VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SEGMENTACE WEBOVÝCH STRÁNEK S VYUŽITÍM SHLUKOVÁNÍ

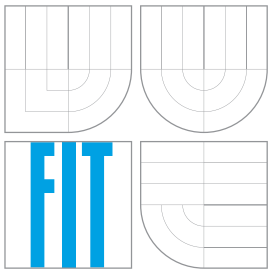
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ LENGÁL

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SEGMENTACE WEBOVÝCH STRÁNEK S VYUŽITÍM SHLUKOVÁNÍ

WEB PAGE SEGMENTATION ALGORITHMS BASED ON CLUSTERING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ LENGÁL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2017

Abstrakt

Tato zpráva se zabývá segmentací webových stránek, jež je důležitou součástí oboru extrakce informací. V první části popisujeme několik obecných způsobů jak se dá implementovat. Následně je pak představena metoda Box Clustering Segmentation, ktrerá přichází s mírně odlišným přístupem k segmentaci. V druhé polovině práce je pak popsána implementace této metody v rámci nástroje FITLayout a závěrečné testování.

Abstract

This report deals with segmentation of web pages, which is important discipline of information extraction. In the first part, we describe several general ways to implement it. After that we introduce method Box Clustering Segmentation, which comes with a slightly different approach towards segmentation. In the second half, we describe implementation of this method as a part of framework FITLayout and final testing.

Klíčová slova

Segmentace webových stránek, extrakce informací, algoritmus Box Clustering Segmentation, framework FITLayout

Keywords

Web page segmemntation, information extraction, Box Clustering Segmentation algorithm, FITLayout framework

Citace

Tomáš Lengál: Segmentace webových stránek s využitím shlukování, diplomová práce, Brno, FIT VUT v Brně, 2017

Segmentace webových stránek s využitím shlukování

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D.

.....
Tomáš Lengál
22. května 2017

Poděkování

Rád bych poděkoval vedoucímu práce, Ing. Radku Burgetovi, Ph.D., za odborné konzultace a rychlé odezvy na dotazy. Dále bych rád poděkoval Mgr. Pavle Lengálové, která se starala o kontrolu pravopisu.

Obsah

Kapitola 1

Úvod

Internet se pro naši generaci stal primárním zdrojem informací. Při zjišťování nových údajů již málokoho z nás napadne zajít do knihovny nebo do fyzických informačních center, ale spíše se připojíme online a použijeme svůj oblíbený vyhledávač. Na netu jsou základním nosičem informace webové stránky. V souvislosti s nárůstem jejich počtu se zvyšují nároky na nástroje, provádějící jejich automatické zpracování.

Segmentace webových stránek patří do oboru zabývajícím se získáváním znalostí z webových dokumentů. Jejím cílem je pomoci identifikovat ty části dokumentu, které obsahují relevantní informace. Samotná činnost segmentace se skládá z vytvoření struktury reprezentující vstupní dokument a výběru těch částí stránky, které mají potenciál obsahovat relevantní informace. Více je její průběh popsán v kapitole 2. Segmentace stránek je důležitá pro vyhledávací nástroje, ale i pro transformaci obsahu webových dokumentů pro další speciální zařízení.

V souvislosti s webovými stránkami je potřeba počítat s tím, že se způsob jejich tvorby neustále mění a přibývá technik pro jejich vytváření. Z tohoto důvodu jsou některé dříve používané segmentační metody dnes již zastaralé a na současných stránkách nepoužitelné. Při vývoji nových segementačních algoritmů je tedy třeba počítat s tím, že webové stránky se budou v budoucnosti psát jiným způsobem a tomu je třeba vývoj těchto algoritmů přizpůsobit a udělat je co nejvíce nezávislé na způsobu psaní stránek.

Tato práce se zabývá metodou Cluster Based Segementation, která byla vytvořen Ing. Zeleným. Tato metoda přichází s novým přístupem k rozdělení stránky, jež je blížší způsobu jakým se na stránku dívá běžný člověk (více viz kapitola 3).

Cílem naší práce je implementace této metody v rámci frameworku FITLayout. Do tohoto nástroje bylo již několik segmentačních metod implementováno v rámci diplomových prací [12] a [10]. Naším cílem je na tyto práce navázat a přidat tak další možnost segmentace stránek pro uživatele.

Tato zpráva se skládá z pěti hlavních kapitol. První z nich (viz 2) se zabývá segmentací obecně. Je zde vysvětlen její základní princip a popsány některé segmentační přístupy a jejich vlastnosti. Důraz je kladen na metodu VIPS, ktrerá je v dalším textu použita pro porovnávání.

V rámci kapitoly 3 je představen algoritmus Box Clustering Segmentation. Je zde vysvětlen jeho základní princip, v čem se liší od ostatních algoritmů a v podkapitolách (3.3, 3.4, 3.6) jsou podrobně vysvětleny jeho jednotlivé fáze.

Kapitole 4 je věnována frameworku FITLayout. Obsah kapitoly se skládá z obecného popisu FITLayoutu, základního vysvětlení jeho architektury a popsání modulů z nichž se tento nástroj skládá.

Implementace algoritmu Box Clustering Segmention je popsána v kapitole 5. Ta se především věnuje způsobu, jakým byly realizovány vzorce a definice z kapitoly 3 a kromě toho rozebírá podrobněji některé problémy, které byly v [17] pouze naznačeny.

V kapitole 6 je popsán způsob testování výsledného produktu a zhodnocení dosažených výsledků jakožto i porovnání s dalšími segmentačními metodami.

Kapitola 2

Segmentace webových stránek

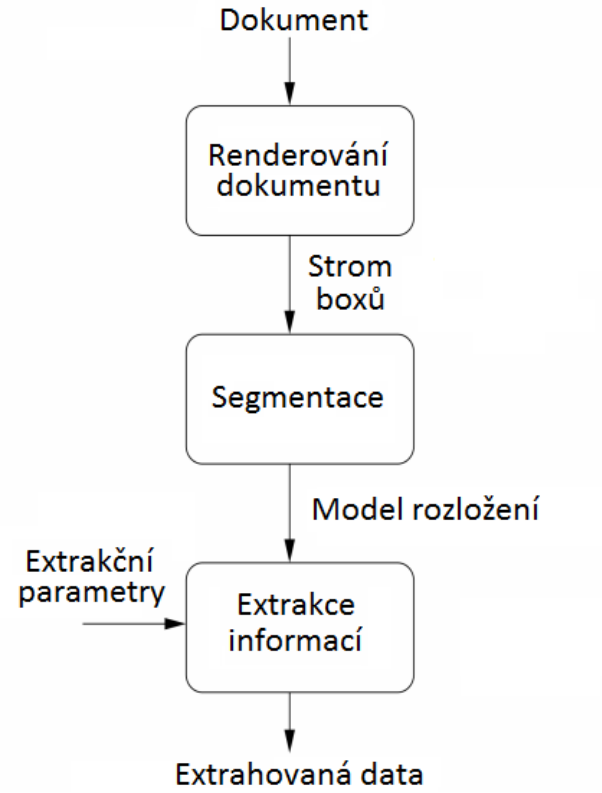
V této kapitole je obecně popsán princip segmentace webových stránek, její význam a dva základní způsoby jak k ní lze přistupovat.

Když mluvíme o segementaci webových stránek, tak je tím myšlen proces, který dostává na vstup data a metadata webového dokumentu a jehož výsledkem je vizuální struktura dokumentu, skládající se ze skupiny významově oddělených bloků a vztahy mezi nimi (viz obrázek 2.1). Tato struktura je obvykle ve formátu stromu [16] a dále se dá využít pro extrakci informací z dokumentu.

Kromě hlavního obsahu lze na obvyklé webové stránce typicky nalézt další dodatečné elementy, které neobsahují relevantní informace. Jedná se například o navigaci nebo reklamy. Jedním z hlavních účelů segmentace stránek je právě tyto bloky identifikovat a oddělit je od částí stránky, které nesou informace důležité z hlediska získávání znalostí.

Základní přístup k segmentaci spočívá v extrakci informací za pomocí HTML tagů, kdy předpokládáme, že specifické tagy mají pevně daný význam. Při tomto přístupu jsou vyhledávány a zpracovávány především HTML tagy <TABLE> (tabulky), <P> (odstavce), <UL> (seznamy) a <H1> až <H6> (různé úrovně nadpisů). Tento přístup je poměrně jednoduchý, nicméně sám o sobě nedostačující. Problém přichází již s faktem, že jazyk HTML je značně flexibilní a velké množství existujících stránek nesplňuje oficiální standard pro tvorbu webových dokumentů. Hlavní nedostatek tohoto přístupu však spočívá v tom, že v praxi je vztah mezi kódem a vzhledem stránky těžké přesně určit a záleží na mnoha vlastnostech, které jsou vypočítány až během vykreslování stránky. Proto se v dnešní době nejčastěji používá přístup, kdy během segmentace proběhne samotné renderování stránky. Metody které využívají tento přístup se označují jako *Visual-based* a jsou podrobněji popsány v kapitole 2.2. Součástí této skupiny je i algoritmus Cluster Based Segmentation (popsaný v kapitole 3), jehož implementace je předmětem této práce.

Segmentace stránek je důležitou součástí procesu získávání znalostí. V [2] je popsán obecný přístup k extrahování obsahu z HTML dokumentů, který je použit i v rámci této práce. Tento přístup byl primárně navržen pro zpracovávání HTML dokumentů, nicméně při provedení mírných úprav ho lze použít i na jiné typy, například na PDF nebo MS Words dokumenty [2]. Jeho základní princip je možné vidět na obrázku 2.1, z něho je možné vyčíst, že se proces skládá ze tří částí. Nejprve je vstupní dokument převeden do stromu boxů (*Box tree*), které reprezentují vizuálně oddělené oblasti. V druhé fázi je provedena segmentace, jejímž výstupem je vizuální struktura celého dokumentu. V posledním kroku je provedena extrakce informací, během níž jsou identifikovány a zpracovávány ty části, které obsahují relevantní informace. Samotné extrahování informace není předmětem této práce a v následujícím textu se mu nebudeme příliš věnovat.



Obrázek 2.1: Proces extrakce informace [2]

Při hodnocení segmentačních algoritmů se používají dvě základní charakteristiky. Jedná se o přesnost a rychlost. Rychlost algoritmu je důležitá neboť se předpokládá, že segmentace bude prováděna nad velkým počtem stránek a v některých případech je nežádoucí, aby výpočet zabral příliš dlouhou dobu. Co se týče přesnosti segmentační metody, tak ji definujeme jako konzistenci mezi výsledkem metody a lidským vnímáním stránky [17].

Testování nové metody se obvykle provádí za pomoci uživatelů nad souborem stránek. Součástí této práce není vyhodnocení algoritmu Cluster Based Segementation, ale pouze jeho implementace, proto se vyhodnocování segmentačních algoritmů z teoretické stránky nemá cenu příliš věnovat. Popisy konkrétních případů jak byly metody testovány lze nalézt v [8] nebo v [17].

V předchozím textu bylo zmíněno, že segmentační metody lze rozdělit do dvou základních skupin - *Visual-based* a *HTML-based*. Jedná se o rozdělení převzato z [14]. V následujících kapitolách jsou obě skupiny podrobněji charakterizovány a jsou uvedeny někteří z jejich zástupců.

2.1 HTML-based metody

Metody v této skupině analyzují obsah webové stránky pouze na základě HTML kódu nebo odpovídajícího DOM stromu, aniž by danou stránku potřebovaly vykreslovat. Jejich výhodou je, že jsou obvykle podstatně rychlejší než *Visual-based* metody [16]. Nevýhodou pak to, že jejich výsledky často nedosahují dostatešné kvality. To je způsobeno tím, že HTML

standard je značně flexibilní a neustále přibývají nové techniky pro vytváření webových dokumentů (viz výše).

2.1.1 Text-based

Algoritmy, které jako vstup používají pouze zdrojový soubor stránky. Příkladem je metoda představená v [11], která předpokládá využití tabulky pro vytvoření rozložení stránky. Ovšem tento přístup se dnes pro celou stránku již příliš nepoužívá a metoda je tedy v praxi nepoužitelná.

2.1.2 DOM-based

Skupina založená na zpracovávání DOM stromu. Jedním ze zástupců je algoritmus WISH, jež je blíže popsán v [9]. Princip WISHe spočívá v hledání takových struktur, které se opakují na stejné úrovni DOM stromu a zároveň mají jiný obsah. Algoritmus tedy předpokládá existenci seznamu položek (tímto způsobem jsou často strukturovány například internetové obchody).

2.2 Visual Page metody

Z pohledu uživatele webová stránka není vnímána jako jediný sémantický objekt, ale jako skupina objektů [8]. V rámci výzkumu [1] byla odhalena skutečnost, že uživatelé při prohlížení stránky očekávají, že určité elementy (hlavní obsah, navigace, reklamy, etc.) se budou nacházet na specifické pozici v rámci stránky. Je tedy zřejmé, že i samotný uživatel využívá prostorové a vizuální vlastnosti za účelem extrakce informací z webové stránky.

Metody patřící do této skupiny se snaží stránku strukturovat stejným způsobem jakým se na ni dívá běžný uživatel a některé z nich dokonce spoléhají na asistenci uživatelů během prosesu učení [16]. Důležitá součást jejich výpočtu je tedy vykreslení stránky, což však zvyšuje časovou náročnost zpracovávání.

Základní koncept těchto algoritmů spočívá v identifikaci elementů stránky a vypočítání jejich určitých vizuálních vlastností (viz dále).

Při zpracovávání webových dokumentů rozlišujeme dva základní typy elementů. Jedná se o textový prvek, u kterého lze určit následující vlastnosti: obsah, třída, velikost, tloušťka, styl, varianta, dekorace a barva. Druhým je obrazový prvek, jež obvykle definujeme pomocí n-tice (šířka, výška, data). V obou případech předpokládáme, že mají prvky tvar obdélníků. Další elementy pak mohou sloužik k obalování skupin prvků. Vstupní dokument se tedy skládá z množiny obdélníků, které do sebe mohou být vnořeny, čímž tvoří stromovou strukturu obdobnou DOM stromu. U těchto obdélníků (prvků) nás mohou zajímat následující atributy[7]:

- Velikost a pozice
- Barva pozadí
- Vlastnosti fontu
- Vlastnosti rámečku
- Obsah testových elementů

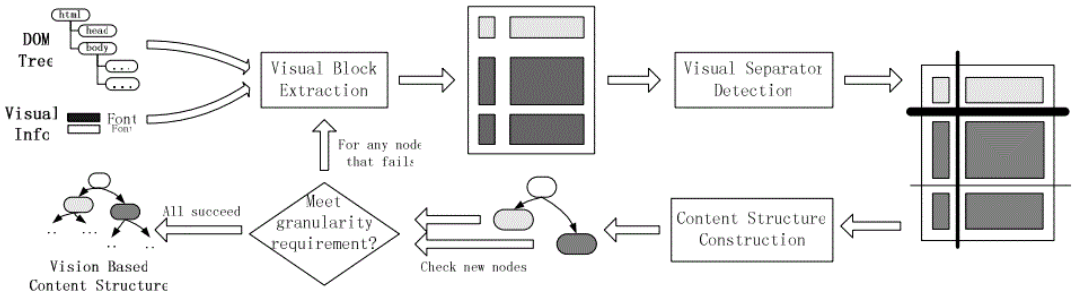
Při práci nad těmito atributy se snažíme vyhýbat absolutním hodnotám a místo nich většinou používáme hodnoty relativní (viz dále v kapitole 3). Podrobnější informace ke klasifikaci prvků dokumentu lze nalézt v [7] nebo [18].

2.2.1 VIPS: Vision-based Page Segementation Algorithm

Metodě VIPS se budeme v rámci této kapitoly věnovat více dopodrobna, neboť ji v kapitole 3 budeme porovnávat s algoritmem Cluster Based Segementation, který je v rámci naší práce implementován.

Algoritmus VIPS byl představen v [8]. Jedná se o jeden z nejznámějších algoritmů používaných pro segmentaci stránek patřící do rodiny *Visual-based*. Metoda vytváří hierarchickou strukturu (strom), která odpovídá vizuální struktuře stránky. VIPS pracuje shora dolů, kdy začíná od celé stránky a tu postupně dělí na menší části.

Vstupem je DOM webové stránky a soubor vizuálních vlastností. Samotná metoda probíhá v iteracích pracujících nad podstromy identifikovaných v předchozích průchodech. Každá iterace se skládá ze tří základních částí, které lze vidět na obrázku 2.2. Jedná se o extrakci vizuálních bloků, detekci separátorů a konstrukce vizuální struktury.



Obrázek 2.2: Architektura algoritmu VIPS [8]

1. Detekce a extrakce vizuálních bloků

   V této části se hledají v podstromu samostatné vizuální bloky, jež odpovídají uzlům v DOM stromu. V první iteraci se hledají bloky nad celou webovou stránkou a v následujích pak nad jednotlivými oblastmy (podstromy). Během hledání je použita skupina pravidel, kterou lze nalézt v [8].

2. Detekce vizuálních separátorů

   Po nalezení bloků je třeba identifikovat jejich vzájemné vztahy. Tohoto je docíleno pomocí separátorů. Seperátor může být vertikální nebo horizontální a má atribut váha, který v podstatě ukazuje jak moc jsou bloky, které separátor rozděluje, vizuálně odlišné.

3. Konstrukce vizuální struktury

   Během třetí fáze algoritmu VIPS je za pomocí separátorů vytvořen strom reprezentující vizuální strukturu.

Po vytvoření vizuální struktury je nad ní provedena kontrola, během níž se určuje zda je výsledek dostatečně strukturovaný (zrnitý). Tato kontrola se provádí pomocí parametru PDoC (*Permitted Degree of Coherence*), který se nastavuje pro celou stránku a který má zásadní vliv na výsledek celého výpočtu. Pokud kontrola selže, tak je nutné provést další iteraci, během níž se zopakují tři výše popsané fáze. Nové vizuální bloky se budou hledat v rámci bloků nalezených v předešlých krocích, u nichž bylo určeno, že nebyly dostatečně rozděleny.

Algoritmus VIPS je již poměrně starý a od publikace [8] se způsob psaní webových stránek v jistých ohledech značně změnil. Vzhledem k tomu Burget a Rudolfová v [7] navrhují vylepšení algoritmu VIPS, které odstraňuje některé jeho nedostatky a přizpůsobuje ho tak, aby lépe fungoval nad modernějším stylem psaní webových stránek. Tato varianta VIPSu byla implementována do nástroje FITLayout v rámci práce [12].

Kapitola 3

Box Clustering Segmentation

Předmětem této práce je implementace algoritmu Box Clustering Segmentation (BCS), který patří do rodiny *Visual-based* metod, jež jsou založeny na práci s vizuálními hodnotamy elementů stránky (viz 2.2). Tato kapitola se algoritmem BCS zabývá podrobněji z teoretické stránky. V první části je algoritmus obecně představen (podkapitola 3.1), v následujících podkapitolách (3.3, 3.4, 3.6) jsou pak podrobně vysvěleny jednotlivé kroky výpočtu a na závěr jsou zmíněny výsledky, kterých algoritmus dosáhl během experimentálního testování a shrnutí možného budoucího postupu v jeho vývoji.

Veškeré vzorce použité v této kapile pochází z článku [17], který byl společně s [15] hlavním zdrojem pro napsání této kapitoly.

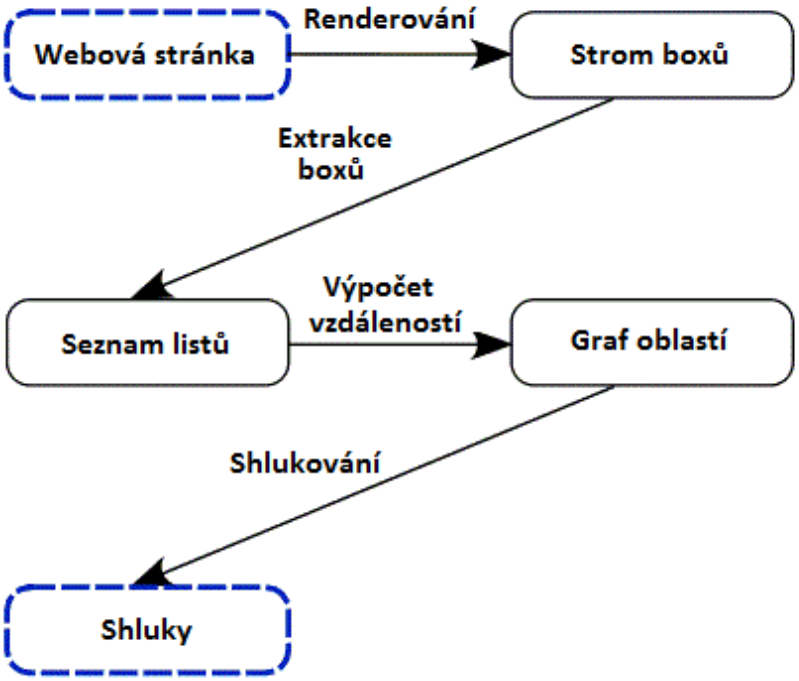
3.1 Úvod do algoritmu

U segmentačních metod je zvykem, že pracují nad stromovými strukturami. Vstupem bývá často strom vizuální oblastí webového dokumentu a výstupem je obvykle strom reprezentující hierarchii segmentů, kde jednotlivé úrovně stromu odpovídají jednotlivým stupňům zanoření. Náš algoritmus přichází s novým přístupem, kdy výstupem je množina vygenerovaných dlaždic (shluků) získaných z webové stránky za pomoci jen velmi základních vizuálních vlastnsotí jednotlivých vstupních boxů. Tento přístup vychází z myšlenky, že uživatel nevnímá stránku jako hierarchii oblastí, ale jako skupinu oblastí (dlaždic), které jsou významově odděleny.

V kapitole 2.2.1 byla obecně představena metoda VIPS, která je v současnosti jednou z nejznámějších segmentačních metod. Účelem algoritmu BCS je dosažení stejně kvalitních výsledků jakých dosahuje metoda VIPS za podstatně kratší dobu [17]. Jedním z hlavních cílů algoritmu BCS je tedy podstatně vyšší rychlost výpočtu v porovnání s ostatními algoritmy z rodiny *Visual-based* metod. Toho je dosaženo přístupem, kdy vybíráme pouze ty oblasti, které jsou důležité z pohledu uživatele, prováděním výpočtů pouze za pomocí velmi základních vizuálních rysů a zvolením vhodných struktur pro reprezentaci dat během výpočtu (viz dále). Během provádění BCS není použit přístup využívající zpracovávání stromových struktur. Místo toho algoritmus spoléhá na techniku shlukování a vhodný model podobnosti (*similarity model*).

Proces výpočtu metody BCS je zobrazen na obrázku 3.1. Jednotlivé boxy reprezentují stavy, ve kterých se nacházejí zpracovávaná data a šipky představují akce, které jsou nad daty prováděny.

Vstup celého procesu je jedna webová stánka, nad kterou je provedeno renderování.



Obrázek 3.1: Architektura algoritmu BCS [17]

Tento krok se považuje za předzpracování a není součástí samotného algoritmu BCS. Renderování je nezávislé na naší metodě a je možno použít více variant. V naší implementaci předpokládáme použití renderovacího enginu CSSBox, ale ten může být bez problémů nahrazen jiným nástrojem za předpokladu, že bude poskytovat data ve stejném formátu (stromu boxů).

Samotná metoda BCS se skládá ze tří základních kroků. V první fázi se provádí extrakce boxů (*box extraction*), ve které jsou vybrány ty oblasti, které nás zajímají (mají nějaký význam z hlediska extrakce informací). Druhý krok zařizuje vypočítávání vzdáleností (*distance computation*), kdy jsou určeny vztahy (vzdálenosti) mezi jednotlivými oblastmi získanými v předchozím kroku. Výstupem tohoto kroku je graf oblastí (*Area graf*). Ve třetí fázi je pak prováděno shlukování oblastí na základě vztahů (vzdáleností) mezi nimi.

3.2 Renderování

Renderování transformuje vstupní dokument reprezentovaný DOM stromem a dalšími dodatečnými dokumenty, jako jsou CSS soubory, na strom boxů, ktrerý je vhodný pro zobrazení a z hlediska struktury odpovídá vstupnímu dokumentu.

Samotný strom boxů se skládá z obdélníků (boxů), které reprezentují elementy na renderované stránce. Tyto obdelníky mohou být vzájemně zanořeny a vytvářejí tedy strom, kdy vztah předek - následník znázorňuje, že následník je na stránce obsažen v předkovi. Listy stromu představují základní elementy. Jedná se například o obrázky nebo řádky textu. Ostatní uzly pak zabalují skupiny ploch do větších elementů, příkladem může být paragraf textu.

Vytváření stromu boxů však není považován za součást samotného algoritmu BCS a proto nás v této práci konkrétněji vnitřní realizace renderování příliš nezajímá. Z našeho pohledu je důležitá pouze validnost výstupního stromu boxů.

3.3 Extrakce boxů

Pokud nepočítáme renderování, pak je extrakce boxů prvním krokem algoritmu Cluster Based Segmentation. Vstupem je strom boxů vegenerovaný během předzpracování a výstupem je množina boxů, které obsahují pouze ty informace, jež nás zajímají (viz dále). Tyto odlehčené boxy jsou základním stavebním blokem našeho přístupu a výsledné shluky jsou vytvářeny právě z nich.

Boxy obsahují ty informace, které nás budou zajímat během vypočítávání modelu vzdálenosti (viz kapitola 3.5). Jedná se o tyto rysy:

- barva
- pozice v rámci stránky
- velikost a tvar

Vzhledem k tomu, že nás zajímají jen ty oblasti, které mají potenciál nést relevantní informace, je třeba provést selekci, během níž jsou vybrány pouze ty boxy, jež jsou důležité z hlediska dalších výpočtů. Ta se uskutečňuje prostřednictvím rekurzivního algoritmu, který provádí pre-order inspekci vstupního stromu boxů. Následující čtyři kroky, převzaté z [17] vysvětlují způsob, kterým se vybírají užitečné listové boxy, se kterými se bude pracovat v dalších fázích algoritmu.

1. Textové boxy jsou považovány za užitečné a jsou automaticky vybrány
2. Obrázkové boxy jsou považovány za užitečné a jsou automaticky vybrány
3. Boxy, které nemají potomky a nespadají do předchozích kategorií, jsou vyřazeny a v dalších výpočtech ignorovány
4. Boxy s jedním potomkem, které nepatří do předchozích kategorií, jsou považovány za podstromy, které jsou posléze prohledány na větve. Pokud nejsou žádné nalezeny, tak se vybere nejmenší box v podstromu s neprůhledným pozadím. Pokud takový neexistuje, tak je vrácen listový uzel.

Po provedení výše popsaných kroků je teoreticky možné, že některé z vybraných boxů budou obsahovat jiné boxy, což v metodě BCS není povoleno. Je tedy třeba provést ještě dodatečnou detekci těchto výskytů a případně je odstranit.

Na závěr sekce ještě uvedeme přesný popis struktury boxu v rámci definice 1, kterou budeme používat v následujících kapitolách.

Definice 1 *Nechť box je n-tice m = (top, bottom, left, right, barva), kde top, bottom, left a right jsou celočíselné hodnoty reprezentující pozici boxu. Pak šířka boxu je rovna right - left a výška je rovna bottom - top. Barva reprezentuje dominatní barvu boxu (její určení je popsáno níže).*

3.4 Propojení boxů

Po získání množiny užitečných boxů následuje určení jejich vzájemných vztahů. Vypočítané vztahy dále využijeme během shlukování (viz kapitola 3.6). Prvním krokem je určení relace polo-zarovnání mezi všemi boxy, která je definována v 2 Relace polo-zarovnání (*semi-aligment*) je jedním ze základních principů algoritmu BCS.

Definice 2 *Nechť m a n jsou boxy. Předpokládané překrytí boxů m a n je definováno jako funkce pov:(m, n) → x, y, o, kde hodnoty x a y značí předpokládaný překryv na příslušných osách webového dokumentu a hodnota o vyjadřuje, že k překryvu nedochází. Funkce je definována následovně:*

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

Říkáme, že boxy m a n jsou v relaci polo-zarovnání, pokud pro ně platí pov(m, n) ≠ o.

Po vypočítání funkce *pov()* můžeme pro ty dvojice boxů, které jsou v relaci polo-zarovnání, určit jejich vzájemnou pozici. Její výpočet a význam je popsána níže v rámci definice 3.

Definice 3 *$P = \{a, b, l, r, o\}$ je množina možných vzájemných pozic mezi dvěma boxy. Její jednotlivé hodnoty jsou definovány následovně:*

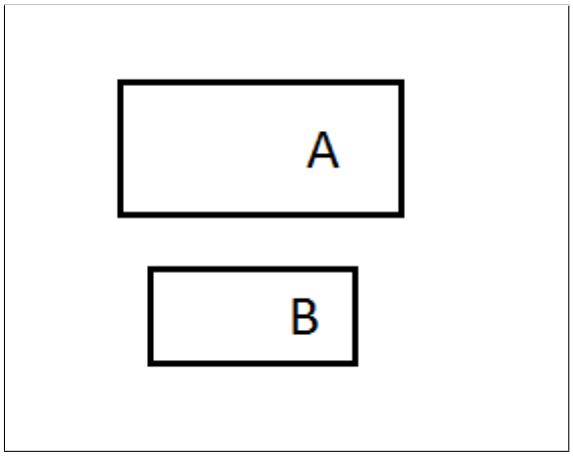
- *a - above (nad)*
- *b - below (pod)*
- *l - left (vlevo)*
- *r - right (vpravo)*
- *x - undefined (nedefinováno)*

Vzájemná pozice je počítána pomocí funkce pos: (m, n) → P :

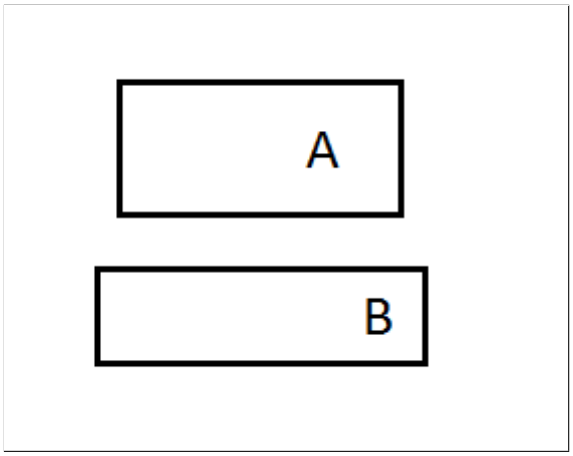
$$pos(m,n) = \begin{cases} a, & m.bottom \leq n.top \wedge pov(m,n) = x \\ b, & m.top \geq n.bottom \wedge pov(m,n) = x \\ l, & m.bottom \leq n.left \wedge pov(m,n) = y \\ r, & m.bottom \geq n.right \wedge pov(m,n) = y \\ o, & jinak \end{cases} \tag{3.2}$$

Prostřednictvím definic 2 a 3 můžeme vyjádřit vztah mezi libovolnými dvěma boxy v rámci stránky. Relace polo-zarovnání je mimo jiné používána při počítání relativní vzdálenosti (viz definice 6).

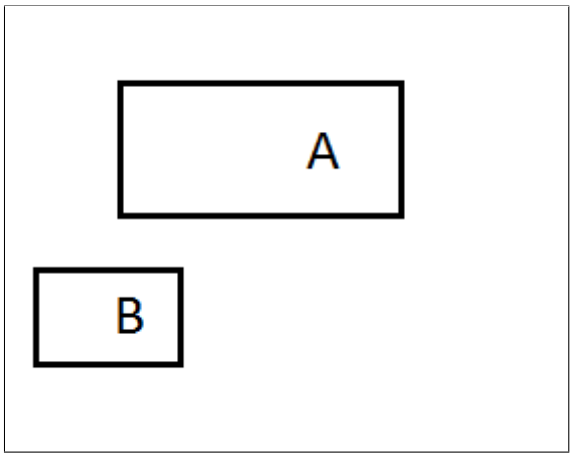
Pro lepší přehlednost jsou příklady, kdy pro dva boxy platí, že jejich vzájemná pozice je *above* (tedy *pos(A, B) = a*) ukázány na obrázcích 3.2 až 3.5. Nutno podotknout, že na těchto i dalších obrázcích je počátek souřadného systému v levém horním rohu a hodnoty na ose y rostou směrem dolů.



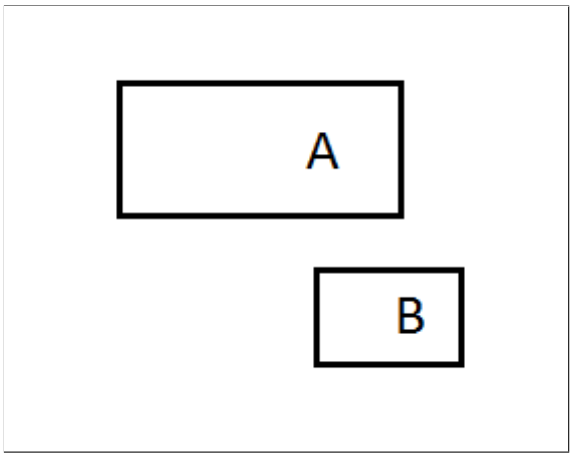
Obrázek 3.2: *pos(A, B) = a*



Obrázek 3.3: *pos(A, B) = a*



Obrázek 3.4: *pos(A, B) = a*



Obrázek 3.5: *pos(A, B) = a*

Po výpočtu vzájemné pozice mezi všemi boxy je dalším krokem určení množiny přímých sousedů pro jednotlivé boxy. Jedná se o množinu prvků, které jsou z hlediska pozice boxu nejblíže a tedy jsou nejpravděpodobnější kandidáti na sloučení (viz dále). Pro určení této množiny potřebujeme mít možnost určit absolutní vzdálenost mezi dvěma boxy. Způsob jejího výpočtu je ukázán v definici 4.

Definice 4 *Nechť m a n jsou boxy v rámci webové stránky. Absolutní vzdálenost mezi nimi je potom definována jako funkce abs: (m, n)* $\rightarrow \mathbb{R}$ *:*

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

Samotné určení množiny přímých sousedů je pak popsáno níže v definici 5.

Definice 5 *Množina přímých sousedů je* $N_m = \{n | n \in B \wedge pos(m,n) \neq o \wedge \nexists k \in B : pos(m,k) = pos(m,n) \wedge abs(m,k) < abs(m,n)\}$

Prvky množiny blízkých sousedů jsou takové dvojice, které mají vysokou pravděpodobnost na sloučení. Při vytváření shluků (viz kapitola 3.6) budou brány jako výchozí kandidáti na sloučení . Během výpočtu jsou doplňovány prvky z množiny přímých sousedů shluků (viz definice 5).

3.5 Model podobnosti

V předchozí podkapitole bylo zmíněno, že v rámci vytváření grafu oblastí se každé hraně mezi oblastmi přiděluje hodnota z množiny reálných čísel. Tato sekce podrobněji popisuje způsob, jak se tato hodnota získá a jakým způsobem se s ní dále pracuje.

Již dříve bylo zmíněno, že v naší metodě používáme dvouúrovňovou strukturu, kde základní úroveň je skupina boxů (listové uzly získané během 3.3) a nad ní je definovaná úroveň shluků, které obsahují jeden nebo více boxů. V rámci této práce tedy budeme rozlišovat termíny box a shluk (*cluster*).

Když v souvislusti s algoritmem BCS mluvíme o podobnosti, tak se jedná o jeden ze dvou případů. První nazýváme základní podobnost (*base similarity*), která se odvozuje z vizuálních vlastností boxů a druhou je shluková podobnost (*cluster similarity*), jež vyjadřuje podobnost mezi dvěma shluky nebo mezi shlukem a boxem.

Podle hodnoty podobnosti se v následujících krocích algoritmu BCS rozhodujeme, které boxy budou patřit do stejného shluku. Proces shlukování je podrobněji definován v kapitole 3.6.

3.5.1 Základní podobnost

Vstupem výpočtu základní podobnosti jsou dva boxy, nad kterými se z několika základních vizuálních rysů spočítá hodnota představující jak moc jsou si podobné. Tato hodnota může být vypočítána pro libovolné dva boxy, ale ve skutečnosti se bude počítat jen pro ty, pro které to má význam. Vypočítání základní podobnosti se skládá ze tří na sobě nezávislých částí. Tyto části jsou odvozené z vlastností boxů představených v kapitole 3.3. Jedná se o vzdálenost, tvar a barvu.

Vzdálenost

Pro vypočítání vzdálenostní hodnoty mezi dvěma boxy je třeba nejprve spočítat absolutní vzdálenost, jež je definována v 4.

Z absolutní vzdálenosti následně vypočítáme vzdálenost relativní, která určuje vzdálenosti mezi boxy vzhledem k jejich nejbližším sousedům. Hodnota relativní vzdálenosti se pohybuje v intervalu <0, 1> a její výpočet lze nalézt v definici 6.

Definice 6 *Nechť B je množina všech boxů na webové stránce. Pro každý box* $b \in B$ *a jeho množinu přímých souseddů* N_m *(definovné výše) je maximální vzdálenost souseda* $maxd(n) = abs(m, k)$*, kde* $k \in N_m \wedge \nexists l \in N_m : abs(m,l) > abs(m,k)$*. Pro všechny* m *a* $n \in B$ *platí, že vzdálenost (distance) mezi nimi je vypočítána jako:*

$$rel_m(m,n) = \frac{abs(m,n)}{maxd(m)} \tag{3.4}$$

$$rel_n(m,n) = \frac{abs(m,n)}{maxd(n)} \tag{3.5}$$

$$distance(m,n) = \frac{rel_m + rel_n}{2} \tag{3.6}$$

Tvar

Naše metoda předpokládá, že u boxů, které jsou si velikostně tvarově podobné je pravděpodobnější, že budou patřit do stejného shluku. Pro vypočítání podobnosti na základě tvaru použijeme dva aspekty. Jedná se o poměr (*ratio*) a velikost (*size*). Jejich výpočet je popsán níže.

$$r_m = \frac{m.right - m.left}{m.bottom - m.top} \tag{3.7}$$

$$r_n = \frac{n.right - n.left}{n.bottom - n.top} \tag{3.8}$$

$$ratio(m,n) = \frac{max(r_m, r_n) - min(r_m, r_n)}{\frac{max(r_m,r_n)^2 - 1}{max(r_m,r_n)}} \tag{3.9}$$

$$s_m = (m.right - m.left) * (m.bottom - m.top) \tag{3.10}$$

$$s_n = (n.right - n.left) * (n.bottom - n.top) \tag{3.11}$$

$$size(m,n) = 1 - \frac{min(s_m, s_n)}{max(s_m, s_n)} \tag{3.12}$$

$$shape(m,n) = \frac{rartio(m,n) + size(m,n)}{2} \tag{3.13}$$

Nad těmito hodnotami je pak třeba provést klasický aritmetický průměr, jehož výsledek představuje podobnost na základě tvaru (*shape*) mezi vstupními boxy.

Barva

Jednotlivé boxy mohou obsahovat více barev. V případě textových boxů se mouhou lišit barvou fontu, okrajů a pozadí a obrázkové boxy se obvykle skládají z velkého množství barev. Pro účely výpočtu je třeba, aby každý box byl reprezentován pouze jednou barvou. Způsob jímž je tohoto docíleno je následující:

- obrázky - tón obrázku
- textové boxy - barva fontu

- ostatní - barva pozadí

Pro reprezentaci barvy použijeme klasický model RGB. Důvod, proč byl zvolen právě tento model, je blíže popsán v [17]. Samotný výpočet podobnosti na základě barvy (*color*) je pak následující:

$$color(n,m) = \frac{\sqrt{(R_n - R_m)^2 + (G_n - G_m)^2 + (B_n - B_m)^2}}{1.732} \tag{3.14}$$

Nyní již známe všechny komponenty potřebné pro vypočítání základní podobnosti mezi dvěma boxy. Celkový vzorec je uveden níže.

$$sim(m,n) = \begin{cases} 0, & distance(m,n) = 0 \\ 1, & distance(m,n) = 1 \\ \frac{distance(m,n)+shape(m,n)+color(m,n)}{3}, & jinak \end{cases} \tag{3.15}$$

3.5.2 Shluková podobnost

Pro účely slučování shluků a přidávání boxů do shluků je třeba, abychom byli schopni vypočítat hodnotu podobnosti mezi dvěma shluky a shlukem a boxem. Tato hodnota je odvozena z vlastností boxů, které shluk obsahuje a je třeba, aby každý box k ní přispěl přiměřeně vzhledem k tomu, jak velký má podíl na podobě celého shluku. Tedy box, který zabírá velké množství prostoru v shluku, by měl ovlivnit hodnoty pobodobností úměrně vzhledem ke své velikosti.

Podobně jako v případě boxů nebudeme počítat podobnost mezi shlukem a všemi elementy stránky. Z důvodu úspory času ji budeme počítat jen pro ty elementy, které mají šanci na sloučení. Obdobně jako u boxů (viz definice 5) si zavedeme množinu přímých sousedů shluku.

Definice 7 *Nechť B je množina boxů a C je množina shluků. Dále $B_c \in C$ je množina boxů patřící do shluku c a N_m je množina přímých sousedů boxu m. B_U je pak množina boxů na stránce, které nepatří do žádného shluku. Tedy $B_U = \{b | b \in B; \nexists B_c \in C : (b \in B_c)\}$*

Pro shluk c potom platí, že jeho množina přímých sousedů je definována jako $N_c = \{m | m \in B_U \wedge \exists n \in B_c : n \in N_m\} \cup \{B_d | D_d \in C \wedge \exists m \in B_c : n \in N_m\}$.

Pro výpočet podobnosti mezi shlukem c a boxem b použijeme rovnici 3.16, která představuje pomocné funkce *cumul()* a *card()*, definované níže. Funkce *card()* v podstatě počítá, kolik boxů z c patří do množiny přímých sousedů boxu b a *card()* spočítá sumu podobností mezi těmito boxy.

$$csim(c,b) = \frac{cumul(c,b)}{card(c,b)} \tag{3.16}$$

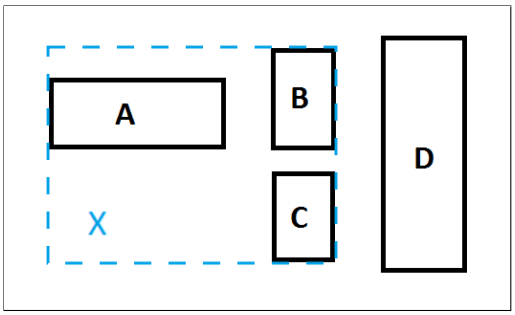
$$card(c,e) = |\{m | m \in B_c \wedge e \in N_m\}| \tag{3.17}$$

$$cumul(c,b) = \sum_{\forall m \in B_c} s(m,e) \tag{3.18}$$

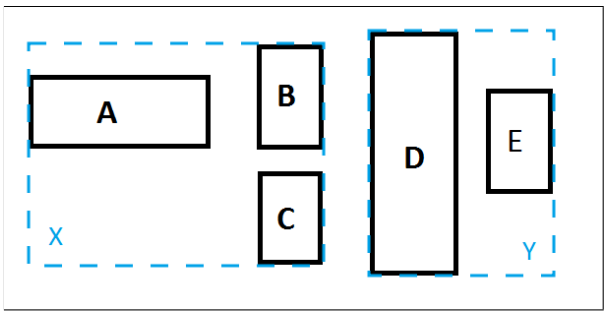
Výše popsané vzorce jsou použitelné i v případě, kdy počítáme podobnost mezi dvěma shluky. Konkrétní příklady jsou pro snadnější pochopení ukázány na obrázcích 3.6 a 3.7. Pro které platí:

$$sim(X, D) = (sim(B, D) + sim(C, D))/2 \tag{3.19}$$

$$sim(X, Y) = (sim(B, Y) + sim(C, Y))/2 = (sim(B, D) + sim(C, D))/2 \tag{3.20}$$



Obrázek 3.6: Podobnost mezi boxem a shlukem



Obrázek 3.7: Podobnost mezi shlukem a shlukem

3.6 Shlukování oblastí

Vstupem je množina boxů B, množina hran mezi boxy E a funkce podobnost sim. Výstupem je pak množina, která obsahuje jednotlivé shluky, ta je zároveň výstupem celého algoritmu BCS.

Samotné shlukování se skládá ze čtyř částí:

1. Vytvoření shlukových zrn (*cluster seed*)
2. Vybrání entit pro sloučení
3. Ošetření překrývání
4. Potvrzení shluku

Hlavní algoritmus vytváření grafu shluků je popsán ve funkci *get_clusters()*. Vytváření shlukových zrn a vybírání entit pro sloučení je pokryto v hlavní smyčce. Obě části jsou principiálně stejné. V obou případech se vytváří nový shluk. Rozdíl je v tom, že v případě, kdy ho vytvářím ze dvou boxů, vznikne shlukové zrno a pokud je alespoň jedna ze vstupních entit shluk, tak říkáme, že vytváříme kandidáta pro shluk [17]. Nicméně praktický rozdíl mezi zrnem a kandidátem je minimální a v dalším textu je tedy nebudeme striktně rozlišovat.

Výběr dvou entit pro sloučení je realizován ve funkci *find_relation()*, která jednoduše najde dvě entity spojené hranou s nejnižší existující hodnotou podobnosti.

Po nalezení této dvojice provedeme kontrolu, zda jejich hodnota podobnosti je větší než hodnota CT. CT (*Clustering treshhold*) je hodnota v intervalu <0, 1>, která je definována pro celou stránku. Tento parametr je podobný proměnné $Pdoc$ v algoritmu VIPS (viz kapitola 2.2.1) a jeho správné nastavení je stěžejní pro získání dobrých výsledků. Vhodná hodnota se liší v závislosti na typu zpracovávaných stránek. O možných hodnotách CT je napsáno více v [17].

Algoritmus 1

function get_clusters(B, E, sim):
$R = \{(e, x) : e \in E, x = sim(e)\}$
$C = \emptyset$
while $R \neq \emptyset$ *:*
    (e, r) = find_relation(R)
    if $r = \infty$ *:*
        break
    d = get_graph_similarity(e, r)
    id $d >$ CT *:*
        continue
    cluster = merge(e)
    if overlaps(C, cluster) :
        continue
    if overlaps(B, cluster) :
        merge_overlaps(B, cluster) :
    commit(C, R, cluster) :
return C

Po vytvoření kandidáta na nový shluk (alternativně shlukového zrna) je provedeno několik kontrol. Až poté co jimi kandidát (zrno) projde je vytvořen nový shluk. Jedná se o následující testy:

1. Kontrola zda se kandidát nepřekrývá s již existujícím shlukem
2. Kontrola zda se kandidát nepřekrývá s některým z boxů

Pokud kandidát neprojde prvním krokem je automaticky zrušen, v případě kdy neprojde druhým, se pokusíme box a kandidáta sloučit, pokud výsledek sloučení nezpůsobí další překrytí, tak ho akceptujeme a potvrdíme vytvoření nového shluku, které proběhne ve funkci *commit()*.

Při vytvoření nového shluku je třeba provést několik činností:

1. Všechny nové boxy přidané do kandidátu jsou označeny jako členy skutečného shluku
2. Vztahy mezi novými členy jsou odstraněny z množiny vztahů E

3. Vztahy mezi novými členy a boxy nacházejícími se mimo nový shluk jsou aktualizovány na hodnotu vztahu mezi samotným shlukem a vnějšími boxy.
4. Duplicitní vztahy vytvořené v předchozím kroku jsou odstraněny
5. Vnitřní indikátory podobnosti jsou přepočítány (viz kapitola 3.5.2)
6. Pokud byl kandidát vytvořen z jiných shluků, tak jsou smazány z množiny shluků
7. Nový shluk je vložen do množiny shluků

Po dokončení posledního kroku algoritmu je možné, že některé boxy nebudou patřit do žádného shluku (jejich počet závisí na zvolené hodnotě CT), tyto boxy považujeme za nedůležité z hlediska extrakce informací a do výsledku je tedy nezahrnujeme.

3.7 Vyhodnocení algoritmu

V rámci práce [17] byl algoritmus Cluste Based Segmentation implementován a následně testován. Dosažené výsledky lze ve většině případů přirovnat z hlediska kvality k výsledkům algoritmu VIPS. Nicméně metoda měla v některých situacích, kdy zpracovávala komlexnější stránku, problém určit správné kandidáty pro sloučení do shluku.

Pro podrobnější informace o výsledcích testování je možné nahlédnout do [17]. Co se týče možností pro další vývoj, tak inženýr Zelený v [17] zmiňuje, že práce na algoritmu se v budoucnosti bude soustředit na vytváření více variant pro počítání funkce podobnosti.

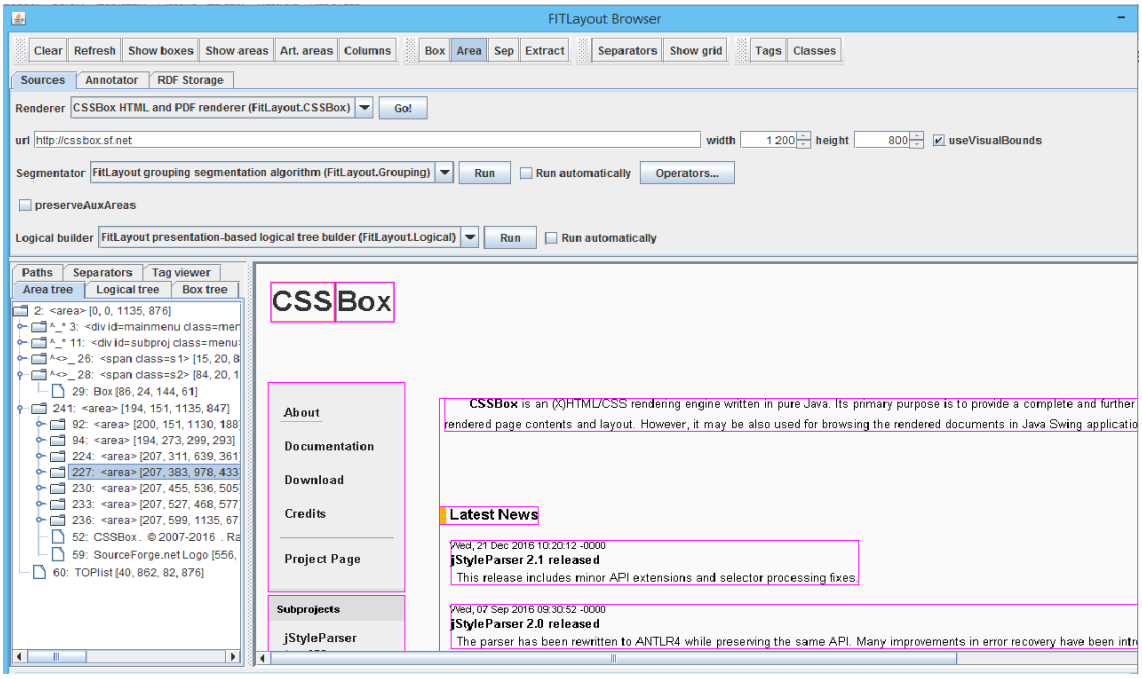
Testovaní naší implementace algoritmu je věnována kapitola 6, kde jsou dosažené výsledky porovnány mimo jiné i s těmi z [17].

Kapitola 4

FITLayout

FitLayout je framework napsaný v jazyce Java sloužící k segmentaci a následné analýze webových stránek. Základní dokumentace k tomuto nástroji je přístupná na [4], manuál je možné nalézt na stránce [5] a vygenerovaná javadoc dokumentace je zveřejněná na [6].

Jak bylo zmíněno výše, cílem této práce je implementovat algoritmus Cluster Based Segmentation (popsaný v kapitole 3) do tohoto nástroje. V této sekci obecně popíšeme framework, jeho moduly a způsob jakým pracuje. V rámci kapitoly je naznačeno jakým způsobem se bude dát postupovat při přidání nové segmentační metody do FITLayoutu.

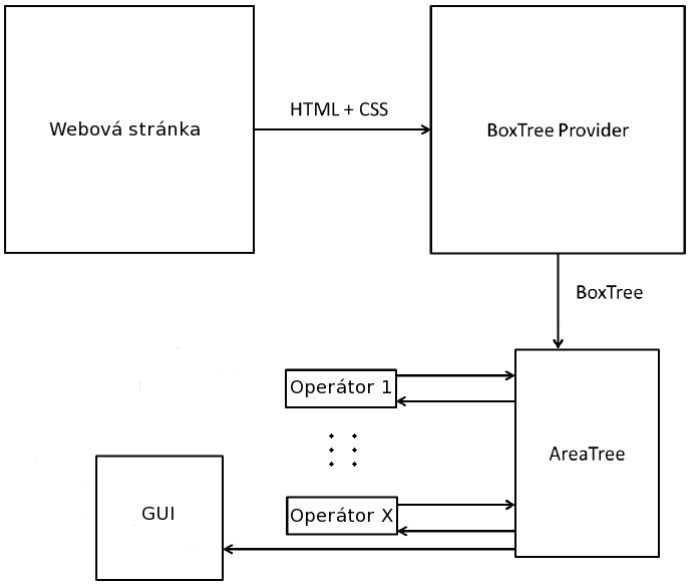


Obrázek 4.1: GUI nástroje FITLayout

Nástroj FITLayout je vyvíjen na Fakultě Informačních technologií VUT v Brně pod licencí GNU General Public License [4]. Součástí frameworku je uživatelské rozhranní (viz obrázek 4.1), engine pro renderování stránky, vzorová segmentační metoda a další nástroje pro práci s výsledky i mezivýsledky segmentačního procesu. Výsledky segmentace lze dále ukládat ve formátu RDF. Celý framework je kontstruován tak, aby bylo snadné kteroukoliv

část upravit, případně nahradit jinou variantou.

Na obrázku 4.2 je možné vidět obecný průběh zpracovávání stránky, jež do značné míry odpovídá principu ukázaném v kapitole 2. Z našeho hlediska je důležitý modul Segmentace, ve které se nachází *AreaTreeProvider*, který poskytuje základní strom oblastí. Tento strom je dále možné modifikovat prostřednictvím Segmentačních operátorů (*AreaTreeOperator* - viz níže).



Obrázek 4.2: Průběh výpočtu nástroje FITLayout

4.1 Moduly

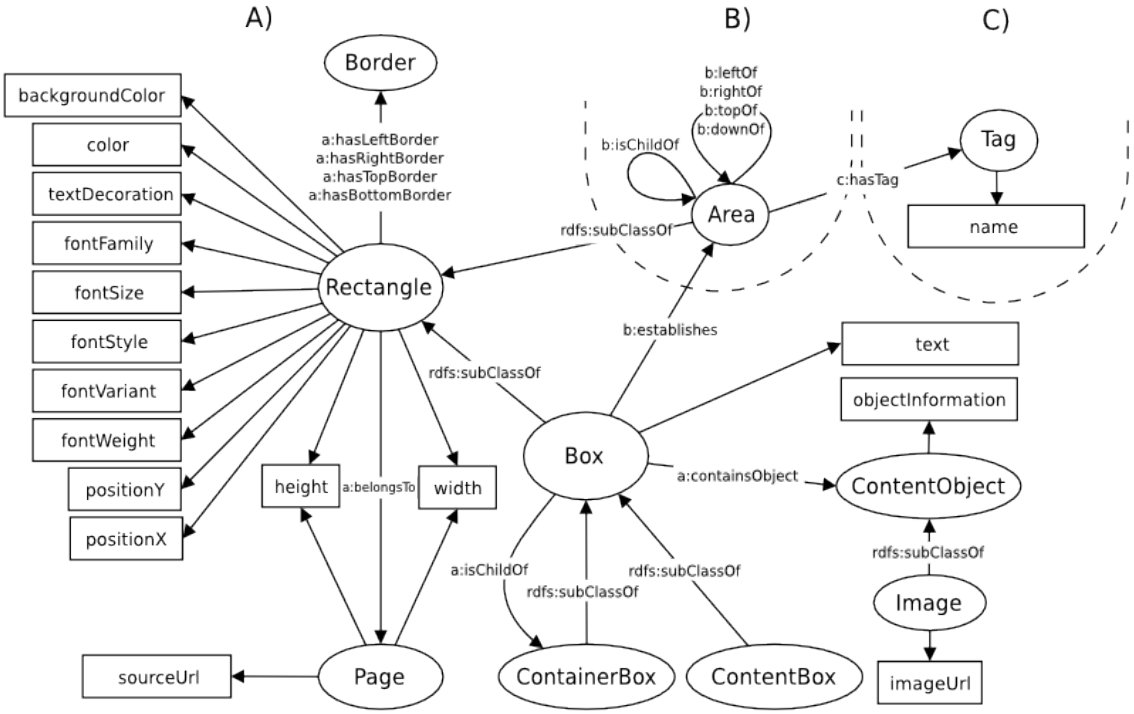
Vzhledem k velikosti projektu a k tomu, že jedním z cílů je možnost snadného rozšíření, byl framework rozdělen do několika modulů. Jedná se o:

API

Zajišťuje propojení mezi moduly. API poskytuje obecná rozhraní a jejich implementace definující aplikační rozhraní pro segmentační metody. Související ontologie je znázorněna na obrázku 4.3.

CSSBox

Jedná se o HTML-CSS renderovací engine. Jeho primární cíl je poskytnout kompletní a dále zpracovatelné informace o obsahu a rozložení webové stránky [3]. Vstupem je webová stránka reprezentovaná DOM stromem a souvisejícími kaskádovými styly. Výstupem je pak model reprezentující rozložení stránky (strom boxů), který je možné přímo zobrazit nebo dále zpracovávat za účelem extrahování informací.



Obrázek 4.3: API ontologie [6]

Přestože je CSSBox součástí frameworku FITLayout, kde je používán jako výchozí nástroj pro renderování, je možné ho bez problémů použít i mimo něj.

Segmentation

Součástí modulu je obecná kostra pro implementaci segmentačních metod. Samotná segmentace je prováděna operátory. Implicitní algoritmus použitý ve frameworku je založen na postupu popsaném v [13].

Skutečnost, že segmentace se provádí prostřednictvím operátorů značí, že není problém vytvořit a přidat operátor vlastní. Což zajišťuje jednoduchou rozšiřitelnost frameworku FITLayout o další segmentační metody.

Výstupem segmentace i všech operátorů je strom vizuálních oblastí (*AreaTree*).

Tools

Nástroje pro běh a řízení segmentace stránky. Do tohoto modulu patří uživatelské rozhraní (viz obrázek 4.1) a processor, který zajišťuje provedení celého segmentačního procesu od zpracování stránky po uložení výsledků.

Classification

Umožňuje klasifikovat oblasti získané během procesu segmentace prostřednictvím tagů.

4.2 Služby

Architetura FITLayoutu byla narvžena tak, aby šla snadno rozšířit o nové pluginy poskytující alternativní přístup k vytváření vniřních struktur pro reprezentaci mezivýsledků během celého precesu zpracovávání stránky [5]. Framework umožňuje definovat následující služby:

- BoxTreeProvider - poskytuje strom boxů (*BoxTree*)
- AreaTreeProvider - poskytuje strom oblasttí (*AreaTree*)
- AreaTreeOperator - operátor služící ke zpracování AreaTree
- LogicalTreeProvider - poskytuje strom logických oblasttí (*LogicalTree*)

Pro správu služeb je vývojáři poskytnut *ServiceManager*.

Kapitola 5

Implementace

V předchozích kapitolách byla rozebrána hlavní teoretická část naší práce. Jedná se především o kapitoly 3 a 4, kde jsme popsali metodu BCS a framework FITLayout. V této sekci na ně navážeme a budeme se zde věnovat způsobu jakým byla metoda BCS implmentována v rámci prostředí nástroje FITLayout.

Framework FITLayout je veřejně dostupný prostřednictvím několika vzájemně dostupných projektů napsaných v jazyce Java v rámci vývojového prostředí Eclipse (viz [3]). Je tedy logické že naše implementace bude využívat stejných prostředků. V rámci práce byl vytvořen projekt *CBSSegmentation*, který umožňuje aplikovat segmentační metodu BCS za pomocí FITLayoutu.

V této části práce je popsáno rozhraní projektu a v podkapitole 5.1.2 pak postup pro instalaci, zprovoznění a použití projektu.

Kapitola 5.2 obsahuje popis způsobu, jakým jsme implementovali samotnou metodu BCS. Je zde ukázána a podrobně vysvětlena naše verze smyčky z funkce *get_clusters* (viz sekce 3.6). Mimo to se v podkapitolách 5.2.5 a 5.2.4 blíže věnujeme problémům, které byly v předchozím textu pouze hrubě naznačeny. Konkrétně se jedná o kontrolu na překrytí mezi prvky a přepočítávání množin přímých sousedů pro jednotlivé shluky.

Na kapitolu Implementace dále navazuje kapitola 6, kde je popsán proces testování a dosažené výsledky.

V rámci kapitoly věnující se FITLayoutu (viz 4) bylo naznačeno, že k implementaci nové varianty pro segmentaci lze přistoupit prostřednitvím vytvořením nového segmentačního operátoru (*AreaTreeOperator*). Nicméně v případě algoritmu Box Clustering Segmentation narážíme na fakt, že jeho předpokládaný vstup není typický strom oblastí (*AreaTree*), ale pouze množina odlehčených boxů, která již proběhla výše popsanou selekcí. V případě implementace prostřednictvím by se tedy zbytečně prováděl krok předzpracování.

Alternativní přístup je vytvoření nového *AreaTreeProvideru*, jehož vstupem by byl strom boxů a který by prováděl celý algoritmus.

Po konzultaci bylo rozhodnuto, že bude použit první způsob a to z důvodu zachování jednotného přístupu k přidávání operátorů. Implementace *AreaTreeOperatoru* byla zvolena i proto, že cílem práce není vytvoření segmentačního procesu, který by pracoval rychleji než ty stávající a tedy krok předzpracování, který je proveden navíc, nevadí.

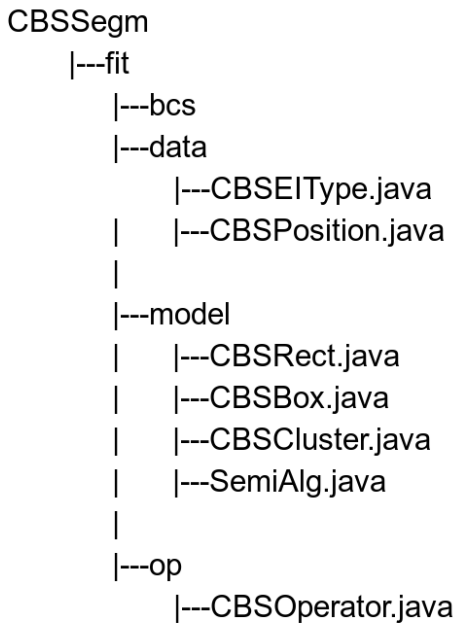
5.1 Projekt CBSSegmentation

Veškerá funkcionalita popsaná v následujících sekcích je implementována v rámci samostatného projektu *CBSSegmmentation*, který je napsán v jazyce Java. Tento projekt je veřejně dostupný na stránce GitHub na adrese *https://github.com/xlenga01/BCSSegmentation*.

Tato kapitola popisuje rozhraní a obecnou strukturu projektu *CBSSegmentation*, tedy z jakých balíčků a tříd se projekt skládá (kapitola 5.1.1). V sekci 5.1.2 jsou pak popsány kroky, které je třeba provést pro zprovoznění projektu.

5.1.1 Struktura a model

Na obrázku 5.1 je možné vidět strukturu projektu *CBSegmentation*. Ten se skládá ze tří balíčků. V balíčku *fit.bcs.data* jsou uloženy konstanty a výčty společně se základními operacemi, jež je nad nimi možné provádět.



Obrázek 5.1: Struktura projektu CBSSegmentations

Balíček *fit.bcs.model* obshuje třídy, které slouží k reprezentaci níže popsaných entit. Rozhraní *CBSRect* je použito pro reprezentaci prvků (boxů a shluků). *CBSRect* implementuje rozhraní *Rect* z projektu *LayoutAPI* (viz [3]). Rozhraní *Rect* bylo zvoleno z toho důvodu, že metoda BCS nevyžaduje znát velké množství informací o elementech stránky (stačí nám pozice a barva). Pro zachování myšlenky jednoduchosti a rychlosti metody, jsme při výbírání vhodného rozhraní hledali takové, které nevyžaduje velké množství metod pro implementaci, ale které nabízí základní funkcionalitu pro práci s obdelníkem v prostoru.

Pro reprezentaci samotných boxů a shluků jsou použity objekty tříd *CBSBox*, respektive *CBSCluster*.

Objekty třídy *SemiAlg* jsou použity k reprezentaci dvojice elementů (*CBSRect*) v relaci *polo-zarovnání* (viz 2), případně se používají pro vyjádření vztahu přímých sousedů (viz

dále). Při provádění kódu hlavní smyčky (viz 5.2.3) pak slouží k reprezentaci kandidántů pro sloučení.

Třída *CBSOperator* v balíčku *fit.bcs.op* reprezentuje samotný operátor provádějící metodu BCS. Třída implementuje rozhraní *BaseOperator*, jehož součástí je funkce *apply()*, která na vstupu dostane strom oblastí, a která se stará o samotnou aplikaci operátoru (viz dále). Velká část výpočtů popsaných v kapitole 3 je prováděna právě v rámci této metody (včetně hlavní smyčky rozebrané v 5.2.3).

Nástroj FitLayout kromě možnosti definovat vlastní operátory umožňuje definování jejich parametrů. Náš operátor pracuje se dvěma takovými parametry. Jedná se o *CT* (defaultně nastaven na hodnotu 0.5) a *calcImgColor* (defaultně nastaven na *false*). Parametr *CT* představuje práh shlukování (*Cluster treshhold*), který ovlivňuje zrnitost výstupu operátoru (více se hodnotě *CT* věnujeme v kapitole 3.6). Booleanová hodnota *calcImgColor* pak určuje do jaké míry budou v rámci předzpracování zpracovány obrázky vyskytující se na segmentované webové stránce (viz dále).

5.1.2 Instalace

Jak bylo výše zmíněno, je projekt dostupný na GitHubu, kde je volně ke stažení.

Pro instalaci je nejjednodužším řešením využití nástroje Eclipse, který umožňuje celý projekt stáhnout jen za pomocí URI. Projekt lze pochopitelně spustit i v jiných vývojových prostředích než Eclipse (například prostředí NetBeans umožňuje přímý import Eclipsovských projektů), nicméně v takovém případě je nutné počítat s podstatně delší dobou na zprovoznění projektu. Je vhodné počítat s tím, že projekty implementující framework FITLayout jsou zpřístupněny stejným způsobem a proces jejich instalace je obdobný.

Projekt *CBSSegmentation* není samostatný a pro správnou funkčnost je třeba zajistit správné nastavení závislostí na projekty, ve kterých je implementován nástroj FITLayout. Pro zprovoznění FITLayoutu je možné se blíže informovat na [3].

Implicitně se předpokládá použití nástroje Maven, ale správné nastavení "Build path"jde samozřejmě docílit i jinak.

Po provedení předchozích kroků je již operátor k dispozici a lze ho vybrat vrámci GUI (viz obrázek 4.1). Při použití našeho oprátoru je třeba pamatovat na to, že je žádoucí, aby byl před ním aplikován operátor *FlattenTreeOperator* (důvod je vysvětlen v sekci 5.2).

Aby uživatel nemusel pokaždé vybírat jím preferovanou kombinaci operátorů, tak je v souboru *src\main\resources\default_operators.js* v projektu *LayoutTools* možně vybrat defaultní seznam, který bude v rámci segmentace použit.

5.2 Implementace BCS

V předchozích sekcích jsme se zabývali rozhraním projektu. Nyní se blíže podíváme na vnitřek modulu, tedy na to, jakým způsobem byly implementovány rovnice a definice nadefinovány v kapitole 3. Následující text popisuje samotný výpočet segmentace a datové struktury, které se používají v jeho jednotlivých fázích. Hlavním cílem této podkapitoly je vysvětlit způsob, jakým byla smyčka definovaná v 3.6 převedená do reálného kódu. Kromě toho se zde podrobněji věnujeme několika problémovým sekcím BCS. Jedná se především o problém přepočítávání přímých sousedů shluků, jež bylo v článku [17] pouze naznačeno. Způsobu, jakým byla tato problematika řešena, se věnujeme v sekci 5.2.5. V podkapitolách 5.2.1 a 5.2.6 jsou pak přesněji popsány vstup, respektivě výstup, celého algoritmu.

5.2.1 Vstup

Předpokládaným vstupem algoritmu je množina boxů. Jedná se o základní elementy stránky jako řádky textu a obrázky. Vzhledem k důvodům uvedeným výše jsme algoritmus BCS implementovali prostřednictvím rozhraní *BaseOperator*, což má ovšem za následek to, že používáme výchozí *areaTreeProvider* frameworku, který nevytváří množinu oblastí, ale strom. Tento strom je tedy nutné převést do námi požadované formy. Za tímto účelem se předpokládá použití bezparametrového operátoru *FlattenTreeOperator*, který je implementován v rámci projektu *LayoutSegm* a je tedy jedním ze základových operátorů ve frameworku FITLayout. Jeho činnost spočívá v předení stromu oblastí na seznam oblastí, respektive na strom o dvouch úrovních, kdy kořen reprezentuje celou stránku a v druhé úrovni jsou obsaženy všechny oblasti. Výpočet operátoru v podstatě odpovídá extrakci boxů popsané v kapitole 3.3. Důležitou vlastností výstupu operátoru je, že jednotlivé oblasti nejsou vzájemně vnořeny ani se nemohou překrývat. Toto je podstatné, neboť metoda BCS překrývání boxů neumožňuje.

Operátor *FlattenTreeOperator* je nutné aplikovat, aby BCS segmentace fungovala správně. V případě, kdy by použit nebyl, by náš kód bral v úvahu pouze první úroveň stromu oblastí a ostatní oblasti by ignoroval, čímž by byl výsledek pochopitelně značně zkreslen.

Použití našeho operátoru *CBSegm* v kombinaci s dalšími, ať vlastními nebo definovanými v základu FITLayoutu, není doporučeno, ale v zásadě je možné. Nicméně při jejich nasazení je třeba vždy pamatovat na to, že oblasti na vstupu BCS se nesmí překrývat.

5.2.2 Předzpracování

Po zajištění správného formátu vstupu, popsaného v předchozí sekci, následuje část implementována v rámci operátoru *CBSegm* (v metodě *apply()*).

Před hledáním shluků na stránce je nicméně ještě potřeba nalézt potenciální kandidáty pro spojení do shluků. Tohoto je dosaženo ve fázi předzpracování, která se skládá z následujících kroků:

1. Výpočet množiny CBSBoxů
2. Výpočet vzájemné pozice boxů
3. Výpočet přímých sousedů
4. Vypočítání podobnosti
5. Určení kandidátů pro spojení

V následujícím textu se na tyto kroky blíže podíváme.

Přestože se jedná o předzpracování, je jeho provedení časově náročnější než hlavní smyčka. To je způsobeno tím, že je potřeba porovnat každý element stránky s každým, což u stránek s velkým obsahem zabere poměrně velké množství času. Samotné spojování oblastí do shluků je pak již poměrně rychlé i v případě komplexních stránek skládajících se z velkého množství elementů, neboť pro jednotlivé elementy budeme používat pouze jejich množiny přímých sousedů (viz dále).

Výpočet množiny CBSBoxů

Po aplikaci operátoru *FlattenTreeOperator* máme již k dispozici strom oblastí, který má vhodnou strukturu. Je třeba ještě zajistit, aby pro každou oblast byly přístupné ty informace, které budeme potřebovat pro výpočet podobnosti. Jedná se o pozici a barvu (viz 3.3). Určení pozice zajišťuje již *areaTreeProvider*. To samé ovšem neplatí pro barvu, která je specifická pro segmentační metody patřící do skupiny *Visual Page* a není tedy operátory typicky vyžadována. Účelem inicializace množiny boxů je především dovypočítání barvy a převedení vstupního seznamu oblastí (objektů třídy *AreaImpl* do seznamu objektů *CBSBox*, které jsou v dalších výpočtech použity pro reprezentaci základních elementů stránky.

V podkapitole 3.5 bylo zmíněno, že v případě barvy nás zajímá buď barva textu nebo tón obrázku. Algoritmus BCS ještě zmiňuje možnost využít barvu pozadí u prvků které nejsou ani text ani obrázek, ale vzhledem k tomu, že tyto prvky nejsou důležité z hlediska extrakce informací, tak je budeme v dalších výpočtech ignorovat. Co se týče barvy textu, tak ji lze odvodit z vlastností definovaných ve třídě *AreaImpl*. V případě obrázků ovšem žádná implementovaná třída námi požadovanou barvu neobsahuje, je ji tedy třeba získat přímo ze zdroje webové stránky. Za pomocí metody *getContentObject()* je možné získat URL obrázku a následně se dá pomocí třídy *BufferedImage* dostat k jednotlivým pixelům.

Vypočítání barvy pro obrázky je poměrně časově náročné, což narušuje základní myšlenku algoritmu BCS, navíc jeho barva je velice často z hlediska spojování s dalšími elementy nepodstatná a naopak může být v mnoha případech spíše zavádějící. Z toho důvodu byl v rámci oprátoru implemntován atribut *calcImgColor*, který je implicitně nastaven na hodnotu *false*, a který určuje, zda se má barva pro obrázky vypočítávat. Pokud je výpočet barvy vypnut, tak se bude u všech obrázků brát černá barva (v RGB 0).

Výpočet vzájemné pozice boxů

Jedním z nejdůležitějších kroků předzpracování je výpočet vzájemné pozice boxů prostřednictvím relace *polo-zarovnání*. Dvojice boxů, pro které platí *rel(m, n) != other* budou dále využívány pro výpočet podobnosti, reprezentaci přímých sousedů a především jako prvotní kandidáti pro spojení do shluků (viz dále).

Ze vzorců 2 vyplývá, že relace *polo-zarovnání* je symetrická. Vztah mezi jednotlivými pozicemi z definice 3 lze vyjádřit pomocí rovnic:

$$pos(a, b) = above <=> pos(b, a) = bellow \tag{5.1}$$

$$pos(a, b) = left <=> pos(b, a) = right \tag{5.2}$$

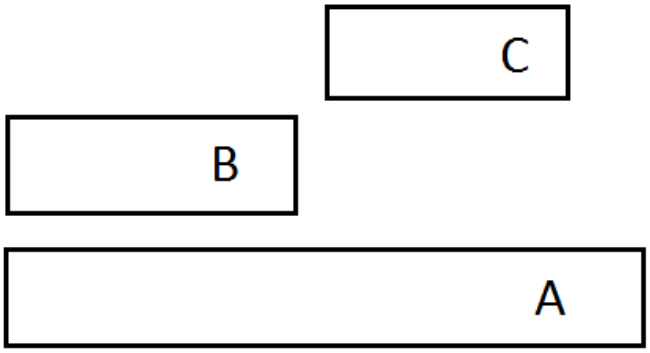
Zároveň s výpočten vzájemné pozice mezi boxy je vždy proveden výpočet absolutní vzdálenosti mezi nimi (viz 4). Tato hodnota je dále použita při počítání množiny přímých sousedů, popsaném v následující sekci a při počítání podobnosti.

Výpočet přímých sousedů

V předchozím kroku jsme kromě vzájemné pozice určili i absolutní vzdálenost mezi všemi boxy. To značí, že máme vše potřebné k určení množiny přímých sousedů pro jednotlivé boxy (viz definice 5). Určení přímých sousedů je důležité z toho důvodu, že se budou dále využíváni při určování kandidátů pro shluky.

Při určování množiny přímých sousedů je třeba počítat s tím, že na rozdíl od relace *polo-zarovnání* se nejedná o symetrický vztah.

To je ukázáno na obrázku 5.2, kde platí, že A patří do množiny přímých sousedů C, ale C není přímý soused boxu A. S tímto faktem je třeba počítat, když budeme v dalším kroku určovat kandidáty pro spojení, ale je nutné na to pamatovat i při určování přímých sousedů shluků (viz dále).



Obrázek 5.2: Příklad pro množiny přímých sousedů boxů

Určení kandidátů pro spojení a Vypočítání podobnosti

Jakmile máme pro všechny boxy určené jejich přímé sousedy, můžeme pro jednotlivé páry vypočítat hodnotu podobnosti. Ta se vypočítá pomocí vzorců z podkapitoly 3.5.1. Je programově zajištěno, že se její hodnota pohybuje v intervalu <0, 1>.

5.2.3 Hlavní smyčka

Tato sekce blíže popisuje způsob, jakým byl realizován pseudokód definovaný v 3.6. Popisujeme zde tedy princip samotného skládání elementů do shluků.

Mírně upravená verze smyčky z funkce *get_clusters()*, je popsána níže.

Před zahájením provádění smyčky je třeba připravit a nainicializovat následující parametry:

- CT - shlukový práh
- B - množina boxů nepatřících do žádného shluku
- C - množina shluků

- D - množina kandidátů na sloučení

Atribut CT je parametr operátoru, který jehož výchozí hodnota nastaven na 0,5. Celý průběh výpočtu zůstává stejný. Jeho význam je blíže popsán v kapitole 3.6 a [17].

Do množiny B na začátku patří všechny boxy identifikované během provádění sekce *Výpočet množiny CBSBoxů* (viz podkapitola 5.2.2). Během provádění smyčky budou boxy postupně odstraňovány v rámci funkce *removeBoxes()* a na konci výpočtu by zde měly být pouze ty boxy, které nemají z hlediska extrakce informace žádný význam. Množství boxů v B je pochopitelně odvozeno z nastavení parametru CT. Tedy čím je hodnota CT větší, tím je pravděpodobnější, že v množině B zůstane více položek.

Množina C obsahuje všechny potvrzené shluky. Na začátku je prázdná a postupně se dopňuje shluky vzniklými z kandidátů z množiny D. Prvky mohou být odstraňovány v rámci provádění metody *removeClusters()*.

D je množina kandidátů, která je inicializována v rámci sekce *Určení kandidátů pro spojení* (viz podkapitola 5.2.2). Její obsah se mění při vybrání nejlepšího kandidáta (po provedění *getBestCandidate()*) a během běhu funkce *recalcNgb()* (viz níže). Na konci provádění smyčky nesmí mezi kandidáty v množině D existovat takový, jehož hodnota podobnosti (vypočítaná během přezpracování v podkapitole 5.2.2) je menší nebo rovna hodnotě CT.

Nyní přejdeme k popisu průběhu samotné smyčky. Ta probíhá dokud množina D obsahuje alespoň jeden prvek nebo dokud pro všechny její prvky platí, že jejich hodnota podobnosti je vyšší než CT. Prvním krokem je nalezení nejvhodnějšího kandidáta na sloučení, o to se stará *getBestCandidate*, která jednoduše projde všechny prvky množiny D a vrátí dvojici s nejnižší hodnotou podobnosti. Dále je zkontrolováno, zda daná podobnost není příliš vysoká. To je prováděno prostřednictvím shlukového práhu (CT). V případě neúspěchu je jasné, že již není možné provést další sloučení elementů a smyčka je ukončena. Pokud je hodnota podobnosti dostatečně nízká, tak se vytvoří shlukové zrno (*seed*), o což se v našem případě stará metoda *createCandidate()*, v reálném kódu pak kostruktor třídy reprezentující shluk (*CBSCluster*).

Vzhledem k tomu, že metoda BCS neumožňuje překrývání výstupních oblastí, tak je třeba naše zrno dále zkontrolovat na překrytí se zbylými boxy a shluky. Během této kontroly je možné, že zrno bude doplněno o jeden nebo více boxů. Funkce *overlaps*, která se o kontrolu a modifikaci shlukového zrna stará, je podrobněji popsána v 5.2.4. Tato funkce vrací boolean hodnotu a v případě kdy vrátí hodnotu *true* je zrno vyhodnoceno jako nevhodné pro vytvoření shluku, diskartováno a vybere se další kandidát na sloučení. Pokud je vrácena hodnota *false*, tak je možné zrno označit za validní shluk a vložit ho do množiny C. Kromě toho je před vybráním další dvojice z množiny D provést několik zásadních věcí. Je třeba identifikovat ty boxy a shluky, ze kterých byl nový shluk složen a odstranit je z množin B, respektivě C. O to se starají funkce *removeClusters()* a *removeBoxes()*. Kromě toho je dále třeba přepočítat množinu D a aktualizovat množiny přímých sousedů jednotlivých shluků. V našem příkladu se o toto stará metoda *recalcNgb()*. Proces přepočítání D je jednou z nejsložitějších a nejzásadnějších částí našeho algoritmu a jeho průběh je blíže popsán v podkapitole 5.2.5.

Algoritmus 2

```
while (|D| > 0):
    d = getBestCandidate(D)
    D.remove(d)
    if (d.similarity > CT):
        break
    seed = createCandidate(d)
    if(overlaps(seed, C, B)):
        continue
    removeClusters(C, seed)
    removeBoxes(B, seed)
    recalcNgb(seed, B, C, D)
    C.add(seed)
```

5.2.4 Kontrola na překrytí

Na úvod této podpakitoly je dobré podotknout, že pokud není řečeno jinak, tak v rámci textu této sekce jsou pojmem box myšleny ty boxy, které v době kontroly nepatří do žádného shluku (viz definice 7).

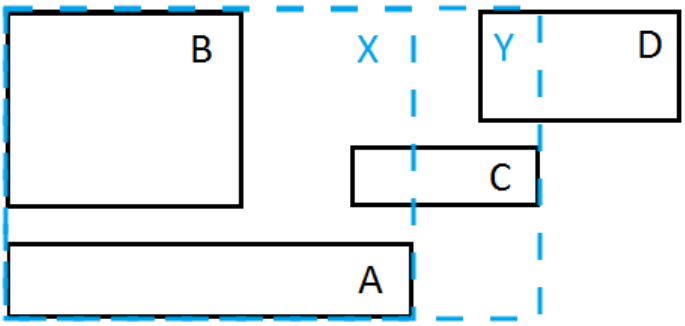
Z definice BCS vyplývá, že výsledný strom oblastí (viz obrázek 5.6) se skládá ze vzájemně se nepřekrývajících shluklů. Je tedy třeba zajistit, aby se výsledné vygenerované shluky nepřekrývaly s ostatními elementy (samostatnými boxy a ostatními shluky). Proces kontroly, která toto zajišťuje je mimo jiné realizována funkcí *overlaps()* (viz předchozí kapitola - algoritmus 2).

Zároveň je však důležité zajistit, aby kontrola na překrytí nebránila vytváření validních shluků. Tím je míněno to, že je třeba zabránit situaci, kdy shlukový kandidát překrývá box, který by bylo možné do kandidáta přidat, ale místo toho se kandidát označí jako invalidní a v dalším výpočtu se ignoruje.

Proces vytvoření shluku a kontroly na překrytí lze popsat prostřednictvím následujících kroků :

1. Vytvoření kandidáta na shluk
2. Nalezení množiny boxů N, které se překrývají s kandidátem
3. Vložení množiny N do shluku a přepočítání jeho prostorových souřadnic
4. Kontrola, zda se nový kandidát překrývá s dalšími boxy
5. Kontrola, zda se nový kandidát překrývá s ostatními shluky

Pokud během bodu 4. nebo 5. vyjde najevo, že se kandidát překrývá, tak je diskartován a vybere se nový (funkce *overlaps()* vrátí hodnotu *true*). Na obrázku 5.3 je vidět příklad, kdy metoda *overlaps()* selhala na čtvrtém kroku.



Obrázek 5.3: Příklad kdy *overlaps()* selže na čtvtém bodu

Během testování byla vyzkoušena varianta, kdy se do kandidáta postupně přidávaly boxy, dokud nedošlo k překrytí se shlukem (teprve potom byl kandidát diskartován) nebo dokud nebylo zajištěno, že se kandidát nepřekrývá s dalšími boxy. Bod 4 z výše uvedeného seznamu se tedy nepoužíval a body 2, 3 a 5 byly prováděny cyklicky. Toto řešení se však ukázalo jako nepraktické, neboť v některých specifických případech docházelo k tomu, že kandidáti na shluk obsahovali velké množství boxů, kdy obsažené boxy neměly z vizuální stránky mnoho společného (hodnota podobnosti vysoce přesahuje stanovenou mez - viz CT). Od této varianty bylo tedy odstoupeno.

Vzhledem k tomu, že se jak v případě boxů, tak shluků jedná o obdélníky, jejichž pozice známe, je samotná kontrola na překrytí dvou elementů triviální. Kontrola je realizována prostřednictví následující rovnice (hodnoty *left*, *right*, *top* a *bottom* odpovídají těm z definice 1):

$$overlaps(m,n) = \begin{cases} false, & m.left \geq n.right \vee m.right \leq n.left \\ false, & m.top \geq n.bottom \vee m.bottom \leq n.top \\ true, & jinak \end{cases} \tag{5.3}$$

5.2.5 Přepočítání sousedů

Po provedení kontroly na překrytí popsané výše je již možné považovat kandidáta za validní shluk a přidat ho do množiny C. Před zahájením dalšího cyklu smyčky je však třeba ještě provést několik zásadních operací.

Jedná se o přepočítání množiny přímých sousedů pro všechny shluky (ne jenom ten nový) a vytvoření nové množiny kandidátů D. Tato funkcionalita je pokryta v rámci metody *recalcNgb(seed, B, C, D)* z algoritmu 2 a v této kapitole se jí budeme blíže věnovat.

Je třeba pamatovat na to, že množina kandidátů D se vytváří z množiny přímých sousedů. Po provedení předzpracování (viz 5.2.2) se množina D skládá pouze z dvojic boxů odvozených z definice 5. Nicméně v průběhu výpočtu bude doplňována o dvojice skládající se ze shluků (odvozených z definice množiny přímých sousedů pro shluk - viz 7).

Navíc je potřeba počítat s tím, že množiny přímých sousedů shluků se mění v průběhu výpočtu. To je způsobeno tím, že během provádění kroků v algoritmu 2 měníme množinu shluků C (pokud se kandidát skládá z jednoho nebo dvou shluků, tak je z C odstraňujeme) a samostatných boxů B (odstraňujeme ty boxy, ze kterých byl nový shluk složen). Když se

podíváme na definici 7, tak je jasné, že množina přímých sousedů shluku se skládá jak ze shluků, tak z boxů z množiny B. Vždy, když potvrdíme vytvoření nového shluku, je tedy třeba přepočítat množiny přímých sousedů pro všechny prvky C (včetně právě vzniklého) tak, aby odpovídaly novým množinám B a C.

Následně je třeba přidat nově vzniklé dvojice do množiny kandidátů D. Kromě toho je třeba odstranit ty kandidáty na sloučení, které již nejsou validní (jeden z jejich prvků byl odstraněn z množiny C nebo B). Kandidáti, kterých se operace nedotknou (ani jeden z jejich prvků nebyl vyjmut z množiny C ani B) zůstanou kandidáty i v dalším výpočtu a je třeba je ponechat v množině D nedotčeny.

Pseudokód funkce *recalcNgb()* je možné vidět níže.

Algoritmus 3

```
function recalcNgb(seed, B, C, D):
    elems = getClusterElem(seed)
    foreach C as c:
        dC = recNgbCluster(seed,c, elems)
    foreach B as b:
        dB = recNgbBox(seed,b,elems)
    foreach D as d:
        (n, m) = getCandidateElem(d)
        if (n ∈ elems OR m ∈ elems):
            D.remove(D)
    D.add(dC)
    D.add(dB)
```

Přepočítání kandidátů se skládá ze tří cyklů. V prvních dvou jsou přepočítávány množiny přímých sousedů shluků a ve třetí se pak vyhodí z množiny D ty dvojice, které již nejsou validními kandidáty na sloučení.

Na závěr se do D přidají dvojice vypočítané v prvních dvou cyklech.

Algoritmus 4

```
function recNgbBox(seed, box, elems):
    N_b = getNgb(box)
    cBoxes = getBoxes(seed)
    foreach cBoxes as b:
        if (b in N_b AND b ∉ elems):
            return newCandidate(seed, box)
    return null
```

Algoritmus 5

```
function recNgbCluster(seed, cluster, elems):
    cEl = getClusterElem(cluster)
    foreach elems as sB:
        N_s = getNgb(sB)
        foreach CEl as cB:
            N_c = getNgb(cB)
            if (cB ∈ N_s OR sB ∈ N_c):
                return newCandidate(seed, cluster)
    return null
```

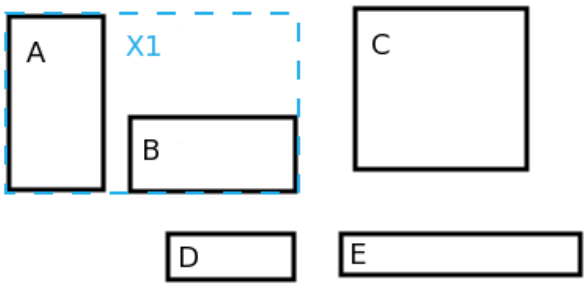
Funkce *recNgbBox()* a *recNgbBox()* se starají o přepočítání sousedů shluku. Obě vrací dvojici odvozenou z množiny přímých sousedů, která bude použita jako kandidát na sloučení v dalším výpočtu nebo prázdnou množinu (*null*) v případě, kdy se nový kandidát nevytváří. Jejich průběh odpovídá rovnicím z definic 5 a 7. V případě *recNgbBox()* je třeba počítat s tím, že je možné, aby se změnila množina přímých sousedů jak pro nový shluk(*seed*), tak pro ten kontrolovaný (*cluster*), a protože to, že shluk A nepatří do přímé množiny shluku B neznamená, že B není přímým sousedem pro A (relace není symetrická - viz výše), je třeba provést kontrolu oboustranně.

Pro větší názornost přepočítání množin přímých sousedů a množiny kandidátů D nyní uvedeme jednoduchý příklad. Předpokládejme situaci, kdy struktura stránky odpovídá obrázku 5.4 a jednotlivé množiny mají následující hodnoty (množiny, které jsou označené písmenem N, jsou množiny přímých sousedů daných prvků):

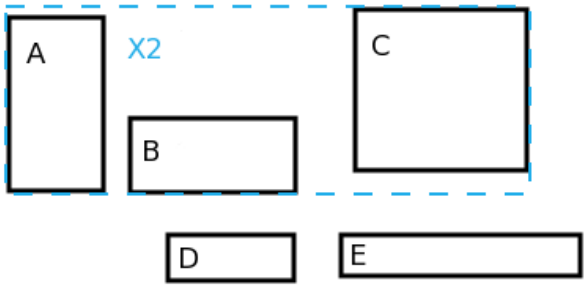
- $B = \{C, D, E\}$
- $C = \{X1\}$
- $D = \{ (X1, C), (X1, D), (C, E), (D, E) \}$
- $N_{X1} = \{C, D\}$, $N_A = \{B\}$, $N_B = \{A, C, D\}$, $N_C = \{B, E\}$, $N_C = \{B, E\}$, $N_D = \{B, E\}$, $N_E = \{C, D\}$

Předpokládejme, že v dalším cyklu smyčky dojde ke sloučení shluku *X1* a boxu *C*. Po provedení funkce *recalcNgb()* a upravení množin B a C bude situace následující:

- $B = \{D, E\}$
- $C = \{X2\}$
- $D = \{(X2, D), (X2, E), (D, E) \}$
- $N_{X2} = \{D, E\}$, $N_A = \{B\}$, $N_B = \{A, C, D\}$, $N_C = \{B, E\}$, $N_C = \{B, E\}$, $N_D = \{B, E\}$, $N_E = \{C, D\}$



Obrázek 5.4: Před sloučením

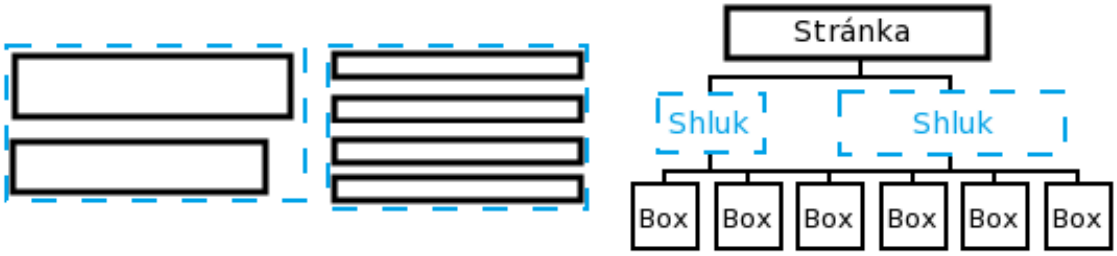


Obrázek 5.5: Po sloučení *X1* a *B*

5.2.6 Výstup

Po dokončení smyčky popsané v 5.2.3, je třeba provést převedení naší množiny shluků do takové podoby, která je očekávána nástrojem FITLayout. To je nutné z toho důvodu, aby mohlo být provedeno zobrazení námi nalezených oblastí. V praxi je tedy třeba vytvořit kolekci objektů implementující třídu *AreaImpl*.

Výstupem operátoru *CBSegm* je tedy strom oblastí o třech úrovních. Příklad struktury je možné vidět na obrázku 5.6. Na tento strom je možné aplikovat další operátory implementující rozhraní *BaseOperator* a dále ho tak upravovat. Nicméně výstup našeho operátoru lze již přímo využít v další fázi zpracování webového dokumentu, tedy extrakce informací.



Obrázek 5.6: Ukázka výstupní struktury BCS

Z výše popsaného kódu vyplývá, že na konci provádění hlavní smyčky (viz 5.2.3) může nastat situace, kdy v množině B budou stále nějaké prvky. Jedná se o boxy, které se nepodařilo zařadit do žádného shluku. Z našeho pohledu jsou tyto prvky stránky nepodstatné pro extrakci informací a jsou proto při dalším zpracovávání a zobrazování zcela ignorovány.

Kapitola 6

Testování a dosažené výsledky

Tato kapitola pojednává o závěrečné fázi projektu, tedy testování. Je zde představen způsob jakým byla kontrolována validnost finálního produktu, jakožto i dosažené výsledky v podobě obrázků obsaujících výstup naší segmentační metody prostředictvím nástroje FITLayout.

V první části (6.1) se věnujeme formě testovaní, tedy jak přesně probíhalo a na jakých vzorcích (v našem případě webových stránkách) se uskutečnilo. V podkapitole 6.2 se podíváme na dosažené výsledky, probereme případy, kdy segmentace neproběhla ideálně a důvody, které to způsobily. Dále se blíže podíváme na parametry operátoru a jejich vliv na úspěšnou segmentaci. Po celkovém zhodnocení ještě následuje porovnání s ostatními segmentačními metodami (sekce 6.4).

Při testování segmentačních metod narážíme na problém zvolení vhodných metrik. Nejjednodušeji měřitelná je rychlost výpočtu, nicméně jak bylo zmíněno výše, nás tato metrika nezajímá a během implementace nebyl na rychlost algoritmu kladen důraz, měření rychlosti tedy není součástí testování.

Přesto je nutné zmínit, že samotná metoda BCS byla navržena za účelem dosažení porovnatelných výsledků k metodě VIPS za podstatně kratší dobu. Porovnání těchto dvou metod proběhlo v rámci práce [17], kde bylo ukázáno, že z hlediska rychlosti je metoda BCS skutečně lepší.

Při našem testování jsme se tedy soustředili pouze na kvalitu a stabilitu výsledků.

Kontrolou stability je myšleno to, že jsme chtěli, aby náš projekt dosahoval stejných výsledků na stránkách se stejnou strukturou. Například bylo žádoucí, aby články na jednom zpravodajském portálu byly segementovány obdobným způsobem.

Co se týče kvality výstupů, tak tady jsme pro kontrolu zvolili především metodu porovnání s dalšímy segmentačními metodami, jakožto i vlastní úsudek. Z hlediska kvality výstupů bylo důležite, aby v rámci segmentace byly správně identifikovány významově odlišné elementy stránky, jako hlavička, patička, navigace a hlavní obsah stránky.

6.1 Průběh testování

Celý proces testování je možné rozdělit do čtyř fází:

1. testování na vlastních stránkách
2. testování na reálných stránkách
3. porovnání s ostatními metodamy ve FITlayoutu

4. porovnání s implementací z [17]

Během první fáze (testování na vlatních stránkách) byl projekt zkoušen na stránkách vytvořených speciálně pro jeho testování. Tato fáze probíhala z velké části během vývoje a příklady, které byly použity v kapitole 5, jsou převzaty právě z této fáze.

Testování na reálných stránkách bylo bráno jako hlavní rozhodující faktor pro určení, zda byl projekt úspěšný, a z tohoto důvodu zabralo většinu času a proběhlo na největším počtu vzorků. Pro kontrolu validity proběhlo na vybraných vzorcích porovnání s ostatními segmentačními metodami a s implementací algoritmu BCS z projektu *ToolsSegmentation*.

V rámci jednoho cyklu testování proběhly následující kroky

1. Vybrání webové stránky
2. Vykreslení stránky (pomocí CSSBox)
3. Několikrát aplikuj operátory *FlattenTreeOperator* a *CBSegm* s různými parametry
4. Kontrola výstupu segmentace
5. Porovnej s dalšimi metodamy

6.1.1 Výběr vzorků

Na internetu je možné nalézt velké množství různých typů stránek. Pro nás je především důležité je rozdělit do skupin z hledika struktury a zajistit tím, že otestujeme jak se náš projekt chová na všech možných typech stránek.

Pro účely této práce bylo převzato rozdělění stránek do tří skupin z [17]. Jedná se o:

- Jednoduché stránky

  Jednoduché stránky, které se kromě hlavního obsahu zkládají jen z malého počtu dodatečných elementů. Hlavní obsah není roztroušen po celé stránce, ale nachází se na jednom místě, kde je již od prvního pohledu snadno identifikovatelný. Jedná se například o osobní blogy nebo vzdělávací stránky.

- Články

  Stránky obsahující jeden hlavní blok textu, který se obvykle skládá z velkého množství paragrafů, doplněný o příležitostné obrázky. Kromě toho se zde nachází celá řada dalších elementů jako navigace, reklamy, patička s hlavičkou a bloky obsahující reference na další články obdobné struktury.

- Seznamy

  Z hlediska doprovodných elementů mohou být podobné článkům, ale nelze na nich nalézt souvislý blok textu, který by šlo označit za hlavní obsah. Místo toho se skládají z několika oblastí složených jen z pár elementů. Jedná se o titulní stránky informačních portálů nebo například stránky zobrazující seznamy na internetových obchodech.

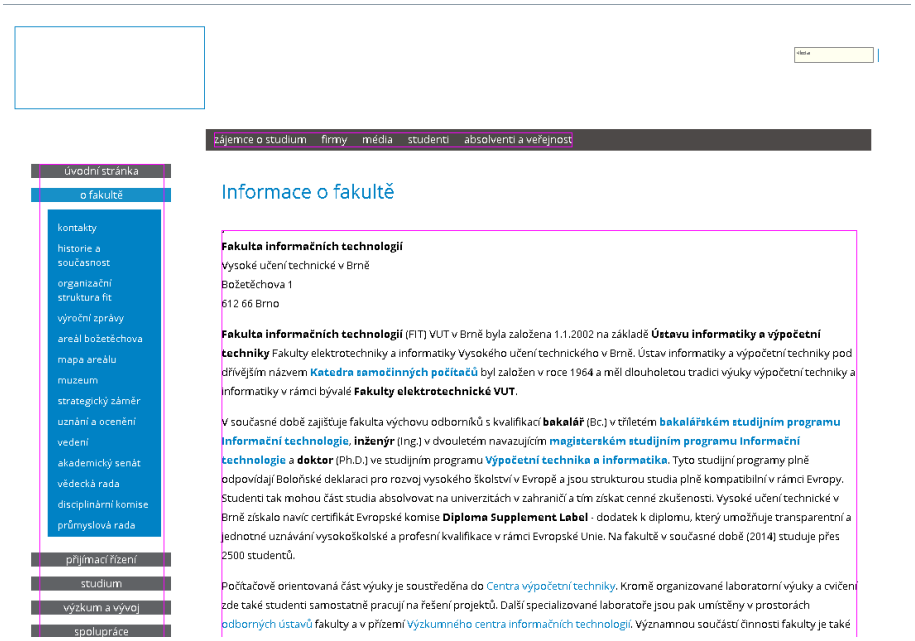
Během testování byly brány vzorky ze všech tří kategorií. Následuje konkrétní seznam stránek, na kterých byla metoda BCS odzkoušena:

www.idnes.cz
www.aktualne.cz

www.novinky.cz
www.fit.vutbr.cz
www.vutbr.cz
www.seznam.cz
www.centrum.cz
www.yahoo.com
www.wikipedia.org

6.2 Dosažené výsledky

Co se týče jednoduchých stránek (*www.vutbr.cz*, *www.fit.vutbr.cz*, ...), tak při správném nastavení vstupního parametru CT (viz níže) byly ve všech případech stránky rozděleny podle požadovaných kritérií. Prvky jako navigace a záhlaví byly správně odděleny od hlavního obsahu, který byl vždy obsažen v jednom nebo více blocích. Příklad úspěšné segmentace stránky *www.fit.vutbr.cz* je možné vidět na obrázku 6.1.



Obrázek 6.1: Výsledek segmentace stránky *www.fit.vutbr.cz*

Je zde vidět, že dokument byl správně rozdělen do tří částí (hlavní navigace, vedlejší navigace a hlavní obsah). Obdobně byly segmentovány i další stránky s obdobnou strukturou. Pouze v případech, kdy byl hlavní obsah příliš velký, docházelo k tomu, že byl rozdělen do více než jedné oblasti, ale navigace, hlavička i patička byly vždy úspěšně identifikovány.

O něco horších výsledků bylo dosaženo v případě článků a to především v případě portálu *www.idnes.cz*, kde jsou články doplněny velkým počtem obrázků, postraních panelů a reklam.

Odstavce článku byly většinou identifikovány správně, ale v některých případech (způsobených nevhodnou volbou CT) docházelo k tomu, že řádky textu byly spojovány s prvky z postranních elementů. Tento jev byl způsoben přílišnou délkou textu, která způsobila že hodnota *maxd* z definice 6 měla tak vysoká, že mezi řádkem textu a prvkem postranního

panelu byla nízká hodnota relativní vzdálenosti (viz definice 6). Tento jev způsobil zvýšení pravděpodobnosti pro jejich sloučení. Nicméně při správném nastavení parametru CT byly postranní prvky správně identifikovány a odděleny od hlavního obsahu. Problém popsaný výše se tedy vyskytoval pouze v ojedinělých případech.

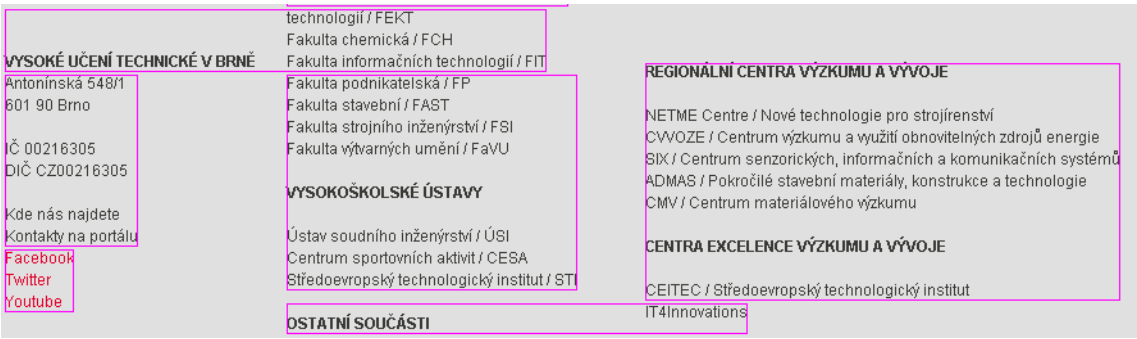
Obrázek 6.2: Výsledek segmentace stránky *www.yahoo.cz*

V [17] bylo zmíněno, že v případě stránek s komplexnější strukturou mívá algoritmus BCS problémy. Během testování byla na nich většina doprovodných elementů, jako navigace, reklamy, hlavička a patička, správně odděleny, ale algoritmus měl občas problémy korektně identifikovat hlavní obsah, který mísil s prvky vedlejších panelů. Přesto lze říci, že i v případě titulních stránek dosahovala naše implementace rozumných výsledků, což je vidět i na obrázku 6.2, kde byly správně sloučeny související odkazy na články.

Barva elementu je jedním ze tří atributů použitých při počítání základní podobnosti (viz kapitola 3.3). Nicméně během testování bylo zjištěno, že barva elementů může být v některých případech značně zavádějící. Problém s určením a vlivu barvy obrázků byl již rozebírán v sekci 5.2.2. Nicméně barva může být zavádějící i v případě textu. Někdy pomáhá identifikovat speciální prvky (jako navigace na obrázku 6.1), ale v některých situacích může dojít k tomu, že prvek nebyl přidělen do správného shluku na základě barvy. Jedná se například o odkazy, jejichž zabarvení může způsobit nevítané odtržení od hlavního obsahu. V rámci testování bylo zjištěno, že v případě některých stránek (např. *wikipedia.org*) byly výsledky segmentace lepší při ignorování vlivu barvy.

V kapitole 3 jsme zmínili, že jednou ze zásadních částí metody BCS je kontrola na překryv. V rámci sekce 5.2.4 bylo zmíněno, že je nutné zabránit tomu, aby metoda *overlaps()* (viz výše) nezabránila vytváření valdních shluků. Extrémní příklad je možné vidět na obrázku 6.3, kde sloučení oblastí brání fakt, že by se výsledný shluk překrýval s již existujícím. Došlo tedy k uváznutí segmentace. V průběhu testování jsme narazili na některé případy, kdy zákaz překryvu shluků způsobil, že výstup segmentace byl zkreslen a neodpovídal skutečnosti, ale většinou se jednalo o obsah vedleších prvků.

Stabilita byla testována především na článcích portálů *idnes.cz*, *aktualne.* cz a *yahoo.com*. Na každém z těchto portálů bylo vyzkoušno deset článků, jež byly segmentovány se stejnou hodnotou parametru CT. Výstupy byly poté v rámci portálů porovnány. Po dokončení porovnánvání lze prohlásit, že naše implementace dosahuje stabilních výsledků.

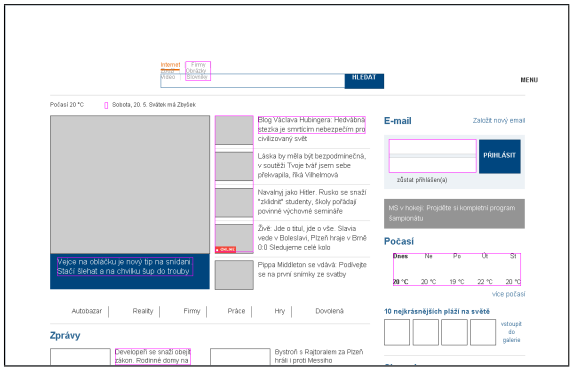


Obrázek 6.3: Důsledky špatného přístupu ke kontrole na překryv

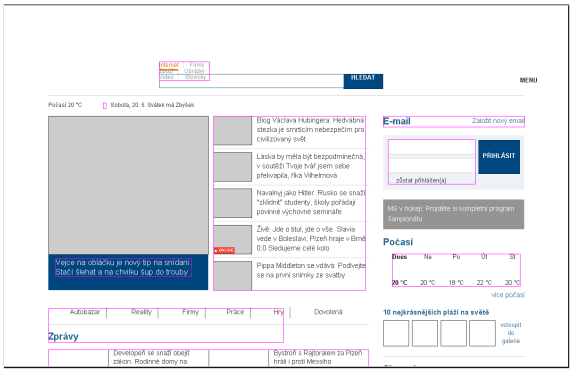
6.3 Vliv parametrů

V této části se podíváme na vliv parametrů *CT* a *calcImgColor* operátoru *CBSegm* na výsledek procesu segmentace.

Výše bylo zmíněno, že vstupní vzorky byly testovány s rozdílnými parametry. Toto se týká především parametru *CT*, pro který byly stránky testovány s hodnotami od 0.1 až po 1 (s krokem 0.1). V praxi testování jedné stránky tedy proběhlo nejméně desetkrát. Na obrázcích 6.4 až 6.7 jsou vidět výsledky segmentace jedné stránky pro čtyři rozdílné hodnoty *CT*. Je zřejmé, že tento parametr přímo ovlivňuje zrnitost výsledku, a že jeho správná hodnota je velice důležitá pro korektní segmentaci.



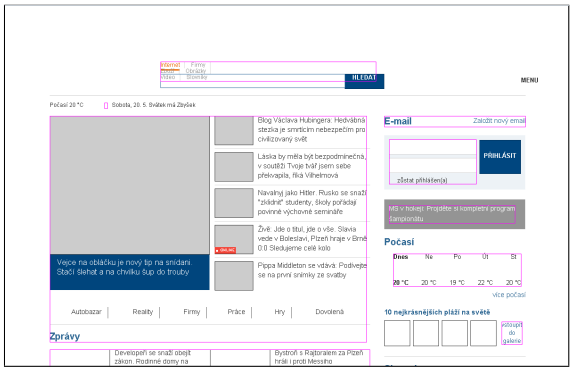
Obrázek 6.4: $CT = 0.1$



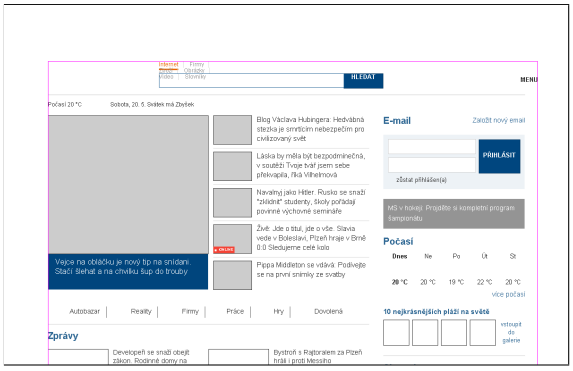
Obrázek 6.5: $CT = 0.2$

Během testování bylo potvrzeno, že pro odlišné stránky je vhodná různá hodnota parametru. To je z velké části způsobeno tím, že základní podobnost je počítána pomocí relativní vzdálenosti (viz definice 6). V případě velkých stránek dochází k tomu, že i nízké rozdíly v parametru *CT* mohou mít velký dopad na výsledek. Pokud ho vybereme příliš nízký, dojde k tomu, že některé důležité boxy nebudou zahrnuty do žádného shluku a budou tedy ignorovány. Při příliš vysoké hodnotě *CT* se pak stává, že hlavní obsah je mísen s prvky neobsahujícími důležité informace (viz obrázek 6.7).

Důvod zavedení parametru *calcImgColor* byl vysvětlen v kapitole 5.2.2. V průběhu testování jsme se snažili odhalit, zda ignorování barvy obrázku má za následek horší výsledky segmentace. Po dokončení testování lze prohlásit, že veškeré dosažené výsledky popsané v této kapitole byly dosaženy i když nebyla barva obrázků brána vpotaz, i z tohoto důvodu



Obrázek 6.6: $CT = 0.3$



Obrázek 6.7: $CT = 0.5$

je operátor *CBSegm* implicitně nastaven na hodnotu *false*.

6.4 Porovnání s dalšími segmentačními metodami

Hlavním cílem naší práce bylo vytvořit variantu pro segmentaci, která odpovídá algoritmu popsaném v článku [17], v rámci něhož byl vytvořen projekt *ToolsSegmentation*, který poskytuje implementaci metody BCS. Veškeré stránky, zmíněné v podkapitole 6.1, byly tedy porovnány s výsledky vygenerovanými tímto projektem.

Příklad porovnání výstupů projektu *ToolsSegmentation* a našeho projektu *CBSSegmentation* je možné vidět na obrázcích 6.8 a 6.9. Je zde zpracovávána stránka s hodnotou CT 0.2.

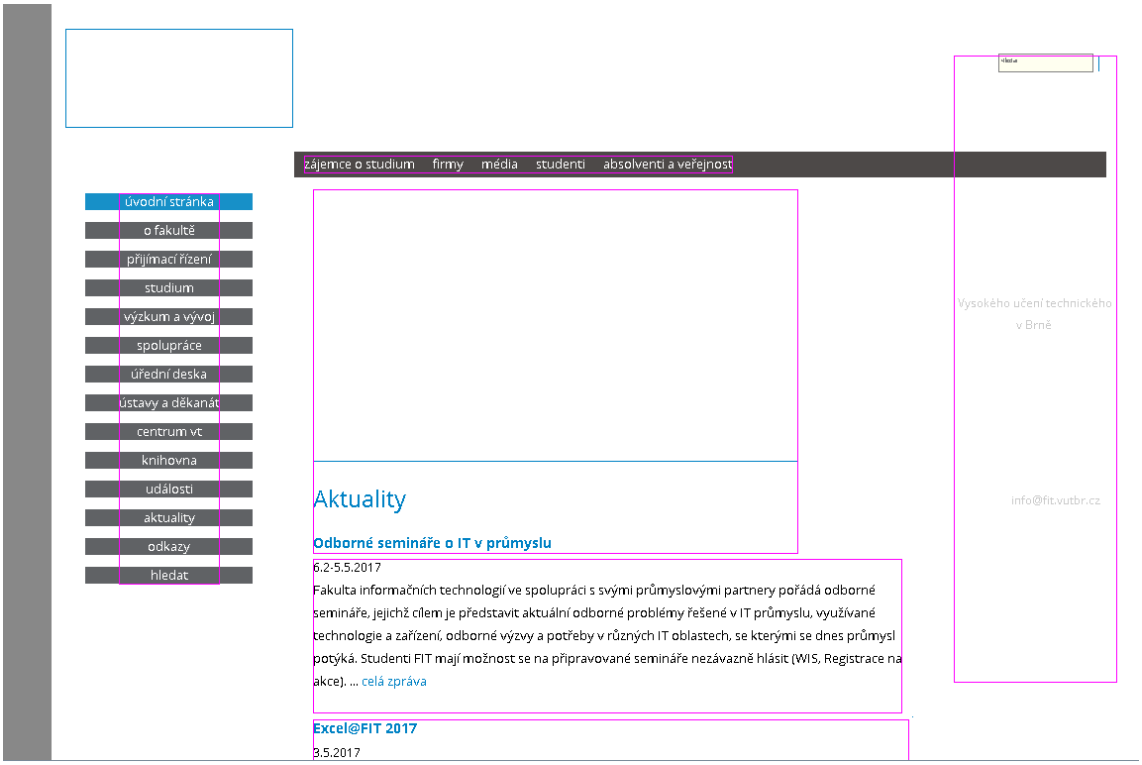
Z obrázků 6.8 a 6.9 je patrné, že výsledky jsou podobné, ale drobně se liší v pozicích a tvarech nalezených shluků. To je způsobeno tím, že porovnávané projekty používají jiné vykreslovací nástroje. Implementace v práci [17] například v podstatě nepracuje s obrázky na stránkách a často je úplně ignoruje. Dalším zdrojem zkreslení je pak operátor *FlattenTreeOperator*, který předzpracování (viz sekce 5.2.2) implementuje jiným způsobem než projekt *ToolsSegmentation*. Výstupní seznamy shluků se tedy na testovaných stránkách mírně lišily, ale z hlediska identifikace hlavního obsahu a nalezení doprovodných prvků dosahovaly stejných výsledků.

Rozdílný způsob vykreslovaní měl ještě za následek to, že pro dosažení stejných výsledků bylo v některých případech třeba zadat rozdílné hodnoty CT. Nicméně odchylka nikdy nepřerostla hodnotu 0.1 (CT se pohybuje v intervalu <0, 1>).

Co se týče ostatních segmentačních metod, tak vzhledem k nedostatečné přístupnosti nástrojů pro segmentaci webových dokumentů, jsme v rámci testování měli k dispozici pouze metody implementované v nástroji FITLayout. Pro porovnání jsme primárně použili metodu, která je ve FITLayout nastavena jako výchozí (viz [5]). Ta se skládá z následujících operátorů:

1. FindLines
2. HomogeneousLeaves
3. SuperAreas

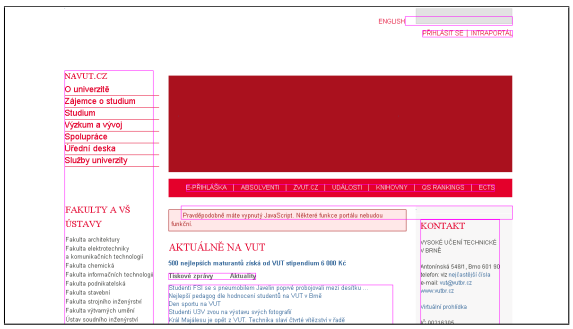
Na první pohled je zřejmý rozdíl mezi metodou BCS a tradičními přístupy k segmentaci. Zatímco klasické algoritmy stránku reprezentují stromem oblastí, výstup naší metody



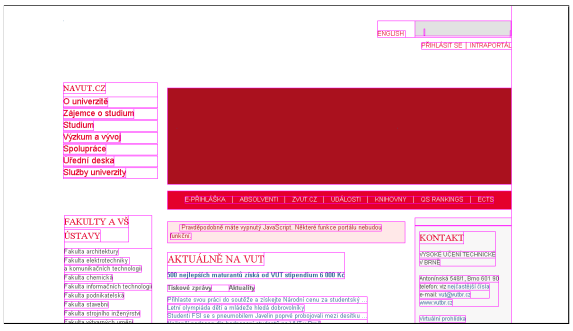
Obrázek 6.8: Segmentace pomocí *CBSSegmentation*

je množina shluků. Respektove strom o dvouch úrovních, kde kořen je celá stránka. Přesto je vidět, že námi nalezená množina do značné míry odpovídá oblastem na nejvyšší úrovni stromu oblastí, získaném z implicitní metody používané FITLayoutem. Pro porovnání algoritmu BCS s metodou VIPS je možné se blíže informovat v článku [17].

Obrázek 6.9: Segmentace pomocí *ToolsSegmentation*



Obrázek 6.10: Segmentace pomocí *CBSSegmentation*



Obrázek 6.11: Segmentace pomocí implicitní metody ve FITLayoutu

Kapitola 7

Závěr

V první části práce jsme se zabývali segmentací z teoretické stránky. Popsali jsme její obecný průběh a její význam při extrakci informací. Dále jsme pro porovnání uveli několik tříd segmentačních metod a jejich nejznámější zástupce.

Kapitoly 3 a 4 pak uvedly potřebné informace ke splnění hlavního cíle práce. V kapitole 3 jsme se především věnovali algoritmu Box Clustering Segmentation. Hlavním a v podstatě jediným zdrojem pro tuto kapitolu byl článek [17].

Vzorce a definice z kapitoly 3 byly následně implementovány pomocí jazyku Java v projektu *CBSegmentation*, jemuž se podrobně věnuje sekce 5. Kromě převodu vzorců je zde detailně popsáno, jakým způsobem jsme přistoupili k problémům kontroly na překrytí a přepočtu množiny přímých sousedů.

Otestování výsledného produktu je popsáno v kapitole 6. Primárním cílem testování bylo ověřit, že naše implementace metody BCS odpovídá implementaci z projektu *ToolsSegmentation*. To se podařilo a projekt *CBSegmentation* lze tedy považovat za úspěšný.

Co se týče zhodnocení samotného algoritmu BCS, tak je dobré zmínit, že jednoduchost a univerzálnost algoritmu BCS znamená, že je možné ho použít v podstatě na jakýkoliv dokument za účelem extrakce informací. Jednou z jeho hlavních charakteristik je, že počítá pouze se základními atributy elementů (vzdálenost, tvar, barva). To má své výhody, ale i nevýhody. Hlavní nevýhoda spočívá v tom, že v některých případech mohou mít barva a tvar elementů negativní vliv na výsledek segmentace. Dalším potenciálním problémem je způsob výpočtu relativní vzdálenosti (tomuto jsme se věnovali v podkapitole 6.2).

Metoda BCS je stále ve vývoji a je zřejmé, že počítání základní podobnosti je jednou z oblastí, jež by ještě potřebovala vylepšit. Jedna možnost, která byla testována v rámci této práce, je zavedení parametrů, které určují vliv vzdálenosti, tvaru a barvy na vypočítanou hodnotu podobnosti. Zvláště v případě barvy bylo na některých typech stránek dosaženo uspokojivějších výsledků.

Git repozitář obsahující projekt *CBSegmentation* lze nalézt na adrese *https://github.com/xlenga01/BCSSegmentation* a po provedení kroků popsaných v podkapitole 5.1.2. lze operátor *CBSegm* použít pro segmentaci libovolné webové stránky.

S ohledem k tomu, lze prohlásit, že hlavní cíl této práce, tedy zajištění nové, veřejně dostupné varianty pro segmentaci v rámci nástroje FITLayout, byl splněn.

Literatura

[1] Bernanrd, M.: Criteria for optimal web design (designing for usability). 2002.

[2] Burget, R.: Layout Based Information Extraction from HTML Documents. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, ročník 2, Sept 2007, ISSN 1520-5363, s. 624–628, doi:10.1109/ICDAR.2007.4376990.

[3] Burget, R.: CSSBox. http://cssbox.sourceforge.net/, 2015.

[4] Burget, R.: FITLayout. http://www.fit.vutbr.cz/~burgetr/FITLayout, 2015.

[5] Burget, R.: *FITLayout Framework Manual*. 2015.

[6] Burget, R.: FITLayout Javadoc. http://www.fit.vutbr.cz/~burgetr/FITLayout/api, 2015.

[7] Burget, R.; Rudolfova, I.: Web Page Element Classification Based on Visual Features. In *2009 First Asian Conference on Intelligent Information and Database Systems*, April 2009, s. 67–72, doi:10.1109/ACIIDS.2009.71.

[8] Cai, D.; Yu, S.; Wen, J.-R.; aj.: VIPS: a Vision-based Page Segmentation Algorithm. Technická zpráva, Microsoft Corporation.

[9] Hong, J. L.; Siew, E.-G.; Egerton, S.: Information extraction for search engines using fast heuristic techniques. 2010.

[10] Laščák, T.: *Algoritmy pro segmentaci webových stránek*. Diplomová práce, Vysoké učení technické v Brně, 2016.

[11] Lin, S.-H.; Ho, J.-M.: Discovering Informative Content Blocks from Web Documents. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, New York, NY, USA: ACM, 2002, ISBN 1-58113-567-X, s. 588–593, doi:10.1145/775047.775134.
URL http://doi.acm.org/10.1145/775047.775134

[12] Malaník, M.: *Nové metody segmentace webových stránek*. Diplomová práce, Vysoké učení technické v Brně, 2016.

[13] Milička, M.; Burget, R.: Web document description based on ontologies. In *2013 Second International Conference on Informatics Applications (ICIA)*, Sept 2013, s. 288–293, doi:10.1109/ICoIA.2013.6650271.

[14] Popeta, T.: *Implementace algoritmu pro vizuální segmentaci www stránek*. Diplomová práce, Vysoké učení technické v Brně, 2012.

[15] Zeleny, J.; Burget, R.: Cluster-based Page Segmentation-a Fast and Precise Method for Web Page Pre-processing. In *Proceedings of the 3rd International Conference on Web Intelligence, Mining and Semantics*, WIMS '13, New York, NY, USA: ACM, 2013, ISBN 978-1-4503-1850-1, s. 7:1–7:12, doi:10.1145/2479787.2479792.
URL http://doi.acm.org/10.1145/2479787.2479792

[16] Zelený, J.: Web page segmentation and classification. 2011.

[17] Zelený, J.; Burget, R.: Box Clustering Segmentation, a new method for vision-based page segmentation.

[18] ZENG, J.; FLANAGAN, B.; HIROKAWA, S.; aj.: A Web Page Segmentation Approach Using Visual Semantics. *IEICE Transactions on Information and Systems*, ročník E97.D, č. 2, 2014: s. 223–230, doi:10.1587/transinf.E97.D.223.

Příloha A

Obsah CD

- Složka CBSSegmentation obsahující zdrojový kód finálního projektu
- Soubor readme.txt obsahující návod pro zprovoznění projektu
- Elektronická podoba diplomové práce