrukcia stránky sa odvíja od logickej štruktúry, ktorá závisí na predchádzajúcich spracovaných štruktúrach. Pseudoalgoritmický zápis algoritmov *c2l* a *g2l* spolu s ich popisom je zahrnutý nižšie v texte.

Na obrázku nižšie sú ukázané vzťahy medzi jednotlivými štruktúrami od HTML elementov až po logickú štruktúru po vykonaní segmentácie.

5.2 Geometrická štruktúra

Geometrická štruktúra popisuje ako sú elementy vsadené do stránky a do akých kategórií sú klasifikované. V tejto štruktúre sa získava geometria vyrenderovaného objektu. Geometriu chápeme ako geometrické vlastnosti (napr. výška, šírka, pozícia) elementu. Tieto vlastnosti je možné získať až po vykreslení stránky a sú dané css boxom, ktorý zaobaľuje daný element. Geometrická štruktúra je strom a jej uzly sú tvorené Geometrickými Objektami. Geometrický objekt je formálne definovaný ako $Go = (e, a, f)$, kde:

- e – je konečná množina elementov ktoré geometrický objekt pokrýva

- a – je konečná množina atribútov ktoré môže geometrický objekt nadobúdať.
- f – je konečná množina operácií/funkcií

V množine atribútov a sa nachádzajú následujúce atribúty: *Geometry a Category*. Atribút *geometry* je definovaný nticou (x, y, w, h). Táto ntica reprezentuje absolútnu pozíciu obdĺžnika vo vyrenderovanom dokumente, ktorý pokrýva všetky elementy e. Hodnota x vyjadruje minimálne odsadenie obdĺžnika od vrchnej strany webovej stránky. Hodnota y zas reprezentuje minimálne odsadenie obdĺžnika od ľavej strany stránky. Hodnota w a h značia šírku a výšku daného obdĺžnika. Atribút *category* obsahuje všeobecné označenie kategórie, ktorú geometrický objekt zastrešuje. Množina funkcií f obsahuje následujúce operácie, ktoré môže geometrický objekt vykonávať. V tejto množine sú tieto operácie: *Diagonal(), rDiagonal()* a *Valid()*. Operácia *Diagonal()* vracia hodnotu dĺžky diagonály atribútu geometrie. Operácia *rDiagonal()* vracia pomer funkcie *Diagonal()* spustenej nad geometrickým objektom a jeho rodičom: *go.Diagonal() / pgo.Diagonal()*, kde *pgo* je rodič *go*. Operácia *Valid()* kontroluje validitu elementu - viditeľnosť elementu vo vykreslenej stránke a reprezentatívnosť plochy a obsahu objektu v rácmci webovej stránky. Geometrickú štruktúru - W_{geom} je teda možné definovať ako konečný strom, kde uzly tvoria geometrické objekty. Celý strom je definovaný následovne - koreň stromu je tvorený geometrickým objektom Go_i, jeho bezprostrednými potomkami sú ďalšie geometrické objekty Go_{ij} a ktoré sa odkazujú na ďalšie podstromy geometrických štruktúr $W_{geom_{ijk}}$. Podstromy $W_{geom_{ijk}}$ obsahujú korene Go_{ijk}, ktoré sú bezprostrednými predchodcami uzlu Go_{ij}.

Logická štruktúra simuluje ľudské vnímanie vizuálneho obsahu stránky. Udržuje v sebe prepojenie medzi blokmi stránky. Jednotlivé uzly v štruktúre sú tvorené logickými objektami.

Algoritmus 5.1: Algoritmus *c2g*

Data: element e
Výsledok: Geometrický objekt Go

```
1  def c2g():
2      if e.childrenSize() = 1 then
3          c2g(child);
4      end
5      else
6          if e.valid() then
7              newGo;
8              Go.elements ← e;
9              Go.geometry ← e.css('offsetLeft, offsetTop, width, height');
10             Go.category ← getCategory(e);
11         end
12         for each e.children as child do
13             Go.children ← c2g(child);
14         end
15     end
16 end
```

5.3 Logický objekt

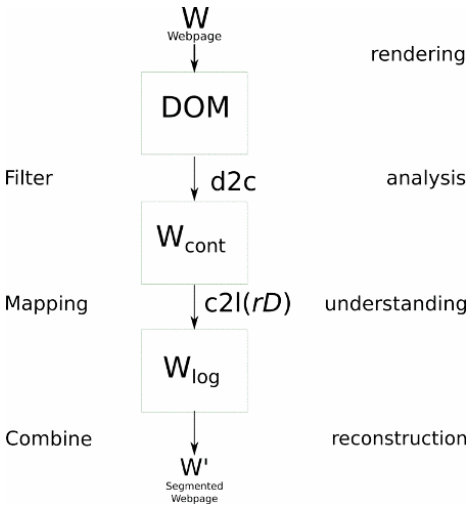
Logický objekt Logický objekt je ntica $Lo = (g, a, f)$ kde:

- g je konečná množina geometrických objektov
- a je konečná množina atribútov ktoré môže logický objekt nadobúdať
- f je konečná množina operácií/funkcií

Množina atribútov obsahuje atribúty *Geometry, Label* a *Follows*. *Geometry* obsahuje množinu bodov popisujúcich polygon - v prostredí vykreslenej webovej stránky najčastejšie obdĺžnik, ktorý reprezentuje minimálny obal geometrických objektov zahrnutých v danom logickom objekte. *Label* reprezentuje popisok sémantickej časti webovej stránky. Označenie môže nadobúdať hodnoty: header (hlavička stránky), navigation (navigácia), title (titulok), content (obsah), table (tabuľka), image (obrázok), logo, footer (päta stránky) a none (nerozoznaná časť). Atribút *Follows* obsahuje predchádzajúci logický objekt. Tento atribút môže nadobúdať hodnoty: independent (nezávislý) alebo obsahuje odkaz na predchádzajúci logický objekt. Množina funkcií obsahuje operácie: *Diagonal(), rDiagonal(), Distance(lo), VisualCuesPresent(), Accept(label)* a *MergeWith(lo)*. Operácia *Diagonal()* vracia hodnotu najdlhšej diagonály geometrie logického objektu. Operácia *rDiagonal()* vracia pomer medzi operáciami *lo.Diagonal()* a *plo.Diagonal()*, kde *plo* je rodičovský uzol objektu *lo*. *Distance(lo)* vracia hodnotu najmenšej vzdialenosti medzi geometriami dvoch logických objektov. Pre príklad *lo.Distance(otherLo)* vracia najmenšiu vzdialenosť medzi logickými objektami *lo* a *otherLo*. Operácia *VisualCuesPresent()* vracia informáciu o tom, či má logický objekt nejaké výrazné vizuálne vlastnosti ako sú farba pozadia a veľkosť fontu. *Accept(label)* priraďuje logickému objektu sémantické označenie a status listového uzla v logickej štruktúre. Status listového uzla znamená, že odstránenie všetkých bezprostredných následníkov logického objektu. Počas tejto operácie sa závislosti medzi štruktúrami W_{geom} a W_{log} nemenia. *MergeWith(otherLo)* vykonáva spojenie geometrií dvoch logických objektov. Napríklad *lo.MergeWith(otherLo)* spája dohromady logické objekty *lo* a *otherLo*. Pri spájaní sa vytvára nová geometria, ktorá pokrýva obe spájané logické objekty. Hľadá sa minimálna ortogonálna obálka.

5.4 Logická štruktúra

Logická štruktúra simuluje ľudské vnímanie vizuálneho obsahu stránky. Udržuje v sebe prepojenie medzi blokmi stránky. Jednotlivé uzly v štruktúre sú tvorené logickými objektami. Logickú štruktúru – W_{geom} je definovaná ako konečný strom kde uzly tvoria geometrické objekty. Celý strom je definovaný následovne – koreň stromu je tvorený geometrickým objektom Lo_i , jeho bezprostrednými potomkami sú ďalšie geometrické objekty Lo_{ij} a ktoré sa odkazujú na ďalšie podstromy geometrických štruktúr $W_{log_{ijk}}$. Podstromy $W_{log_{ijk}}$ obsahujú korene Lo_{ijk} ktoré sú bezprostrednými predchodcami uzlu Lo_{ij}. Algoritmus, ktorý celý proces vytvárania logickej štruktúry formálne popisuje je znázornený na 5.2



Obr. 5.1: Schéma algoritmu Block-o-Matic, ktorá znázorňuje postupnosť vytvárania jednotlivých štruktúr. Zdroj: [19]

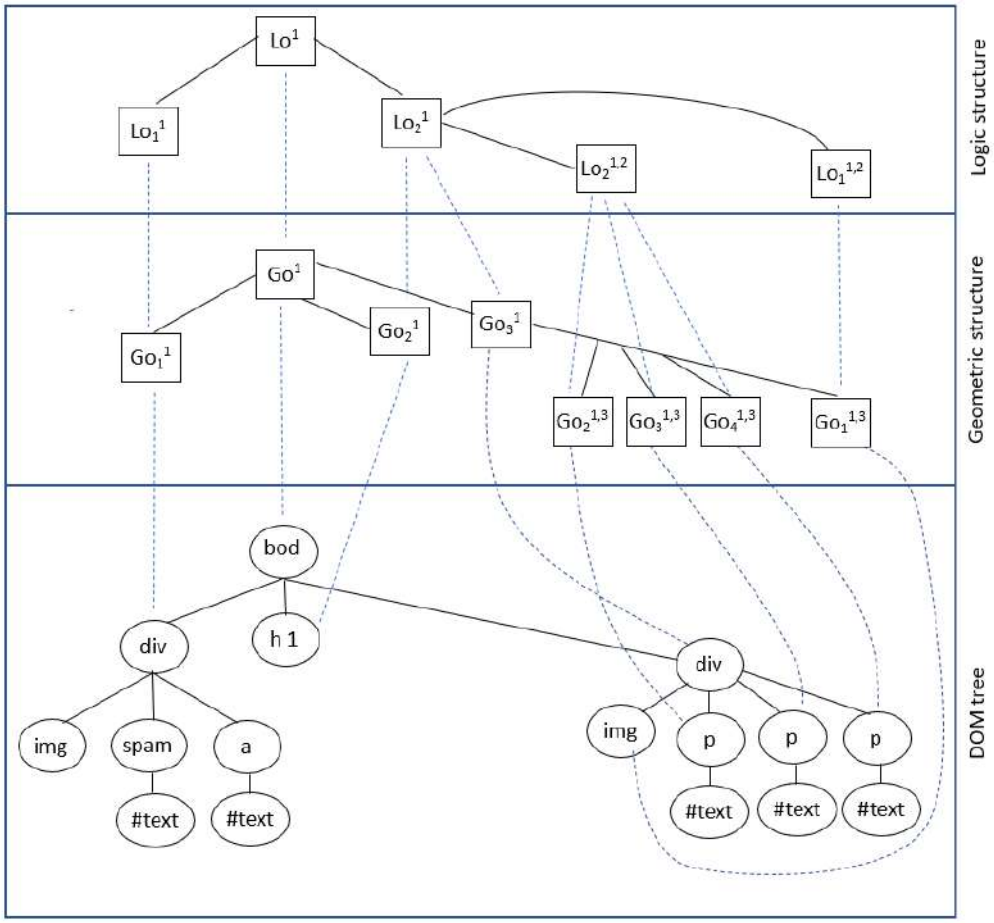
Algoritmus 5.2: Algoritmus *g2l*

Data: Logický objekt *Lo*
Výsledok: Logický objekt *Lo*

```
1  def g2l():
2      if Lo.rDiagonal() ≥ pAC then
3          Lo.label ← getLabelFromContent(Lo);
4          for each Lo.children as child do
5              g2l(child)
6          end
7      end
8      if Lo.VisualCuesPresent() then
9          Lo.Accept();
10         Lo.label ← getLabelFromContent(Lo);
11     end
12     else
13         for each sibling object of Lo as Lo_s do
14             g2l(child) if Lo.Distance(Lo_s) < ϵ and Aligned(Lo, Lo_s) then
15                 Lo.MergeWith(Lo_s);
16                 Lo.label ← getLabelFromContent(Lo);
17             end
18         end
19         if Lo.rDiagonal() < pAC and for any Lo_s.rDiagonal() < pAC then
20             Lo.Accept();
21         end
22     end
23 end
```

5.5 Predspracovanie

Na začiatku segmentácie prebieha predspracovanie dokumentu, kedy sa prechádza celá DOM štruktúra a analyzujú sa jednotivé tagy elementov. Algoritmus sa snaží klasifikovať elementy do štyroch kategorií – content, content container, container a default.Výstupom tejto fázy je štruktúra W_{cont} . Celý proces začína od elementu body. Prakticky sa jedná o DOM štruktúru v ktorej sú elementy označkované vyššie spomínanými kategóriami. Analýza stránky Analýza stránky má za úlohu vytvorit geometrickú štruktúru z predspracovanej DOM štruktúry. DOM štruktúra sa prechádza opäť od elementu body a postupne sa vytvára strom geometrickej štruktúry z geometrických objektov. Nový geometrický objekt je nalpnený informáciami o kategorií a geometrií, ktoré sú získané z jednotlivých elementov.

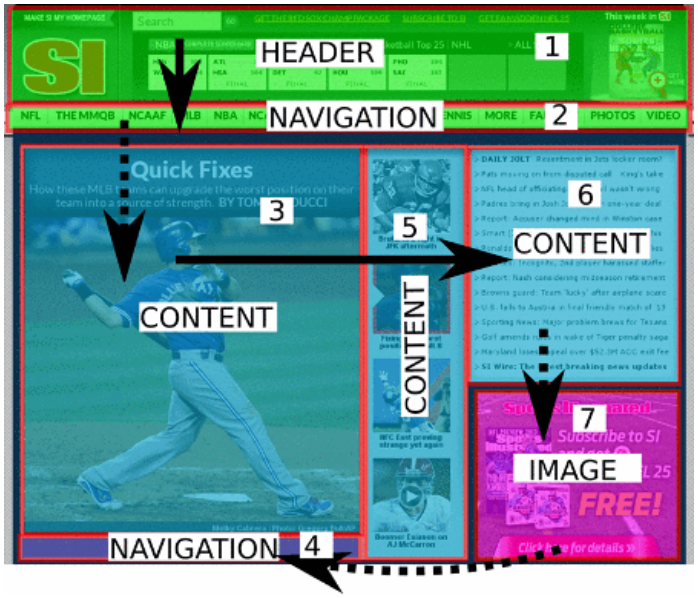


Obr. 5.2: Schéma, ktorá znázorňuje jednotlivé prepojenia medzi štruktúrami vytvorenými počas segmentácie

5.6 Proces porozumenia

Proces porozumenia mapuje geometrickú štruktúru na logickú štruktúru. Počas mapovania sa berie v úvahu pomer diagonál medzi logickými objektmi, kategória objektov, ich poloha v rámci stránky a vzdialenosť medzi objektami. Počas tohto procesu sa algoritmus snaží

zoskupiť objekty, ktorých pomer diagonál nie je väčší ako parameter *pAC* a maximálne oddelenie objektov medzi sebou nepresahuje parameter *pDC*. Parameter *pAC* reprezentuje granularitu. To znamená, že v závislosti na hodnote sa bude meniť veľkosť blokov vo výslednej segmentácií. V prípade malých hodnôt bude stránka rozsegmentovaná na viac malých blokov. Analogicky, v prípade väčších hodnôt bude výsledná segmentácia obsahovať menej velkých blokov - sémantické bloky budú väčšie. Celá táto stratégia sa snaží simulovať ľudské vnímanie obsahu. Toto vnímanie vychádza zo štyroch *Gestaltových zákonov* [**?**]: zákon blízkosti, zákon podobnosti, zákon uzavrenosti a zákon jednoduchosti. Zákon blízkosti pojednáva o tom, že vizuálne objekty, ktoré spolu súvisia, zvyknú byť zoskupené blízko seba. Zákon podobnosti poukazuje na to, že objekty, ktoré sú si vizuálne podobné, sú taktiež zoskupené blízko seba. Zákon uzavrenosti poukazuje na to, že objekty sú zoskupené, ak spolu tvoria nejakú štruktúru v dokumente. Posledný zákon poukazuje na to, že objekty v dokumentoch bývajú zoskupené do jednoduchých štruktúr, ktoré v sebe udržujú symetriu a pravidelnosť. Celý proces porozumenia sa drží týchto zákonov. Napríklad, zhlukujú sa objekty, ktoré sú k sebe čo najbližšie - zákon blízkosti, alebo sú medzi sebou zarovnané - zákon jednoduchosti. Tento proces je ovplyvnený zvolenými parametrami *pAC* a *pDC*. Parameter *pDC* je prahová hodnota ktorá určuje minimálnu odchýlku vertikálneho a horizontálneho zarovnania objektov.



Obr. 5.3: Konečná výsledok segmentácie webovej stránky, ktorý je obsiahnutý v stromovej štruktúre logických objektov. Štruktúra zoskupuje vizuálne oblasti do jednej veĺkej sémantickej oblasti. Zdroj: [19]

Kapitola 6

Implementácia algoritmu do rámca FitLayout

Implementácia algoritmu BoM sa nachádza v balíku *cz.vutbr.fit.layout.bom.imlp*. Celkovo balík obsahuje trinásť tried, z ktorých hlavné časti algoritmu sú implementované v triedach *GeometricObject.java*, *LogicalObject.java* a *Bom.java*. Pomocné metódy pre chod algoritmu sa nachádzajú v triede *Utils.java*.

Celý proces začína v triede `BomProvider` ktorá rozširuje triedu `BaseArtifactService`. Trieda teda očakáva artefakt ako vstup. Predanie artefaktu prichádza v metóde `process()`, ktorá potom volá metódu `createAreaTree`. Tá následne berie ako vstup artefakt stránky a vstupuje sa do samotnej implementácie segmentačného algoritmu. Časť štruktúry metódy `precess()` vyzerá následovne:

```
public Artifact process(Artifact input) throws ServiceException {
    if (input != null && input instanceof Page)
    {
        AreaTree atree = createAreaTree((Page) input);
        ....
```

6.1 Trieda Bom

Bom je vstupná trieda do celého procesu segmentácie. V tejto časti implementácie sa volajú všetky jednotlivé fázy segmentácie. Trieda je volaná v metóde `createAreaTree()` a získavajú sa v nej aj parametre poskytnuté cez rozhranie príkazovej riadky. Postup segmentácie v metóde vyzerá naśledovne:

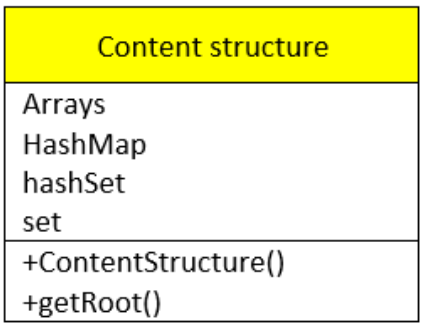
```
Bom bom = new Bom();
bom.setPredefinedAC(pAC);
bom.setPredefinedDC(pDC);
bom.startSegmentation(page);
```

V metóde startSegmentation() sa začína hľadaním sa uzla s tagom BODY. Vyhľadáva sa pomocou funkcie getBoxByTagName(). Táto funkcia simuluje vyhľadávanie elementu v prostredí prehliadača, cez rozhranie DOM, kde sa podobná operácia vykonáva za pomoci metódy getElementByTagName(). V prípade rámca FitLayout ale nepracujeme s rozhraním DOM ale s objektom ktorý v sebe udržuje strom objetov typu Box. Tieto objekty reprezentujú vizuálne vlastnosti jednotilivých elementov po vykreslení. Trieda Bom v sebe

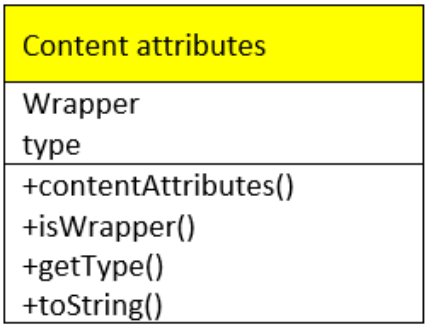
zaobaľuje dve premenné ktoré sú esenciálne pre celkový proces segmentácie a to pDC a pAC. Pri nastavovaní týchto premenných klientským kódom, prebieha kontrola rozsahu hodnôt. Premenná pDC je obmedzená na rozsah 0 až 1000, pre typ integer a jej prednastavená hodnota je 50. Premenná pAC sa musí zas pohybovať v rozmedzí 0 až 1 pre typ float a jej prednastavená hodnota je 0.3. Postupne sa prechádza cez tok segmentácie volaním metód processContentStructure(), processGeometricStructure(), prepareLogicStructure() a processLogicStructure(). Jednotlivé volania a vytvárania štruktúr vyzerajú v metóde `startSegmentation` následovne:

```
Box root = Utils.getBoxByTagName("BODY", this.page);
processContentStructure(root);
this.contentStructure.setRoot(root);
GeometricObject geoObjRoot = processGeometricStructure(this.contentStructure);
this.logObjPage = prepareLogicStructure(geoObjRoot, null);
processLogicStructure(this.logObjPage);
```

Metóda processContentStructure() berie ako svoj vstupný argument nájdený podstrom, ktorý má za koreň Box objekt s tagom BODY. Hlavná funkcia metódy je predspracovanie jednotlivých objektov v Box strome. V tejto časti je potrebné označiť jednotlivé Box objekty príslušnou kategóriou – CONTENT, CONTAINER, CONTENT_CONTAINER alebo DEFAULT. Tieto kategórie sú definované v enumerátore ContentStructure.ContentType. Pri označovaní objektu sa používa objekt ContentStructure ktorý v podstate funguje ako asociatívne pole v ktorom sa mapujú Box objekty na príslušné objekty ContentAttributes. Objekt ContentAttributes reprezentuje dodatočné atribúty pre daný Box objekt. Tento krok je z dôvodu že nechceme modifikovať štruktúru



(a) Trieda ContentStructure ktorá reprezentuje tabulku pre Box uzly a jej atribúty



(b) Trieda ContentAttributes ktorá drží dodatočné atribúty

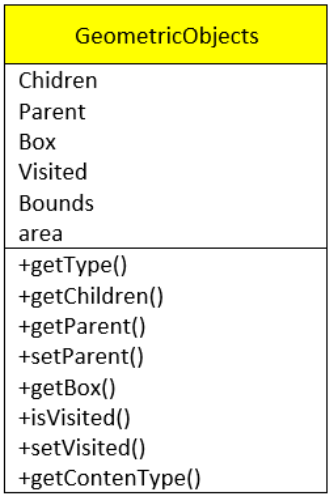
Obr. 6.1: Triedy ContentStructure a ContentAttributes.

Box stromu a ani rozhranie objektu neposkytuje možnosť vkladať nové atribúty. Metóda processContentStructure() je rekurzívna a prakticky v nej prebieha testovanie Box objektu aby sa mu vedela priradiť správna kategória. Metóda isTextContent() testuje či je daný Box objekt textový uzol a či obsahuje text. V prípade že áno, tak sa daný Box objekt uloží do tabuľky v objekte ContentStructure a priradí sa mu kategória CONTENT. Ďalej sa testuje či už daný Box objekt nie je označený v tabuľke. Metóda isExcluded() testuje či je objekt obsahuje tag ktorý má byť vyradený z ďalšieho spracovania – toto sú elementy s ktoré v

stránke nie sú vykreslené. Jedná sa o tagy ako napr. SCRIPT, STYLE, BR. Následne sa testuje viditeľnosť objektu metódou isVisible(). Objekt je viditeľný ak jej obsah je nenulový a zároveň ak box objekt má nastavený visibility atribút a atribút display nie je nastavený na hodnotu none. Metóda isValid() prehľadáva podstrom objektu a zisťuje či sa v ňom nachádzajú nejaké vizuálne alebo textové uzly. V prípade ak sa objekt dostane cez tieto testy tak sa mu priraďuje kategória pomocou funkcie getContentType() a objekt sa uloží do tabuľky.

6.2 Implementácia gemoetrickej štruktúry

Metóda processGeometricStructure() vytvára geometrickú štruktúru z Box stromu a z tabuľky atribútov, kde sú označené a priradené kategórie jednotlivým uzlov v podstrome Box stromu, začínajúcim od koreňa s tagom BODY. Spracovanie prebieha rekurzívne, prechádza sa testovaním na textový obsah, kontroluje minimálna šírka a výška geometrie objektu, ktorá je nastavená na minimálne 10 pixelov vo vyrenderovanej stránke. Po prechode týmto testovaním sa vytvára nový geometrický objekt cez funkciu createGeometricObject() a pokračuje sa rekurzívnym zanáraním do Box stromu. Metóda createGeometricObject() po vytvorení geometrického objektu volá nad týmto objektom metódu addContentBox(), ktorá prepojí Box objekt s geometrickým objektom. Pre objekt sa vypočíta relatívny obsah priraďovaného Box objektu. Relatívny obsah sa počíta ako pomer Box objektu a jeho rodiča. Tento výpočet realizuje funkcia Utils.computeRelativeArea(). Výstupom tejto funkcie je geometrická štruktúra a vrátená hodnota predstavuje koreň tejto štruktúry.

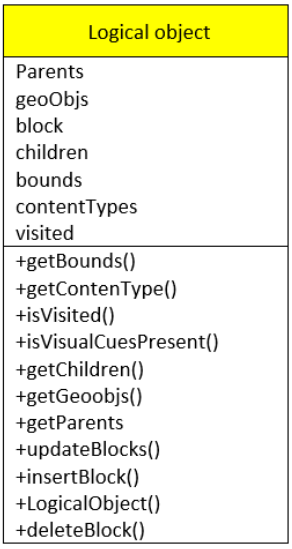


Obr. 6.2: Trieda GeometricObject a jej metódy

6.3 Implementácia logickej štruktúry

Metóda prepareLogicStructure() rekurzívne prechádza celú geometrickú štruktúru, vytvára logické objekty a prepája tieto objekty s geometrickými objektami prehchádzanej štruktúry. Podmienkou pre vytvorenie logického objektu je to aby prepájaný geometrický objekt bol jedným z kategórie CONTAINER, CONTENT_CONTAINER, CONTAINER alebo DEFAULT. V rámci vytvárania nového logického objektu sa volá funkcia createLogicalObject()

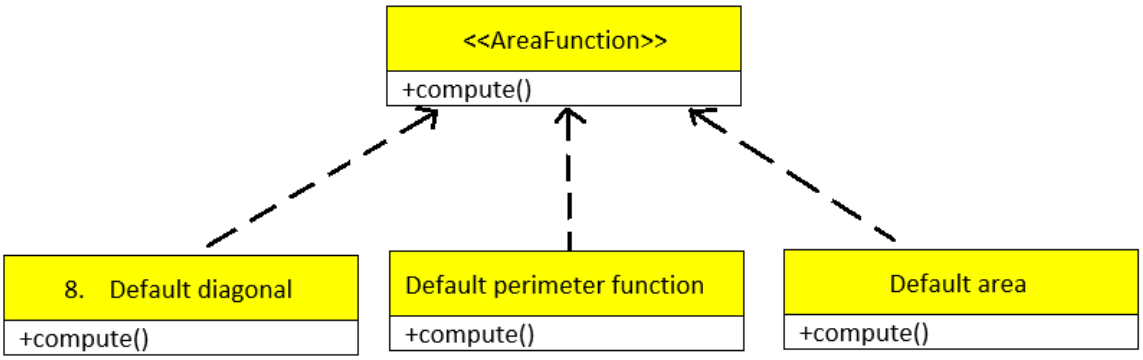
v ktorej prepájanie logického objektu a geometrického objektu prebieha volaním metódy addGeometricObject(). Počas tohto procesu sa taktiež prepája logický objekt s ďalším logickým objektom, čím sa vytvára rodičovská závislosť a postupne sa tak vytvára strom logických objektov – logická štruktúra. Pri tomto prepájaní vzniká aj nová geometria ktorá je reprezentovaná logickým objektom. Realizuje sa to tak, že sa okolo geometrie geometrického objektu a geometrie logického vytvorí nová geometria ktorá predstavuje minimálnu obdĺžnikovú obálku nad oboma spájanými geometriami – bounding box. Následne sa zisťuje prítomnosť vizuálnych vlastností ako je farba pozadia boxu. Ak má box nastavenú nejakú farbu pozadia – premenná uchováva referenciu na nejaký Color objekt, tak sa nastavuje flag pre prítomnosť vizuálnej vlastnosti. Tiež sa kontroluje veľkosť fontu oproti súrodeneckým boxom. Ak je font výraznejší než u súrodencov tak v takom prípade sa rovnako nastavuje vizuálny flag. Metóda vracia koreň ktorý sa odkazuje na začiatok stromu logickej štruktúry. Metóda processLogicStructure() prechádza vytvorenú logickú štruktúru a postupne odstraňuje alebo spája jednotlivé uzly v logickom strome v závislosti na nastavených parametroch pAC a pDC. Odstraňovanie logického objektu zo štruktúry prebieha v prípade ak má objekt práve jedného bezprostredného potomka. Inak prebieha testovanie a spájanie logických objektov v štruktúre.



Obr. 6.3: Trieda LogicalObject a jej metody

O odstránenie uzla sa stará metóda removeLogicObject(a), ktorá prepojí všetkých potomkov uzla a s rodičovským uzlom uzla a. Pri priechode funkciou sa najprv testuje pomer obsahov koreňového objektu logickej štruktúry, ktorý reprezentuje celú stránku a bezprostrednými následníkmi spracovávaného logického objektu. Toto realizuje metóda areaRatioTo(). Táto metóda berie ako jeden zo svojich vstupných argumentov objekty ktoré implementujú rozhranie AreaFunction. Momentálne sú k dispozicií tri triedy ktoré implementujú iné výpočty obsahu, resp. hodoty medzi ktorými sa počíta pomer. Trieda DefaultDiagonal počíta velkosť diagonály geometrie logického objektu. Trieda DefaultPerimeterFunction počíta obvod geometrie logického objektu a trieda DefaultAreaFunction počíta objem. Každá trieda ktorá implementuje rozhranie AreaFunction, musí implementovať metódu compute(). Trieda Bom má predvolene nastavený výpočet cez DefaultDiagonal. Je ale možné experimentovať, meniť a nadstavovať segmentáciu implementovaním nových tried kde výpočet

realizuje funkcia compute. Ďalej sa vo funkcií znovu zisťuje pomer medzi potomkom a rodičom. Ak je pomer menší ako prahová hodnota pAC tak sa vykonáva zisťovanie vzdialenosti dvoch logických blokov volaním metódy getDistanceBetweenLogicalObjects() a zarovnanie oboch logických objektov volaním getAligmentOfLogicalObjects(). Ak je vzdialenost medzi objektmi menšia ako prahová hodnota pDC a zároveň sú zarovnané vertikálne alebo horizontálne, tak sa vykoná spojenie objektov volanim metódy mergeWith(). Po výstupe z funckcie processLogicStructure(), máme k dispozicií naplnenú logickú štruktúru segmentovanej stŕanky. Štruktúra je prechádzaná a nakoniec sa ukladá do artefaktu AreaTree metódou buildAreaTree() ktorá je volaná v triede BomProvider.



Obr. 6.4: Rozhranie AreaFunction

Kapitola 7

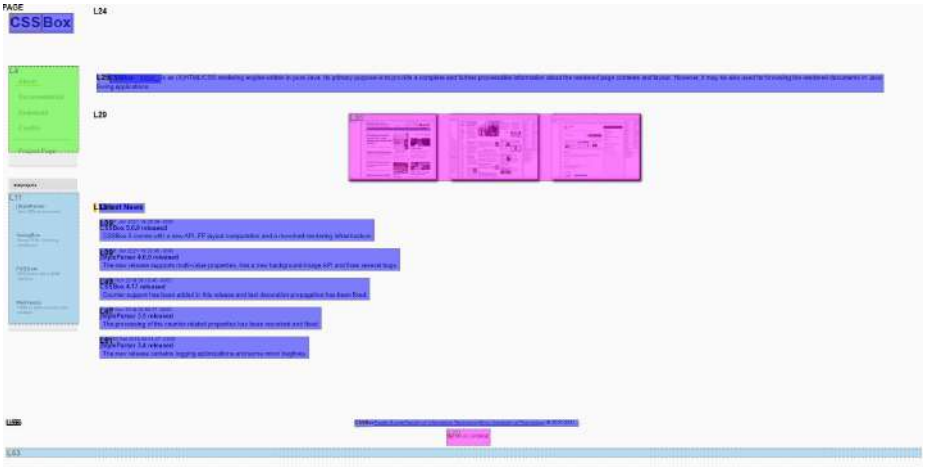
Testovanie

Táto kapitola sa zaoberá experimentami a vyhodnotením segmentácie implementovaného algoritmu. Ako referečná metóda poslúžila javascriptovská implementácia BoM algoritmu, dostupná na https://github.com/openpreserve/pagelyzer/tree/master/SettingsFiles/js. Ďalšiu refernečnú metódu predstavuje algoritmus VIPS, ktorý je integrovaný v systéme FitLayout. Jednotivé experimenty a výsledky segmentácie sú vyobrazené nižšie. Testovanie segmentačných metód prebehlo v prostredí operačného systému Ubuntu 20.04 LTS s použitím rozhrania príkazoveho riadku.

7.1 Vizuálny test 1

```
Prikaz a parametre:
    java -jar FitLayout.jar RENDER -b puppeteer http://cssbox.sf.net \
                           SEGMENT -m bom -O pAC=0.3 pDC=50 \
                           EXPORT -f png
```

(a) Segmentácia získaná pomocou implementovanej segmentačnej metódy Block-o-Matic s vykresľovacím jadrom Puppeteer



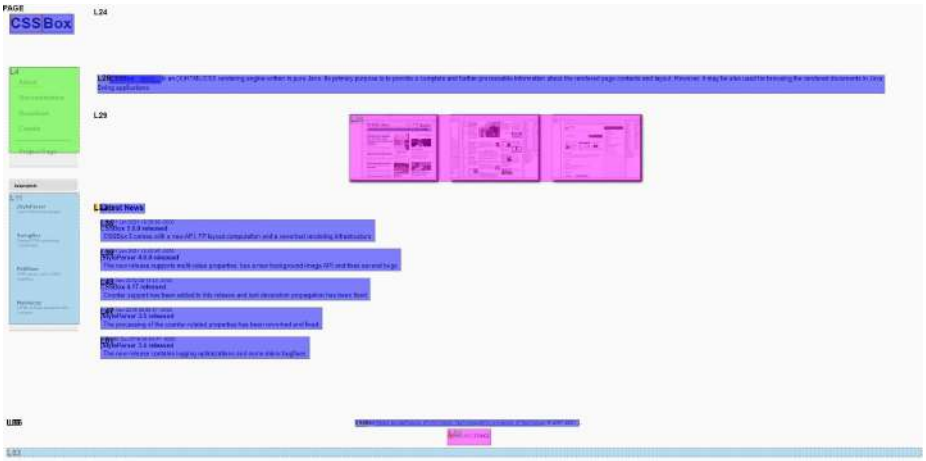
(b) Segmentácia získaná pomocou referenčnej segmentačnej metódy Block-o-Matic v prostredí prehliadača Chrome

Obr. 7.1: Porovnanie referenčnej implementácie s implementáciou v rozhraní FitLayout

7.2 Vizuálny test 2

```
Prikaz a parametre:
    java -jar FitLayout.jar RENDER -b puppeteer http://cssbox.sf.net \
                           SEGMENT -m bom -O pAC=0.3 pDC=50 \
                           EXPORT -f png
```

(a) Segmentácia získaná pomocou implementovanej segmentačnej metódy Block-o-Matic s vykresľovacím jadrom Puppeteer



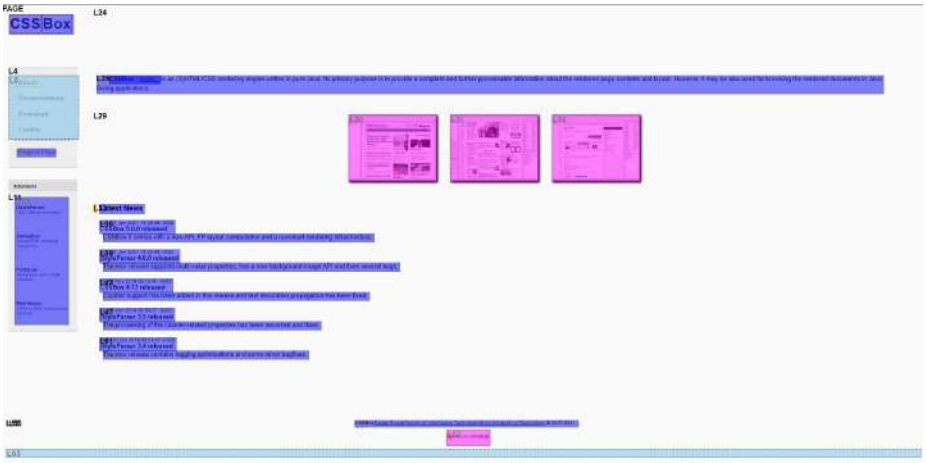
(b) Segmentácia získaná pomocou referenčnej segmentačnej metódy Block-o-Matic v prostredí prehliadača Chrome

Obr. 7.2: Porovnanie referenčnej implementácie s implementáciou v rozhraní FitLayout

7.3 Vizuálny test 3

```
Prikaz a parametre:
    java -jar FitLayout.jar RENDER -b puppeteer http://cssbox.sf.net \
                           SEGMENT -m bom -O pAC=0.1 pDC=50 \
                           EXPORT -f png
```

(a) Segmentácia získaná pomocou implementovanej segmentačnej metódy Block-o-Matic s vykresľovacím jadrom Puppeteer



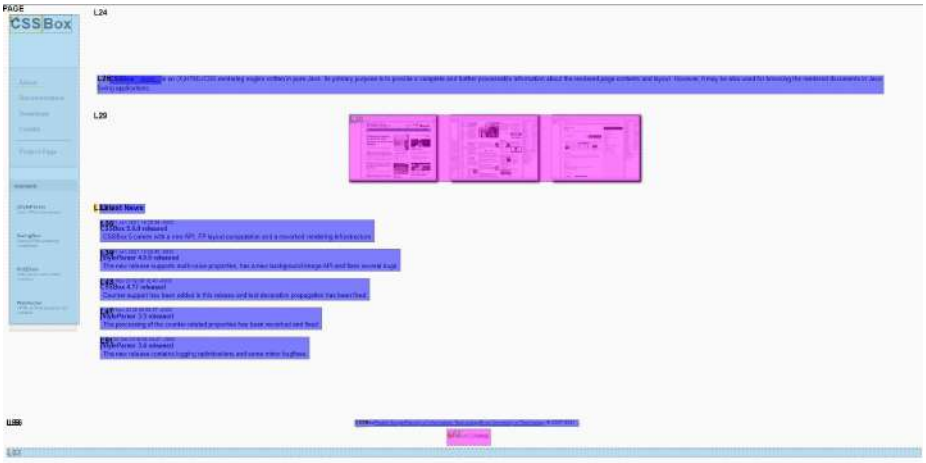
(b) Segmentácia získaná pomocou referenčnej segmentačnej metódy Block-o-Matic v prostredí prehliadača Chrome

Obr. 7.3: Porovnanie referenčnej implementácie s implementáciou v rozhraní FitLayout

7.4 Vizuálny test 4

```
Prikaz a parametre:
        java -jar FitLayout.jar RENDER -b puppeteer http://cssbox.sf.net \
                                SEGMENT -m bom -O pAC=0.3 pDC=100 \
                                EXPORT -f png
```

(a) Segmentácia získaná pomocou implementovanej segmentačnej metódy Block-o-Matic s vykresľovacím jadrom Puppeteer



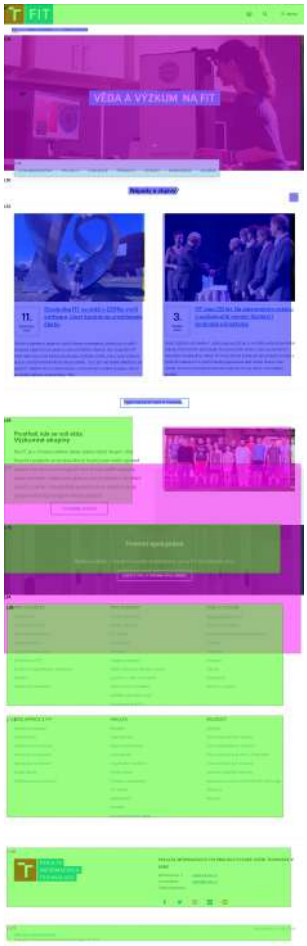
(b) Segmentácia získaná pomocou referenčnej segmentačnej metódy Block-o-Matic v prostredí prehliadača Chrome

Obr. 7.4: Porovnanie referenčnej implementácie s implementáciou v rozhraní FitLayout

7.5 Vizuálny test 5

```
Prikaz a parametre:
    java -jar FitLayout.jar RENDER -b puppeteer https://www.fit.vut.cz/.cs \
                                SEGMENT -m bom -O pAC=0.3 pDC=50 \
                                EXPORT -f png
```

(a) Segmentácia získaná pomocou implementovanej segmentačnej metódy Block-o-Matic s vykresľovacím jadrom Puppeteer



(b) Segmentácia získaná pomocou referenčnej segmentačnej metódy Block-o-Matic v prostredí prehliadača Chrome

Obr. 7.5: Porovnanie referenčnej implementácie s implementáciou v rozhraní FitLayout

7.6 Vizuálny test 6

```
Prikaz a parametre:
    java -jar FitLayout.jar RENDER -b puppeteer https://www.fit.vut.cz/.cs \
                                SEGMENT -m bom -O pAC=0.7 pDC=50 \
                                EXPORT -f png
```

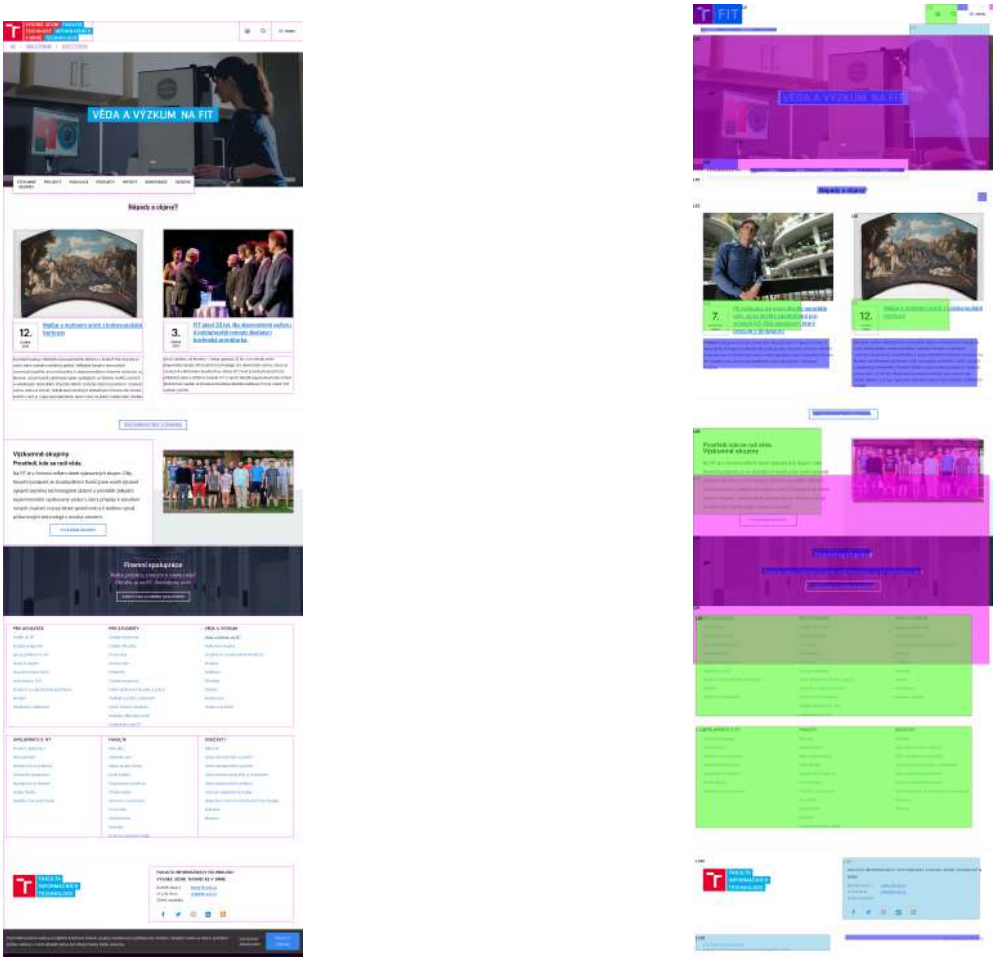
(a) Segmentácia získaná pomocou implementovanej segmentačnej metódy Block-o-Matic s vykresľovacím jadrom Puppeteer

(b) Segmentácia získaná pomocou referenčnej segmentačnej metódy Block-o-Matic v prostredí prehliadača Chrome

Obr. 7.6: Porovnanie referenčnej implementácie s implementáciou v rozhraní FitLayout

7.7 Vizuálny test 7

```
Prikaz a parametre:
    java -jar FitLayout.jar RENDER -b puppeteer https://www.fit.vut.cz/.cs \
                                   SEGMENT -m bom -O pAC=0.2 pDC=50 \
                                   EXPORT -f png
```



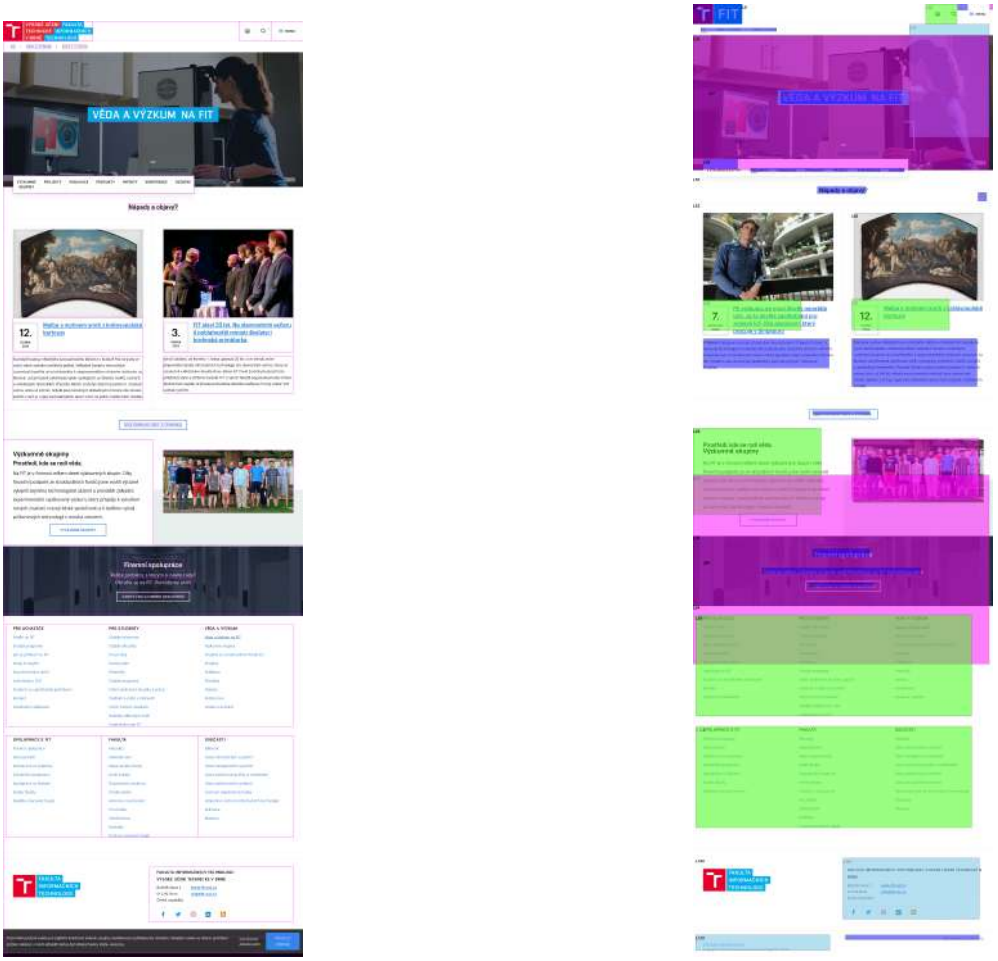
(a) Segmentácia získaná pomocou implementovanej segmentačnej metódy Block-o-Matic s vykresľovacím jadrom Puppeteer

(b) Segmentácia získaná pomocou referenčnej segmentačnej metódy Block-o-Matic v prostredí prehliadača Chrome

Obr. 7.7: Porovnanie referenčnej implementácie s implementáciou v rozhraní FitLayout

7.8 Vizuálny test 8

```
Prikaz a parametre:
    java -jar FitLayout.jar RENDER -b puppeteer https://www.fit.vut.cz/.cs \
                                   SEGMENT -m bom -O pAC=0.2 pDC=50 \
                                   EXPORT -f png
```



(a) Segmentácia získaná pomocou implementovanej segmentačnej metódy Block-o-Matic s vykresľovacím jadrom Puppeteer

(b) Segmentácia získaná pomocou referenčnej segmentačnej metódy Block-o-Matic v prostredí prehliadača Chrome

Obr. 7.8: Porovnanie referenčnej implementácie s implementáciou v rozhraní FitLayout

7.9 Test segmentácie metód VIPS a BoM

Prikaz a parametre:

```
java -jar FitLayout.jar RENDER -b puppeteer http://cssbox.sf.net \
                        SEGMENT -m bom -O pAC=0.3 pDC=50 \
                        EXPORT -f png

java -jar FitLayout.jar RENDER -b puppeteer http://cssbox.sf.net \
                        SEGMENT -m vips -O pDoC6 \
                        EXPORT -f png
```

(a) Segmentácia získaná pomocou implementovanej segmentačnej metódy Block-o-Matic s vykresľovacím jadrom Puppeteer

(b) Segmentácia získaná pomocou segmentačnej metódy VIPS s vykresľovacím jadrom Puppeteer

Obr. 7.9: Porovnanie dvoch segmentácií odlišných metód

Kapitola 8

Záver

V rámci práce boli preštudované rôzne segmentačné algoritmy z ktorých bol ako najvhodnejší kandidát pre implementáciu zvolený algoritmus BoM (Block-o-Matic). Tento algoritmus sa bol implementovaný do aplikačného rozhrania rámca FitLayout. Implementovaný algoritmus bol v rámci experimentov porovnaný s jeho JavaScript variantou. Pri experimentoch bolo zistené že referečná implementácia a implementácia v rámci FitLayout segmentujú do istej miery rovnako a dosahujú približne rovnakých výsledkov pri segmentácií. Isté odchýlky vo výsledkoch sú pravdepodobne spôsobené tým aké vykresľovacie jadro obe segmentačné metódy používajú. Taktiež sú odchýľky spôsobené v štruktúre ktorá vstupuje do procesu segmentácie, V JavaScriptovskej variante ide na vstup DOM a prechádzajú sa jednotlivé elementy v stromovej štruktúre v rámci FitLayout sa prechádza interná štruktúra artefaktu Stŕanka, ktorý reprezentuje vyrenderovanú stránku bez redundantných elementov a elementov ktoré nie sú vykreslené.

Literatúra

[1] *Wayback Machine* [Internet Archive].

[2] Ben Saad, M. a Gançarski, S. Using Visual Pages Analysis for optimizing Web Archiving. In:. Marec 2010. DOI: 10.1145/1754239.1754287.

[3] Burget, R. Automatic Document Structure Detection for Data Integration. In: *Proceedings of the 10th International Conference on Business Information Systems*. Berlin, Heidelberg: Springer-Verlag, 2007, s. 391–397. BIS'07.

[4] Burget, R. *CSSBox* [https://cssbox.sourceforge.net/]. 2023.

[5] Cai, D., Yu, S., Wen, J.-R. a Ma, W.-Y. *Vips: A Vision Based Page Segmentation Algorithm*. MSR-TR-2003-79. Microsoft Research, 2003.

[6] Developers, G. C. *Puppeteer* [https://github.com/puppeteer/puppeteer]. Year. Accessed: 2023.

[7] Hattori, G., Hoashi, K., Matsumoto, K. a Sugaya, F. Robust Web Page Segmentation for Mobile Terminal Using Content-Distances and Page Layout Information. In: ACM. *Proceedings of the 16th International Conference on World Wide Web*. Banff, Alberta, Canada: [b.n.], 2007, s. 361–370.

[8] Kohlschütter, C. a Nejdl, W. A Densitometric Approach to Web Page Segmentation. In: ACM. *Proceedings of the 17th ACM Conference on Information and Knowledge Management*. New York, NY, USA: [b.n.], 2008, s. 1173–1182.

[9] Li, X., Phang, T.-H., Hu, M. a Liu, B. Using Micro Information Units for Internet Search. In: ACM. *Proceedings of the Eleventh International Conference on Information and Knowledge Management*. New York, NY, USA: [b.n.], 2002, s. 566–573.

[10] Liu, W., Deng, X., Huang, G. a Fu, A. An antiphishing strategy based on visual similarity assessment. *IEEE Internet Computing*. 2006, zv. 10, č. 2, s. 58–65. DOI: 10.1109/MIC.2006.23.

[11] Liu, X., Lin, H. a Tian, Y. Segmenting Webpage with Gomory-Hu Tree Based Clustering. *Journal of Software*. december 2011, zv. 6, č. 12, s. 2421–2425.

[12] Liu, Y., Liu, W. a Jiang, C. User Interest Detection on Web Pages for Building Personalized Information Agent. In: Li, Q., Wang, G. a Feng, L., ed. *Advances in Web-Age Information Management*. Springer Berlin / Heidelberg, 2004, sv. 3129, s. 280–290. Lecture Notes in Computer Science.

[13] Mahmud, J. U., Borodin, Y. a Ramakrishnan, I. V. CSurf: A Context-Driven Non-Visual Web Browser. In: *Proceedings of the 16th International Conference on World Wide Web*. New York, NY, USA: ACM, 2007, s. 31–40. WWW '07.

[14] Manning, C. D., Raghavan, P. a Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[15] Milička, M. a Burget, R. Information Extraction from Web Sources based on Multi-aspect Content Analysis. In: *Semantic Web Evaluation Challenges, SemWebEval 2015 at ESWC 2015*. Portorož: Springer International Publishing, 2015, sv. 2015, s. 81–92. Communications in Computer and Information Science. ISBN 978-3-319-25517-0.

[16] Mukherjee, S., Yang, G. a Ramakrishnan, I. V. Automatic Annotation of Content-Rich HTML Documents: Structural and Semantic Analysis. In: *Intl. Semantic Web Conf. (ISWC*. 2003, s. 533–549.

[17] Pnueli, A., Bergman, R., Schein, S. a Barkol, O. *Web Page Layout via Visual Segmentation*. HP Laboratories, 2009.

[18] Sanoja, A. Web page segmentation, evaluation and applications. Január 2015.

[19] 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.

[20] Sun, F., Song, D. a Liao, L. DOM Based Content Extraction via Text Density. In: *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Beijing, China: ACM, 2011, s. 245–254. SIGIR '11.

[21] Wu, O., Chen, Y., Li, B. a Hu, W. Evaluating the Visual Quality of Web Pages Using a Computational Aesthetic Approach. In: *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*. New York, NY, USA: ACM, 2011, s. 337–346. WSDM '11.

[22] Xiao, Y., Tao, Y. a Li, Q. Web Page Adaptation for Mobile Device. In: *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*. 2008, s. 1–5. DOI: 10.1109/WiCom.2008.1182.

Príloha A

Príloha

Priložené CD obsahuje:

- Projekt FitLayout s implementovaným algoritmom Block-o-Matic v ceste:
- `fitlayout-bom/fitlayout-segm-bom/src/main/java/cz/vutbr/fit/layout/bom`
- Zdrojový kód technického textu práce zabalený v priečinku `DP-FIT.zip`