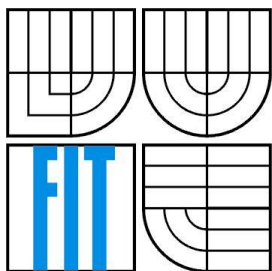


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ALGORITMY PRO SEGMENTACI WEBOVÝCH STRÁNEK

WEB PAGE SEGMENTATION ALGORITHMS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. Tomáš Laščák

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. Radek Burget, Ph.D.

Abstrakt

Segmentace webových stránek je jednou z disciplín extrakce informací. Umožňuje dělit stránky na různé sémantické bloky. Diplomová práce se zabývá seznámením se samotnou segmentací a také implementací konkrétní segmentační metody. V práci jsou popsány různé příklady metod jako je VIPS, DOM PS atd. Je zde teoretický popis zvolené metody a taktéž Frameworku FITLayout, který bude o tuto metodu rozšířen. Dále je tu podrobněji popsána implementace zvolené metody. Popis implementace je zaměřen především na popis různých problémů, které jsme museli vyřešit. Nechybí zde ani testování, které pomohlo odhalit některé nedostatky. V závěru se nachází shrnutí výsledků a možné nápady, jak by se dalo navázat na tuto práci.

Abstract

Segmentation of web pages is one of the disciplines of information extraction. It allows to divide the page into different semantic blocks. This thesis deals with the segmentation as such and also with the implementation of the segmentation method. In this paper, we describe various examples of methods such as VIPS, DOM PS etc. There is a theoretical description of the chosen method and also the FITLayout Framework, which will be extended by this method. The implementation of the chosen method is also described in detail. The implementation description is focused on describing the different problems we had to solve. We also describe the testing that helped to reveal some weaknesses. The conclusion is a summary of the results and possible ideas for extending this work.

Klíčová slova

Vizuální sémantika, Java, WWW, Segmentace, Document Object Model, Framework FITLayout

Keywords

Visual semantic, Java, WWW, Segmentation, Document Object Model, Framework FITLayout

Citace

LAŠČÁK, Tomáš. *Algoritmy pro segmentaci webových stránek*. Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Burget Radek.

Algoritmy pro segmentaci webových stránek

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Laščák

19. 5. 2016

Poděkování

Rád bych tímto poděkoval vedoucímu této práce, Ing. Radku Burgetovi, Ph.D., za odborné rady a vedení při tvorbě této práce.

© Tomáš Laščák, 2016

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

| | | |
|-------|--|----|
| 1 | Úvod..... | 2 |
| 2 | Segmentace webových stránek | 4 |
| 2.1 | DOM-based Page Segmentation | 4 |
| 2.2 | Vision-based Page Segmentation – VIPS | 6 |
| 2.3 | Box Clustering Segmentation..... | 8 |
| 3 | Segmentace webových stránek za pomoci vizuální sémantiky | 10 |
| 3.1 | Příprava | 10 |
| 3.1.1 | Vizuální bloky..... | 11 |
| 3.1.2 | Předzpracování webových stránek..... | 11 |
| 3.2 | Algoritmus segmentace | 12 |
| 3.2.1 | Rozpoznání podobných vizuálních bloků..... | 12 |
| 3.2.2 | Výpočet stupně sousednosti bloků..... | 14 |
| 3.2.3 | Výpočet podobnosti obsahu bloků..... | 15 |
| 3.2.4 | Segment webové stránky | 18 |
| 4 | FITLayout | 20 |
| 4.1 | FITLayout Moduly..... | 20 |
| 4.1.1 | FITLayout API..... | 20 |
| 4.1.2 | CSSBox..... | 21 |
| 4.1.3 | Segmentace | 22 |
| 4.1.4 | Klasifikace | 23 |
| 4.1.5 | Služby | 23 |
| 4.1.6 | Nástroje | 23 |
| 4.2 | Práce s Frameworkem | 24 |
| 4.2.1 | Operátory | 25 |
| 5 | Implementace algoritmu | 27 |
| 5.1 | Sousednost bloků..... | 27 |
| 5.1.1 | Nesousedící bloky..... | 27 |
| 5.1.2 | Sousedící bloky..... | 30 |
| 5.2 | Vizuální obsah bloků..... | 33 |
| 5.2.1 | Vektory obsahu | 34 |
| 5.2.2 | Podobnost obsahu | 35 |
| 5.3 | Řízení algoritmu..... | 35 |
| 5.4 | Vstup a výstup algoritmu | 36 |
| 6 | Dosažené výsledky..... | 39 |
| 6.1 | Testování | 39 |
| 6.2 | Změna vstupních parametrů | 41 |
| 7 | Závěr | 45 |
| | Literatura..... | 46 |
| | Seznam příloh | 48 |

1 Úvod

Mnoho uživatelů internetu o ní neví, že existuje, ale využívají ji prakticky každý den. Řeč je o segmentaci webových stránek. Na webových stránkách se nachází mnoho irelevantního obsahu, který běžný uživatel obvykle nepoužívá, a je tak potřeba tento obsah filtrovat. Použití segmentace může být v některých případech velmi užitečné. Například by ji mohli využít webové vyhledávače, pokud by se snížila její výpočetní náročnost. Velkým přínosem segmentace je hlavně filtrace nevyžádaného obsahu na webové stránce, kdy nám umožňuje získat podstatné a nejdůležitější informace z této stránky.

Dřívější techniky segmentace webových stránek jsou založeny především na algoritmech strojového učení a na algoritmech využívající různá heuristická pravidla. Avšak vzhledem k malému rozsahu testovacích množin dat, metody strojového učení mohou být aplikovány pouze v některých určitých oblastech webové stránky. Zatímco metody založené na heuristických pravidlech mohou fungovat na malých sadách stránek, nemohou být využitelné na větších sadách stránek.

V této práci implementujeme jednu z metod segmentace webových stránek. Předpokládáme, že se webová stránka skládá z konečného počtu bloků, a může být segmentována pomocí vizuálních vlastností těchto bloků. Metoda využívá těchto vizuálních vlastností. Metoda bude implementována v rámci Frameworku FITLayout a bude do něj integrována.

Co se týče obsahu práce, ve druhé kapitole si popíšeme příklady tří metod pro segmentaci. Nejprve se stručně seznámíme s metodou DOM-based Page Segmentation, která realizuje segmentaci webové stránky na základě jejího DOM stromu. Dále se poté seznámíme s jednou z nejznámějších metod pro segmentaci. Jedná se o metodu Vision-based Page Segmentation (VIPS). VIPS pro segmentaci využívá vizuálních vlastností prvků na webové stránce. Na závěr této kapitoly si řekneme ještě o metodě Box Clustering Page Segmentation.

Ve třetí kapitole se zaměříme na metodu pro segmentaci webových stránek, kterou se diplomová práce zabývá. Jedná se o metodu využívající vizuálních vlastností stránek. Zde se soustředíme především na teoretickou část této metody. Ukážeme si, jaké se zde využívají vzorce pro výpočty různých vlastností, a také v neposlední řadě pseudokód samotného algoritmu.

Čtvrtá kapitola se soustředí na seznámení se s Frameworkem FITLayout. Metoda, která bude implementovat segmentaci webových stránek, importujeme do zmíněného Frameworku. Taktéž bude využívat jeho již hotové části a je zapotřebí se s některými nejprve obeznámit. Je vhodné, abychom si také ukázali, ovládání grafického uživatelského rozhraní Frameworku.

V páté kapitole se dostaneme v podstatě k tomu nejdůležitějšímu, co se v této práci můžeme dočíst. Řekneme si všechny detaily implementace našeho segmentačního algoritmu.

Nacházejí se zde rozebrána různá úskalí, se kterými se můžeme při implementaci setkat, a taktéž jakým způsobem je vyřešit. Většina takových informací je řečena prostřednictvím obrázků, protože se jedná o nejrychlejší způsob vysvětlení.

Dostáváme se na předposlední kapitolu, jež rozebírá dosažené výsledky v diplomové práci. Jedná se o různá testování na konkrétní sadě webových stránek. Nalezneme zde několik ukázek výsledků segmentace a srovnání s výchozí metodou nacházející se ve Frameworku. Poněvadž metoda může být ovlivněna několika parametry, ukážeme si, o jaké se jedná a jakým způsobem ovlivní výsledek segmentace.

Na závěr si shrneme dosažené výsledky a splnění cílů, které diplomová práce měla. Také se zamyslíme nad možnostmi rozšíření práce, a jakým způsobem by se dalo na práci navázat.

2 Segmentace webových stránek

Segmentace webových stránek umožňuje dělení stránky podle určitých pravidel nebo vizuálních vlastností. Stránka je rozdělena na několik sémanticky rozdílných bloků a jejich obsah můžeme dále zkoumat nebo různě filtrovat podle potřeby.

Jak jsme již řekli, segmentace je slibná metoda, která by mohla být využívána u webových vyhledávačů. Slouží ke zpřesnění nalezených výsledků pro vyhledávaný řetězec a dokáže taktéž identifikovat jeho polohu v rámci webové stránky, na které byl nalezen. Můžeme tak rozpoznat a vynechat výskyty vyhledávaného řetězce. Náznými příklady použití můžeme uvést například v komentářích pod článkem nebo v odkazech sloužících k navigaci po stránce.

Existuje mnoho metod pro segmentaci webových stránek a v následujících podkapitolách si stručně charakterizujeme příklady tří z nich. Metoda, kterou se zabývá tato práce, bude popsána v samostatné kapitole.

2.1 DOM-based Page Segmentation

Jednou z metod segmentace je metoda DOM-based Page Segmentation (dále DomPS) [12]. Ta realizuje segmentaci webové stránky na základě jejího DOM¹ stromu se značkami jazyka HTML. Příklad DOM stromu je zobrazen na obrázku 2.2. Segmentovanými bloky jsou v případě této metody části uvozené v HTML značkách. Příkladem těchto značek je `<TITLE>`, `<H1>` až `<H6>`, `<META>`, `<TABLE>` atd. Tato metoda však má několik problémů, se kterými se musí potýkat:

- Značky jako `<TABLE>` a `<P>` slouží nejen k prezentaci obsahu, ale také pro rozložení struktury. Je obtížné zajistit vhodnou úroveň segmentace.
- Mnoho webových stránek nedodrží specifikace HTML a ztěžují tak správnou segmentaci.

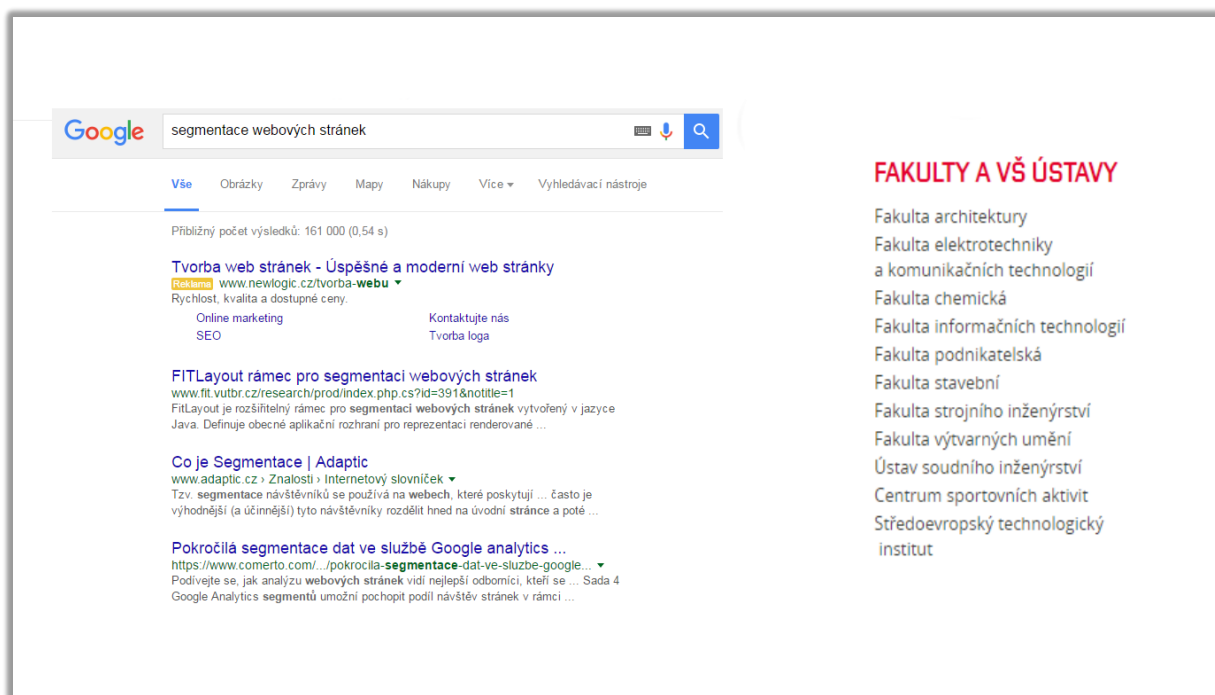
Jako příklad metody DomPS můžeme zmínit algoritmus WISH [17]. Algoritmus zpracuje DOM strom webové stránky a extrahuje požadovaný obsah. Jeho specifikem jsou speciální struktury, které se nazývají *datové záznamy*. Jedná se o části webové stránky, jež se opakují, avšak pokaždé s jiným obsahem. Datové záznamy jsou definovány jako elementy, které se nacházejí na stejné úrovni v DOM stromu. Dále také musí obsahovat sekvence opakujících se potomků a musí mít podobného předka. Předek se nazývá *datová oblast*. Na obrázku 2.1 můžeme vidět ukázky datových záznamů. Jak vidíme, jsou to například výsledky po vyhledávání nebo menu s položkami.

¹ Document Object Model - <http://www.w3.org/DOM/>

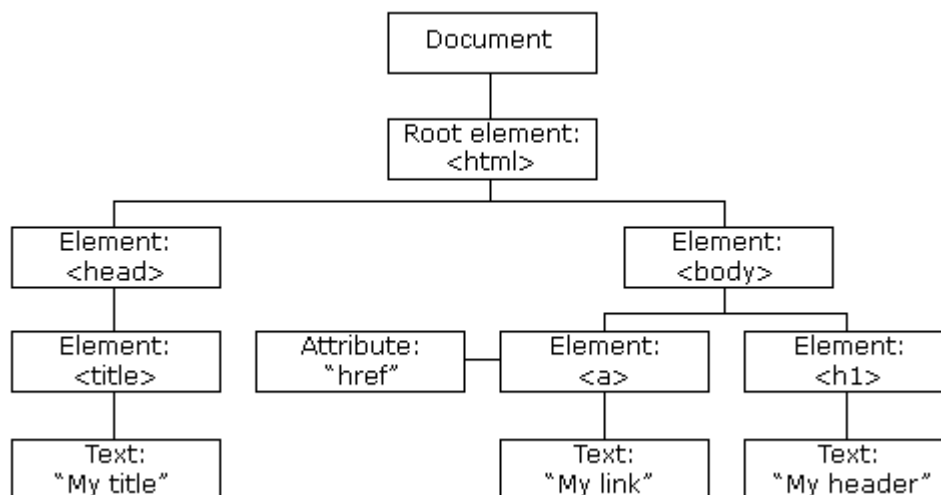
WISH algoritmus nejdříve extrahuje uzly DOM stromu a hledá potenciální datové záznamy. Hledání vykonává pomocí postupného procházení DOM stromu po jednotlivých úrovních. Jestliže není na dané úrovni nalezen datový záznam, hledání se přesune o úroveň níže. Jakmile je hotová detekce všech potenciálních kandidátů na datové záznamy, zahájí se fáze filtrování. Ta využívá několik pravidel pro konečné rozhodnutí, zda se jedná o datový záznam či nikoliv. Pravidla jsou následující:

- Datové záznamy mají velké rozměry v porovnání s rozměry celé stránky.
- Datové záznamy se opakují více než třikrát na celé stránce.
- Z popisu datového záznamu můžeme vytvořit vzor, který můžeme aplikovat na ostatní záznamy.
- Záznamy se obvykle skládají z nízkého počtu HTML značek.

Po dokončení filtrace potenciálních datových záznamů je všem datovým oblastem přiřazena číselná hodnota. Vypočítáme ji na základě obsahu datových záznamů. Datové záznamy se nacházejí uvnitř datových oblastí. Do výpočtu jsou taktéž zahrnuty znaky textu, obrázky atd. Datovou oblast, jejíž vypočítaná hodnota je největší, označíme za hlavní obsah stránky.



Obrázek 2.1: Ukázka datových záznamů algoritmu WISH. Vlevo je seznam po vyhledání, vpravo seznam menu.



Obrázek 2.2: Příklad DOM stromu

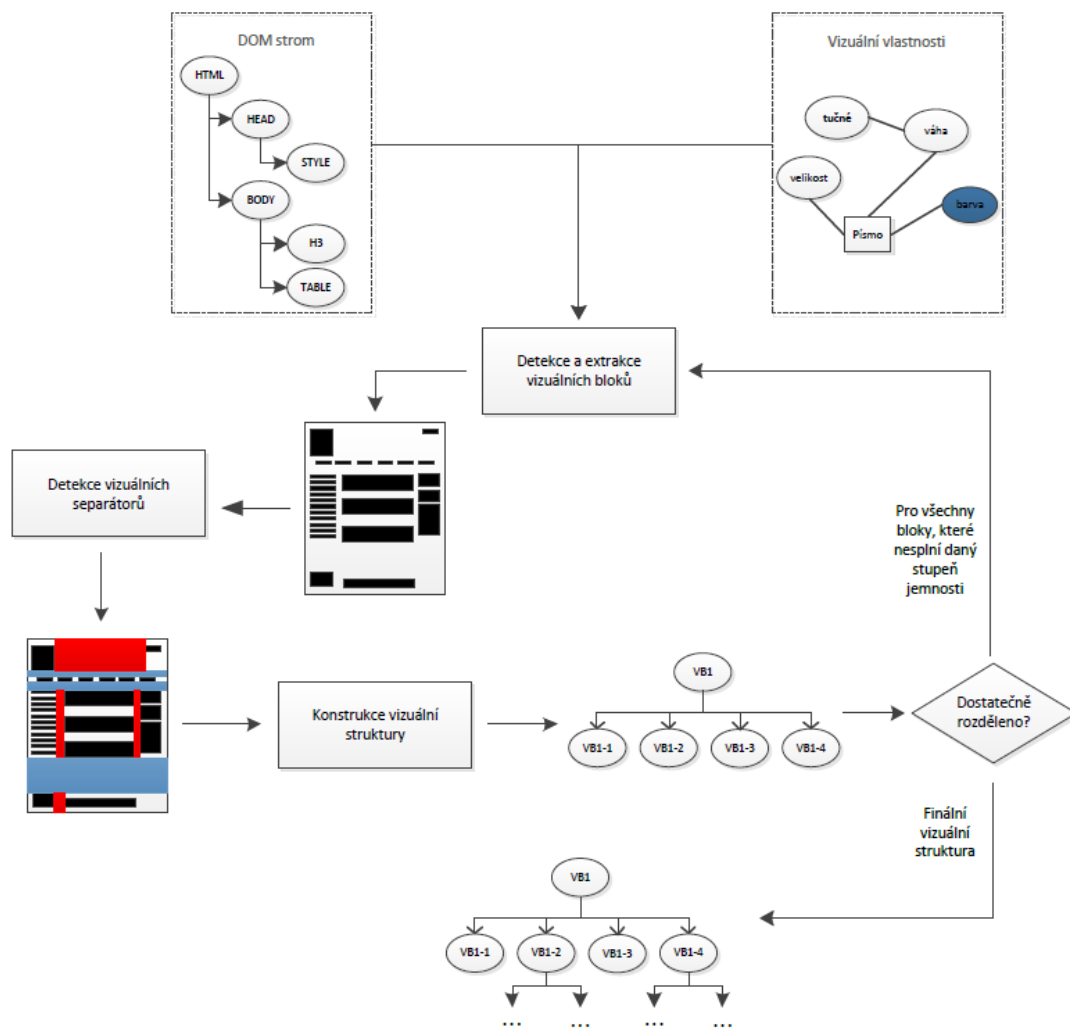
2.2 Vision-based Page Segmentation – VIPS

Webová stránka obsahuje mnoho vizuálních podnětů, které pomáhají člověku rozeznávat pomocí webového prohlížeče jednotlivé stránky. Stránka obsahuje několik dílčích vizuálních prvků, jako jsou obrázky, barvy, čáry a jiné značky. Metoda využívající vlastnosti vizuálních prvků webové stránky je Vision-based Page Segmentation neboli VIPS. Podrobnější popis algoritmu najdeme zde [5].

Vysvětleme si stručně, jakým způsobem VIPS funguje. Nejprve se získá DOM strom z webového prohlížeče. Jakmile je strom získán, začíná se od kořenového uzlu. Poté probíhá proces extrakce vizuálního uzlu na vizuální uzly dané úrovně z DOM stromu, podle vizuálních podnětů. Každý uzel DOM stromu se zkontroluje a hodnotí se, zda tvoří jednoduchý blok nebo ne. Pokud není jednoduchým blokem, synovský uzel je zkontrolován a hodnocen stejným způsobem jako rodič. Jakmile jsou extrahovány všechny bloky dané úrovně, vloží se do dané ukládací struktury. Dále pak těmto blokům identifikujeme vizuální separátory. Poté, je vytvořena struktura rozložení dané úrovně. Každý nově vytvářený vizuální blok je kontrolován, zda vyhovuje požadavku granularity. Pokud ne, dále se dělí. Jakmile jsou zkontrolovány všechny bloky, výsledná datová struktura tohoto algoritmu je vložena na výstup. Schéma algoritmu VIPS lze vidět na obrázku 2.3 a skládá se z tří částí:

- Extrakce vizuálních bloků – v této části jsou především hledány všechny vizuální bloky nacházející se podstromu. Dalo by se říct, že každý uzel v DOM stromu může reprezentovat vizuální blok.

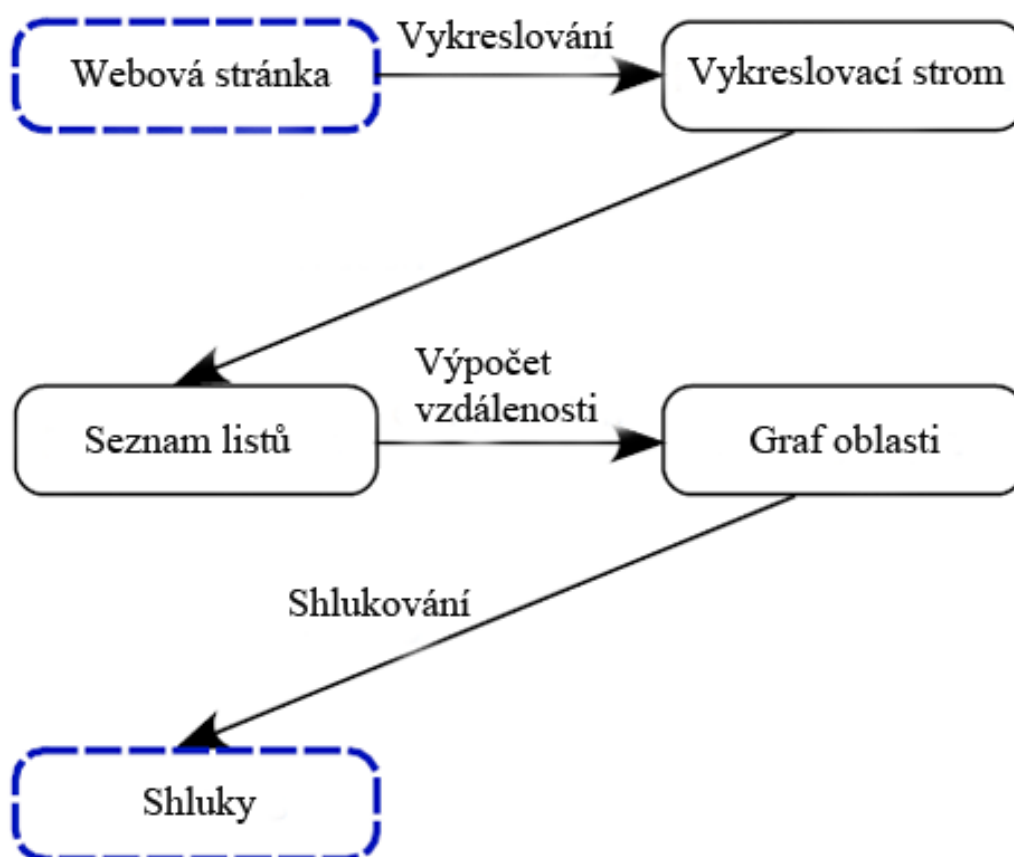
- Detekce vizuálních separátorů – zde se detekují vizuální separátory. Jsou horizontální nebo vertikální. Pomocí nich dokážeme rozpoznat rozdílné vizuální vlastnosti v rámci webové stránky.
- Konstrukce struktury obsahu – v poslední fázi se vytváří výsledná struktura, která je vrácena na výstup. Konstrukce začíná od separátoru s nejnižší váhou a opakuje se až po separátory s nejvyšší váhou. Na závěr tak máme naplněnou strukturu s vizuálními bloky.



Obrázek 2.3: Postup segmentace metody VIPS. Obrázek převzat z [6].

2.3 Box Clustering Segmentation

Posledním příkladem algoritmu, který zde uvedeme, se nazývá Box Clustering Segmentation [7]. Metoda je založená čistě na vzhledu stránky. Když ji porovnáme s ostatními metodami, pak výsledek není v žádné hierarchické struktuře, ale spíše je uspořádán ve struktuře podobné dlaždicím.



Obrázek 2.4: Architektura algoritmu segmentace Box Clustering. Obrázek převzat z [7].

Celou architekturu algoritmu můžeme vidět na obrázku 2.4. Každý box na obrázku představuje stav zpracovávaných dat. Přejít mezi dvěma stavy reprezentuje akci, která se právě provádí. První box označený jako *Webová stránka* představuje vstup celého algoritmu. Vykreslování je prováděno mimo algoritmus a může být pro to použit libovolný způsob vykreslování. Výsledkem vykreslovacího procesu je *Vykreslovací strom*. Samotný algoritmus je uskutečněn pomocí tří kroků:

- Extrakce boxu – zde se ve vykresleném stromu odfiltrují všechny jeho části, které nejsou užitečné pro následující shlukování.

- Výpočet vzdálenosti – vzdálenost mezi zbývajícími boxy se spočítá v druhém kroku a je založena na dalších kritériích popsanych v [7]. V tomto kroku se tvoří neuspořádaný vážený graf.
- Shlukování – v poslední části se zpracuje graf získaný v předešlém kroku a identifikují se segmenty webové stránky. Shlukují se tak boxy patřící do stejného segmentu.

3 Segmentace webových stránek za pomocí vizuální sémantiky

Práce je zaměřená na segmentaci webových stránek za pomocí algoritmu používajícího vizuálních vlastností jednotlivých elementů na webu [8]. Je tak důležité se se zmíněným algoritmem podrobně seznámit. V této kapitole se zaměříme na podrobnější popis již zmíněné metody.

Budeme předpokládat, že se webová stránka skládá z konečného počtu bloků a je možné ji za pomocí vizuálních vlastností, již zmíněných bloků, rozdělit. Vizuální vlastnosti můžeme rozdělit do tří částí:

- Podobné vizuální bloky mají podobné vizuální vlastnosti (např. na stránce internetového obchodu záznamy produktů jsou uspořádány podobným rozvržením).
- Příslušné bloky jsou vždy přehledně uspořádány a to vizuálně blízko sebe.
- Bloky s rozdílnou funkcionalitou obsahují jiné typy obsahu (např. na zpravodajském webu, dlouhý text může obsahovat hlavní obsah stránky, velký obrázek může být inzerát atd.).

I když jsou vizuální vlastnosti rozdílné, tak jsme schopni identifikovat každý segment bez konkrétního popisu. Vizuální vlastnosti zde budeme nazývat vizuální sémantikou. Sémantika je tudíž intuitivní a pro člověka přátelská. Pro stroj však není. Počítač obtížně rozpoznává vizuální sémantiku a je zapotřebí ji formulovat tak, aby ji chápal správně. Musíme tak definovat opatření pro reprezentaci vizuální sémantiky:

- Layout tree (strom rozvržení) – je používán k rozpoznání podobných vizuálních bloků.
- Seam degree (stupeň sousednosti) – používá se k popisu, jak přehledně jsou bloky uspořádány.
- Content similarity (podobnost obsahu) – je používán k popisu obsahu stupně soudržnosti mezi bloky.

3.1 Příprava

Než se pustíme do samotného algoritmu, měli bychom se seznámit s různými vlastnostmi a připravit si webové stránky pro aplikaci algoritmu.

3.1.1 Vizualní bloky

Webová stránka se skládá z konečného počtu bloků. Můžeme zde bloky nazývat buď vizualní bloky, nebo pouze bloky pro zkrácení. Uvažujme, že vizualní bloky jsou viditelné obdélníkové oblasti na webové stránce. Potom vizualní blok definujeme následovně:

Definice 3-1: Vizualní blok $B = (Obj, Rect)$, kde Obj je DOM objekt a $Rect$ reprezentuje viditelnou obdélníkovou oblast, kde B je zobrazen na webové stránce.

Podle W3C² standardu, může být webová stránka transformována do DOM stromu a každý DOM objekt má odpovídající elementy nacházející se na stránce. Pokud je element viditelný, zobrazí se uvnitř obdélníkové oblasti na webové stránce. Proto můžeme říct, že DOM objekt a jeho obdélníková oblast reprezentují blok na webové stránce. Nicméně potřebujeme definovat synovský blok a koncový blok. Definujeme je následovně:

Definice 3-2: Mějme dva vizualní bloky $B_1 = (Obj_1, Rect_1)$ a $B_2 = (Obj_2, Rect_2)$, pokud Obj_1 je synovský uzel Obj_2 , tak B_1 je synovský blok bloku B_2 .

Definice 3-3: Pokud vizualní blok $B = (Obj, Rect)$ nemá žádný synovský blok, pak se jedná o koncový blok.

3.1.2 Předzpracování webových stránek

Podle Definice 3-1, každý vizualní blok má odpovídající DOM objekt. Tudíž první krok našeho algoritmu, který učiníme, je získání DOM stromu z webové stránky.

V DOM stromu se rovná každý uzel DOM objektu. DOM objekt může být rozdělen na pět typů: element, text, atribut, komentář a dokument. Elementy můžeme dále dělit na dvě kategorie a to na viditelné a neviditelné elementy. Viditelné elementy mají dvě vlastnosti: výšku a šířku. Pokud je šířka a výška nenulová, tak lze element zobrazit pomocí prohlížeče. Neviditelné elementy obsahují objekty, jejichž tagy jsou např. `<head>`, `<script>`, `<meta>` atd., které nemají vizualní vlastnosti. Nicméně nás zajímají především ty viditelné, které si můžeme rozdělit do dvou kategorií. Těmi jsou řádkové objekty a více řádkové objekty. Řádkové objekty ovlivňují vzhled textu a mohou být použity na řetězce znaků, které nejsou odřádkovány a zahrnují objekty, jejichž tagy jsou ``, `<big>`, ``, atd. DOM strom však obsahuje spoustu objektů, které jsou v rámci segmentace zanedbatelné. Je tak vhodné je o tyto objekty oříznout. K tomu nám slouží následujících 5 pravidel:

- Atributy uzlů, komentáře k uzlům, uzly dokumentu by měly být vypuštěny.
- Neviditelné uzly, jejichž tagy jsou `<head>`, `<script>`, `<meta>`, atd. by měly být vypuštěny.
- Viditelné uzly, jejichž šířka a výška je nulová a nejsou tak zobrazeny v prohlížeči, by měly být vypuštěny.

² W3C - World Wide Web Consortium je mezinárodní konsorcium, jehož členové společně s veřejností vyvíjejí webové standardy pro World Wide Web

- Pokud uzel obsahuje pouze jeden uzel, jehož uzlové jméno je `<#text>`, potom `<#text>` by měl být vypuštěn.
- Pokud jediný obsah uzlu jsou `<#text>` uzly, řádkové uzly a každý řádkový uzel má pouze jeden `<#text>` uzel, potom všechny `<#text>` uzly a řádkové uzly jsou zanedbatelné.

Jakmile je DOM strom ořezán, jeho zbývajícím obsahem jsou viditelné a textové uzly. Poznamenejme, že oba elementární uzly a textové uzly mají odpovídající objekty v DOM stromu. Podle Definice 3-1 každému vizuálnímu bloku odpovídá elementární DOM objekt a obdélníková oblast. Tudíž je důležité znát obdélníkovou oblast každého elementárního objektu. Elementární objekt neobsahuje absolutní souřadnice odpovídající HTML elementu, ale pouze relativní souřadnice rodičovského HTML elementu. Naštěstí většina prohlížečů nabízí API k získání absolutních souřadnic, a lze tak jednoduše souřadnice získat. Zaměříme-li se na textové uzly, jejich šířka a výška může být vypočítána analýzou výšky a šířky jejich rodičovských a příbuzných uzlů. Po vymezení obdélníkové oblasti, jsou určeny i odpovídající vizuální bloky.

3.2 Algoritmus segmentace

Po pečlivé přípravě se dostáváme k samotnému algoritmu segmentace s využitím vizuálních bloků. Jakmile máme webovou stránku v ořezaném DOM stromu, předáme jej na vstup algoritmu. Ten se skládá z několika částí. Konkrétně ze čtyř, se kterými se dále podrobněji seznámíme.

3.2.1 Rozpoznání podobných vizuálních bloků

Některé webové stránky obsahují podobné vizuální bloky. Např. na obrázku 3.1, se nachází webová stránka e-shopu a modré obdélníky ilustrují bloky s vizuálními vlastnostmi. Každý blok popisuje záznam produktu s jeho krátkým popisem a rozmezím udávající jeho cenu. Můžeme říct, že vyznačené bloky mají podobné vizuální vlastnosti a neměli bychom je dále dělit. A na hledání těchto bloků bychom se měli přesně zaměřit.

Metoda k identifikování podobných vizuálních bloků a získání tak stromu rozložení, se nachází v článku [9]. Každý blok, který není koncovým blokem, můžeme transformovat do stromu rozložení. Příklad takové transformace se nalézá na obrázku 3.2.

Na obrázku se nacházejí dva separátory S_1 a S_2 . Každý separátor může rozdělit blok na dvě menší části. Separátory mohou být považovány za kořeny stromu a dvě menší části mohou být brány jako pravý a levý podstrom. Obvykle, pokud je separátor horizontální, pak je horní část levý podstrom a nižší část pravý podstrom. Pokud je separátor vertikální, pak je levá část levý podstrom a pravá část pravý podstrom. Tudíž můžeme daný blok transformovat na strom.

Máme-li dva bloky, které nejprve transformujeme na stromy rozložení, můžeme vypočítat jejich podobnost. K tomu slouží algoritmu TED (Tree Edit Distance), jehož popis najdeme v článku [10]. Pokud je podobnost menší než nastavený práh, potom jsou dva bloky označeny jako vizuálně podobné. Podle článku [8] je optimální práh 0.4. Za pomoci této metody mohou být rozpoznány všechny podobné vizuální bloky.

3.2.2 Výpočet stupně sousednosti bloků

Bloky chápeme jako viditelné obdélníky nacházející se na webové stránce, a proto jsou vždy uspořádány podle nějakých pravidel. Relevantní bloky se nacházejí vždy vizuálně blízko sebe. Pro každé dva bloky může být jejich uspořádání klasifikováno do tří typů:

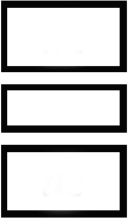
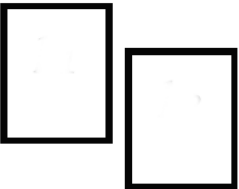
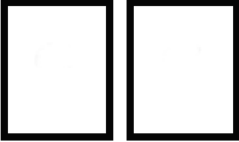
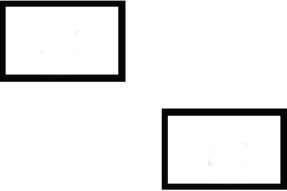
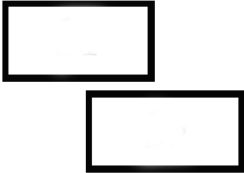
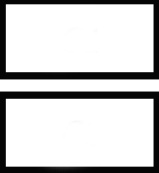
- Nesousedící – tyto bloky považujeme za vizuálně nerelevantní.
- Sousedící – sousedností tvoří obdélník.
- Sousedící – jejich sousednost tvoří minimální obdélník.

Příklad těchto vlastností lze vidět na obrázku 3.3. Zaměříme-li se na vlastnosti jednotlivých sousedností, tak sousedící dva bloky mohou tvořit segment. Nejpravděpodobnější volbou jsou však především bloky, které spolu tvoří minimální obdélník. Jak blízko se dva sousedící bloky nacházejí lze vypočítat, a spočtenou hodnotu budeme nazývat stupeň sousednosti (Seam Degree). Pro dva sousední bloky B_1 a B_2 , konkrétně sousedící shora či zdola, vypočítáme stupeň sousednosti $SD(B_1, B_2)$ následovně:

$$SD(B_1, B_2) = \frac{SeamLength(B_1, B_2)^2}{Width(B_1) \times Width(B_2)} \quad (1.)$$

Kde $SeamLength(B_1, B_2)$ reprezentuje délku sousednosti B_1 a B_2 , a $Width(B_i)$ udává šířku B_i . Podobně pokud jsou dva bloky B_1 a B_2 sousedící zleva nebo zprava, stupeň sousednosti $SD(B_1, B_2)$ vypočítáme:

$$SD(B_1, B_2) = \frac{SeamLength(B_1, B_2)^2}{Height(B_1) \times Height(B_2)} \quad (2.)$$

| | | |
|---|---|---|
|  |  |  |
|  |  |  |
| Nesousedící bloky | Částečně sousedící bloky | Sousedící bloky |

Obrázek 3.3: Ukázka sousednosti bloků. Obrázek převzat z [8].

Kde $Height(B_i)$ je výška B_i . $SD(B_1, B_2)$ se nachází v rozmezí 0 a 1. Jelikož je stupeň sousednosti založen na vizuální informaci bloků, může indikovat vizuální koherentní stupeň odsazených bloků.

Pokud má blok také synovský blok, průměrný stupeň sousednosti navzájem sousedních synovských bloků může indikovat vizuální koherentní stupeň synovských bloků daného bloku. Pro daný vizuální blok B , množina synovských bloků bloku B je $Child(B) = \{b_1, b_2, \dots, b_n\}$. Pokud jsou dva synovské bloky b_i a b_j sousedy, tak se počítají jako jeden pár. Řekněme, že máme m párů navzájem odsazených synovských bloků. Průměrný stupeň sousednosti $AvgSD(B)$ se vypočítá:

$$AvgSD(B) = \frac{\sum SD(b_j, b_i)}{m} \quad (3.)$$

Kde b_j a b_i ($j \neq i$) jsou dva navzájem sousední bloky. $AvgSD(B)$ se nachází taktéž mezi 0 a 1. Pokud získáme výsledek blíže 0, potom vizuální koherentní stupeň synovských bloků je nižší. V opačném případě, pokud získáme výsledek blíže 1, je vizuální koherentní stupeň synovských bloků vyšší.

3.2.3 Výpočet podobnosti obsahu bloků

Bloky s různou sémantikou mají vždy rozdílné typy obsahu. Rozdílné obsahy rovná se také různé vizuální vlastnosti. Pokud se obsah dvou bloků podobá, potom tyto dva bloky nesou vysoký obsahový koherentní stupeň. Nyní uvedeme podobnost obsahu k popisu obsahového koherentního stupně a stručně charakterizujeme obsahy do čtyř skupin:

- Textový obsah (zkráceně TO) – do této kategorie spadá všechn text, kromě textu, který obsahuje hypertextový odkaz.
- Odkazový textový obsah (zkráceně OTO) – tato kategorie obsahuje text, který obsahuje hypertextový text.
- Obrázkový obsah (zkráceně OO) – do této kategorie patří obrázky, fotky, ikony atd.
- Vstupní obsah (zkráceně VO) – tato kategorie zahrnuje elementy, které mohou přijímat vstup, jako jsou: textové boxy, rádiové tlačítko, vysouvací menu atd.

Pro daný blok B , je obsah množina $C = \{c_1, c_2, \dots, c_n\}$. Zaprvé, obsahy klasifikujeme do čtyř kategorií, jenž, byly popsány výše. Poté můžeme získat obsahy těchto čtyř množin, které píšeme zkratkami TO, OTO, OO a VO. Zmíněné množiny jsou podmnožiny množiny C . Pokud se jedna z podmnožin rovná \emptyset , znamená to, že B neobsahuje odpovídající typ obsahu. Obsahem mohou být taktéž elementy na webové stránce a každý z nich, odpovídá nějakému bloku. Použijeme $Area(c_i)$ pro reprezentaci oblasti odpovídajícího bloku obsahu c_i . Pokud c_i je textový nebo odkazový textový obsah, aproximací vypočítáme oblast pomocí:

$$Area(c_i) = Length(c_i) \times FontSize(c_i)^2 (c_i \in TO \cup OTO) \quad (4.)$$

Kde $Length(c_i)$ reprezentuje byte textu, $FontSize(c_i)$ reprezentuje velikost fontu daného textu.

Pro danou obsahovou podmnožinu, v závislosti na oblasti obsahů, může být obsah dané podmnožiny seřazen od velké po menší oblast. Využitím seřazeného obsahu podmnožin, čtyři obsahové oblastní vektory získáme a označíme V_{to} , V_{oto} , V_{oo} , V_{vo} . Hodnoty elementu v těchto čtyřech vektorech jsou oblasti odpovídající obsahům. Jakmile určíme obsahové oblastní vektory dvou daných bloků, podobnost každého obsahového oblastního vektoru můžeme vypočítat.

Existuje mnoho algoritmů, pro vypočítání podobnosti dvou vektorů. Zde využijeme jednoduchého algoritmu a to kosínové podobnosti [11]. Vezmeme vektor textového obsahu jako příklad pro výpočet kosínové podobnosti. Mějme dva bloky B_1 a B_2 , jejich textové obsahové oblastní vektory jsou $V_{tc_1} = (u_1, u_2, \dots, u_m)$ a $V_{tc_2} = (v_1, v_2, \dots, v_n)$. Řekněme, že $V_{tc_1} = \emptyset$, $V_{tc_2} = \emptyset$ a $n > m$. Protože kosínová podobnost vyžaduje, že oba vektory musí mít stejný počet elementů, proto potřebujeme dát do V_{tc_1} $(n - m)$ elementů, jejichž hodnota bude 0. Takto nově vytvořený vektor popisujeme $V'_{tc_1} = (u_1, u_2, \dots, u_m, u_{m+1}, \dots, u_n)$. Kosínová podobnost V'_{tc_1} a V_{tc_2} může být vypočítána pomocí:

$$\text{Cos}(V_{tc_1}', V_{tc_2}) = \frac{\sum_{i=1}^n u_i \times v_i}{\sqrt{\sum_{i=1}^n (u_i)^2} \times \sqrt{\sum_{i=1}^n (v_i)^2}} \quad (5.)$$

Pokud oba V_{tc_1}' a V_{tc_2} , jsou \emptyset , $\text{Cos}(V_{tc_1}', V_{tc_2})$ je nepoužitelný. V tomto případě, definujeme $\text{Cos}(V_{tc_1}', V_{tc_2})$ jako nulový. Podobně vypočítáme podobnost obsahu pro ostatní vektory.

Dodatečně, čtyři typy obsahů mohou mít rozdílné váhy B_1 a B_2 . Zde také můžeme vzít jako příklad textový obsah a vysvětlit na něm výpočet váhy. Pro dané dva bloky B_1 a B_2 , jejichž textové obsahové oblastní vektory jsou $V_{tc_1} = (u_1, u_2, \dots, u_m)$ a $V_{tc_2} = (v_1, v_2, \dots, v_n)$. Váha textového obsahu může být vypočítána vzorcem:

$$\text{Weight}(To) = \frac{\sum_{i=1}^m u_i + \sum_{j=1}^n v_j}{\text{Area}(B_1) + \text{Area}(B_2)} \quad (6.)$$

Kde $\text{Area}(B_i)$ reprezentuje totální oblast všech obsahů v B_i . Znamená to, že čím větší oblast odpovídajícího obsahu tím větší váhu blok bude mít.

Poté co máme vypočítané kosínovou podobnost a váhu všech obsahových vektorů oblastí, můžeme vypočítat podobnost obsahu $CS(B_1, B_2)$ pro B_1 a B_2 následovně:

$$CS(B_1, B_2) = \sum \text{Weight}_i \times \text{Cos}_i \quad (7.)$$

Kde Weight_i reprezentuje váhu čtyř typů obsahu a Cos_i reprezentuje kosínovou podobnost odpovídající obsahovému vektoru oblasti. $CS(B_1, B_2)$ je mezi 0 a 1. Protože podobnost obsahu je založena na informaci obsažené v bloku, může indikovat obsahový koherentní stupeň bloku.

Pokud blok má synovský blok, průměrná podobnost obsahu sousedních synovských bloků může indikovat obsahový koherentní stupeň synovských bloků v bloku. Mělo by být poznamenáno, že pouze uvažujeme o podobnosti obsahu sousedních synovských bloků. Pro daný vizuální blok B je množina jeho synovských bloků $\text{Child}(B) = \{b_1, b_2, \dots, b_n\}$. Pokud dva synovské bloky b_i a b_j spolu sousedí, počítáme je jako 1 pár. Předpokládejme, že máme m párů sousedních synovských bloků. Průměrná podobnost obsahu $\text{AvgCS}(B)$ vypočítáme pomocí:

$$\text{AvgCS}(B) = \frac{\sum CS(b_i, b_j)}{m} \quad (8.)$$

Kde b_i a b_j ($i \neq j$) jsou sousední synovské bloky. Jako předchozí výpočty i $AvgCS(B)$ se výsledek nachází mezi 0 a 1. Pokud se blíží 0, je koherentní stupeň obsahu synovského bloku nižší. V opačném případě pokud se blíží 1, je vyšší.

3.2.4 Segment webové stránky

Jakmile získáme ořezaný DOM strom, můžeme rozpoznat podobné vizuální bloky a stupeň sousednosti. Za další stanovíme podobnost obsahu. Zde uvedeme prahovou hodnotu α $AvgSD(B)$ a prahovou hodnotu β $AvgCS(B)$. Empiricky stanovíme, že α se rovná 0.9 a β 0.8. Naše metoda pro segmentaci webové stránky se provádí shora dolů. Začíná kořenovým uzlem DOM strom, který nastavíme na aktuální právě zpracovávaný uzel. Odpovídající blok aktuálního uzlu zpracováváme podle kroků uvedených v tabulce 3.1. Pokud se aktuální uzel dělí, potom jeho synovské bloky hodnotíme stejným způsobem. V opačném případě aktuální blok nedělíme, ale vložíme jej do seznamu výsledných segmentů a jeho synovské bloky již dále neřešíme. Pseudokód můžeme vidět níže na algoritmu 3.1.

| | |
|--------|---|
| Krok 1 | Pokud je daný blok blokem koncovým tak jej nadále neděl. Jinak pokračuj krokem 2. |
| Krok 2 | Pokud daný blok obsahuje opakující se bloky, potom jej rozděľ. Jinak pokračuj krokem 3. |
| Krok 3 | Pokud je daný blok jedním z opakujících se bloků, potom jej dále neděl. Jinak pokračuj krokem 4. |
| Krok 4 | Pokud daný blok obsahuje pouze jediný synovský blok, potom jej rozděľ. Jinak pokračuj krokem 5. |
| Krok 5 | Pokud $AvgSD(B)$ daného bloku je menší než α , potom jej rozděľ. Jinak pokračuj krokem 6. |
| Krok 6 | Pokud $AvgCS(B)$ daného bloku je menší než β , potom jej rozděľ. Jinak pokračuj krokem 7. |
| Krok 7 | Pokud je oblast daného bloku větší než polovina klientské oblasti, potom jej rozděľ. Jinak pokračuj krokem 8. |
| Krok 8 | Pokud daný blok nepatří do žádné z předešlých podmínek, potom ho neděl. |

Tabulka 3.1: Kroky pro vyhodnocování bloků.

Vstup: DOM strom webové stránky T

Výstup: seznam segmentů $MnožinaSegmentů$

Begin

```
1  $MnožinaSegmentů = \emptyset$ 
2  $AktuálníUzel = T \rightarrow Kořen$ 
3 Segment( $AktuálníUzel$ ) {
4   if  $AktuálníUzel$  je nedělitelný then
5     vlož  $AktuálníUzel$  do  $MnožinaSegmentů$ 
6   end if
7   if  $AktuálníUzel$  je dělitelný then
8      $SeznamSynů = AktuálníUzel \rightarrow Syn$ 
9     for each  $Syn_i \in SeznamSynů$ 
10      Segment( $Syn_i$ )
11   end for
12 end if
13 }
14 return  $MnožinaSegmentů$ 
End
```

Algoritmus 3.1: Pseudokód webové segmentace pomocí vizuálních vlastností.

4 FITLayout

Základním stavebním kamenem práce je dozajista Framework FITLayout³. Jedná se o jednoduše rozšiřitelný Framework pro analýzu webových stránek. FITLayout Je implementován v jazyce Java a definuje obecné Java API pro reprezentaci renderované webové stránky a její rozdělení na vizuální oblasti. Dále poskytuje základ pro implementaci segmentačních algoritmů se společným aplikačním rozhraním.

Vzorová segmentační metoda obsažená ve Frameworku je založena na rekurzivním slučování vizuálních oblastí a detekování separátorů. Framework také zahrnuje nástroje pro zpracování výsledků segmentace různými texty nebo vizuálními klasifikačními metodami. V neposlední řadě poskytuje nástroje pro řízení segmentace a zkoumání jejích výsledků prostřednictvím grafického uživatelského rozhraní.

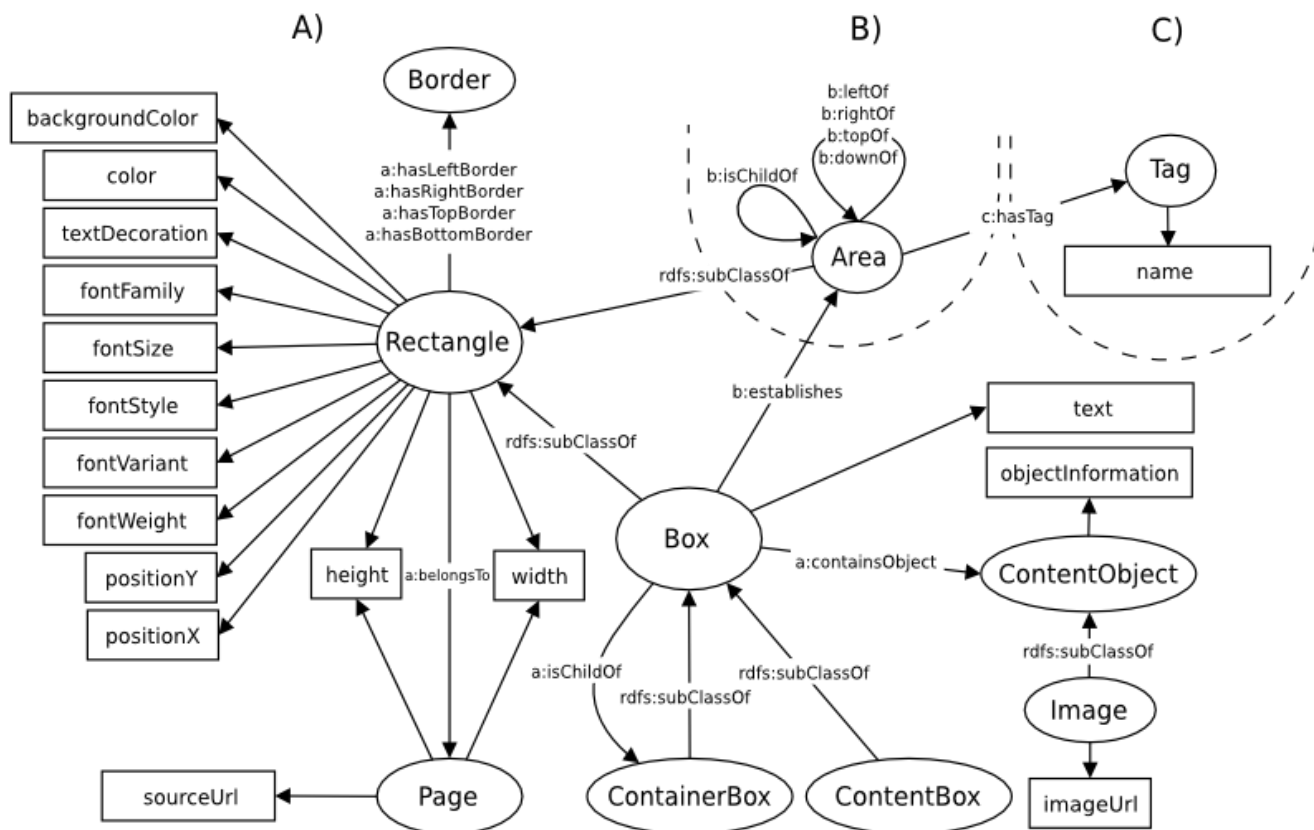
4.1 FITLayout Moduly

Framework se skládá z několika modulů, které jsou navzájem propojeny za pomoci API. Dále také poskytuje přidání dalších modulů či rozšíření stávajících. Za další, můžeme defaultní modul CSSBox, což je renderovací nástroj, nahradit jinou implementací.

4.1.1 FITLayout API

API je založeno na ontologickém popisu zpracovávané stránky [1]. Související ontologie popisujeme na obrázku 4.1. Pro každou třídu ontologie, API definuje rozhraní v jazyce Java s podobnými vlastnostmi. Tyto rozhraní jsou dostupné v balíku `org.fit.layout.model`.

³ Ke stažení na <https://github.com/FitLayout/FitLayout.github.io>



Obrázek 4.1: API ontologie A) box ontologie B) ontologie vizuálních oblastí C) tagování. Obrázek převzat z [13].

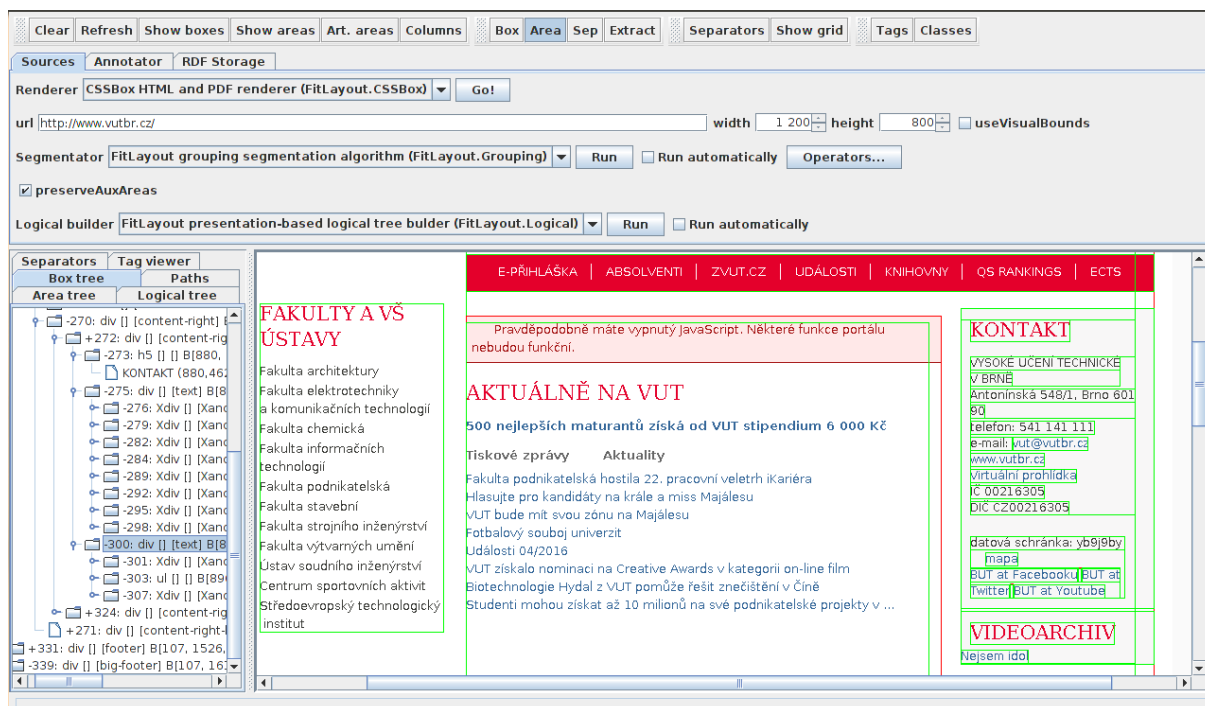
4.1.2 CSSBox

CSSBox [15] používáme jako výchozí renderovací nástroj. Jedná se o (X)HTML/CSS renderovací nástroj napsaný v jazyce Java. Hlavním účelem nástroje je poskytnout kompletní a zpracovatelné informace o poskytovaném obsahu rozložení webové stránky. Nicméně CSSBox se dá taktéž využít pro prohlížení vyrenderovaných dokumentů v Java Swing aplikacích⁴.

Vstupním parametrem nástroje je DOM strom webové stránky a sada stylů odkazovaných z této stránky. Výstupem získáme objektově orientovaný model rozložení stránky. Tento model můžeme přímo zobrazit, avšak převážně jej dále zpracováváme a různým způsobem analyzujeme. Máme na mysli především segmentační algoritmy. V našem případě výstup nazýváme box strom (dále *BoxTree*). Ukázku vyrenderované stránky i s vyznačenými prvky *BoxTree* můžeme vidět na obrázku č. 4.2.

Jádro knihovny CSSBoxu můžeme také použít pro získání bitmapy nebo vektoru obrázku renderované stránky. V kombinaci s Java Swing aplikací CSSBox funguje jako interaktivní součást webového prohlížeče.

⁴Více o těchto aplikacích zde <https://docs.oracle.com/javase/tutorial/uiswing/>



Obrázek 4.2: Ukázka vyrenderované stránky ve FITLayoutu z označenými boxy z *BoxTree*.

4.1.3 Segmentace

Modul segmentace poskytuje obecný Framework pro implementaci algoritmů segmentace webových stránek. Představuje stránku, na které se nachází strom obsahující zjištěné vizuální oblasti. Další důležitou částí, kterou segmentace poskytuje, je definování rozhraní pro implementaci metod zpracování libovolných stromů. Jedná se o tzv. operátory, které implementují skutečnou segmentaci.

Výchozí algoritmus segmentace nalezneme v [2]. Používá přístup zdola nahoru a slučuje jednotlivé boxy do větších vizuálních oblastí. Framework FITLayout je navržen a implementován tak, že se dá snadno rozšířit o další algoritmy nejen pro segmentaci. Výsledkem segmentace stránky je strom vizuálních oblastí.

Při provádění segmentace využíváme několik struktur pro správnou funkčnost algoritmu. Tyto struktury jsou již implementovány ve Frameworku. První využívaná struktura se nazývá *Box*. *Box* reprezentuje box z vyrenderované webové stránky. Konkrétně se jedná o prvek z *BoxTree*. *Box* může obsahovat například text nebo objekt s obsahem (můžeme zmínit například obrázky). *BoxTree* reprezentuje celou vyrenderovanou vstupní webovou stránku.

Další důležitou strukturou, se kterou pracujeme, se nazývá vizuální oblast (*Area*). *Area* obsahuje jeden vizuální box. Strukturu *Area* vnímáme důležitou především proto, poněvadž se na vstupu našeho algoritmu nachází strom oblastí (dále *AreaTree*). *AreaTree* se skládá právě z vizuálních oblastí. Získáme jej z *BoxTree* a použijeme, jak již bylo řečeno,

jako vstupní prvek našeho algoritmu. Můžeme konstatovat, že *AreaTree* považujeme taktéž za výstup našeho algoritmu.

4.1.4 Klasifikace

Jednotlivé oblasti zjištěné při segmentaci stránky mohou být označeny libovolnými značkami. Modul klasifikace implementuje značkování oblastí podle klasifikace jejich vizuálních vlastností [3] nebo dle textových vlastností [4].

4.1.5 Služby

Architektura Frameworku FITLayout se dá jednoduše rozšířit vytvářením nových pluginů jako nový kód pro *BoxTree* (render dokumentů), segmentační algoritmy, přidání nových operátorů pro zpracování *AreaTree* nebo rozšíření grafického rozhraní. Typy služeb, které Framework obsahuje:

- *BoxTreeProvider* – služba vytvářející *BoxTree*. Založen na vstupních parametrech, jako je webová stránka atd. Renderuje stránku na *BoxTree*.
- *AreaTreeProvider* – služba vytvářející *AreaTree*. Veme *BoxTree* a transformuje jej na vizuální *AreaTree*, který reprezentuje vysegmentovanou stránku.
- *AreaTreeOperator* – operace aplikované na vizuální *AreaTree*. Mohou být přidávány další.
- *LogicalTreeOperator* – analyzátor, který přijme výsledný *AreaTree* a přiřadí vybraným oblastem sémantiku.

Každá služba je identifikována za pomoci unikátního identifikátoru poskytovaného metodou `getId()`. Všechny služby mohou přijímat několik vstupních parametrů. Implementují rozhraní *ParametrizedOperation*, jenž dovoluje získat informaci o vyžadovaných vstupních parametrech (jejich jméno a typ) a přiřazuje jim hodnoty.

Pro přístup k službám, FITLayout poskytuje jednoduchý *ServiceManager*, který obsahuje statické metody pro lokalizování služeb daného typu.

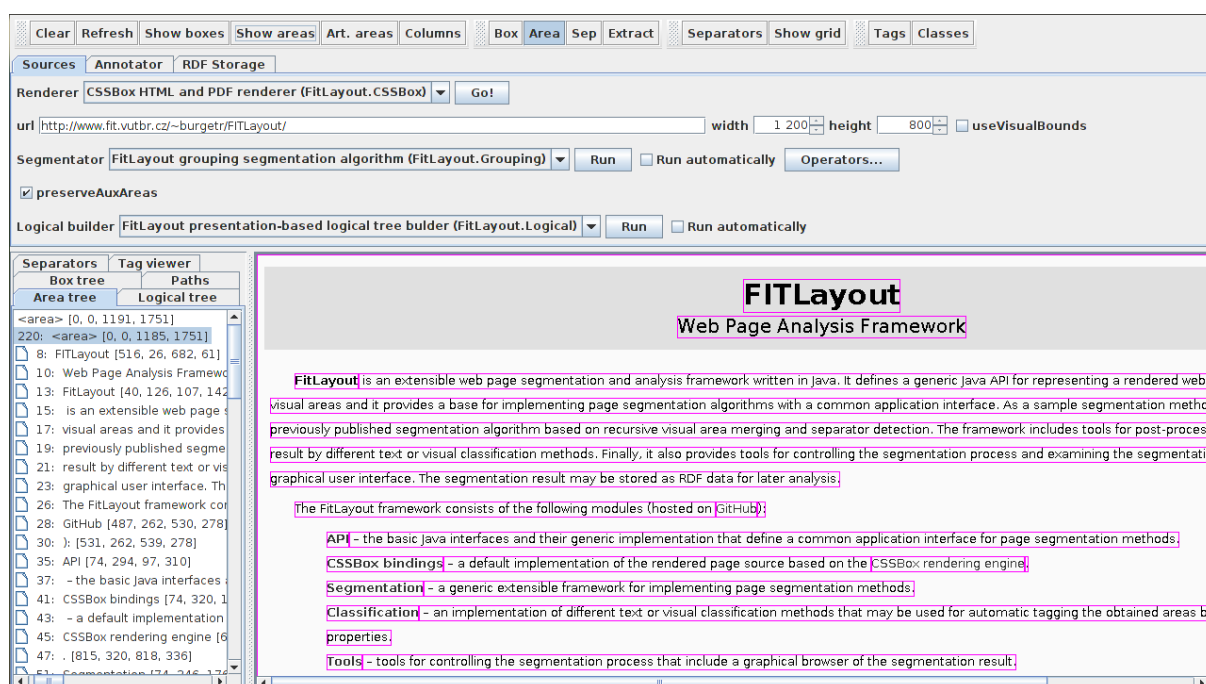
4.1.6 Nástroje

Nástroje FITLayoutu umožňují spustit segmentační algoritmy nad danou URL adresou a zkoumat pak výsledky segmentace za pomoci jeho GUI. Jedná se o dvě části procesor a grafické uživatelské rozhraní.

Procesor je třída zodpovědná za provádění všeho, co se kolem segmentace odehrává. Jedná se především o vytváření stromu základních vizuálních oblastí a aplikování segmentačních operátorů na tento strom. Základní funkčnost je definována pomocí abstraktní třídy *BaseProcessor*. Ve FITLayoutu existují dvě dostupné implementace:

- `ScriptableProcessor` používá Javascript pro nastavení oblastních operátorů, které by měly být aplikovány.
- `GUIProcessor` zde se nachází nastavení operátorů, které mohou být modifikovány zvenčí, typicky za pomoci grafického uživatelského rozhraní.

O spuštění grafického rozhraní se stará třída `BlockBrowser`, kde je implementován výchozí prohlížeč za pomoci Swing prvků. Uživatel si zde může nastavit různá nastavení `BoxTree`, `AreaTree` nebo vybrat a nastavit aplikované operátory. Příklad GUI máme na obrázku 4.3.



Obrázek 4.3: Ukázka GUI ve FITLayoutu. Na obrázku jsou vyznačené vizuální bloky nalezené výchozím segmentačním algoritmem.

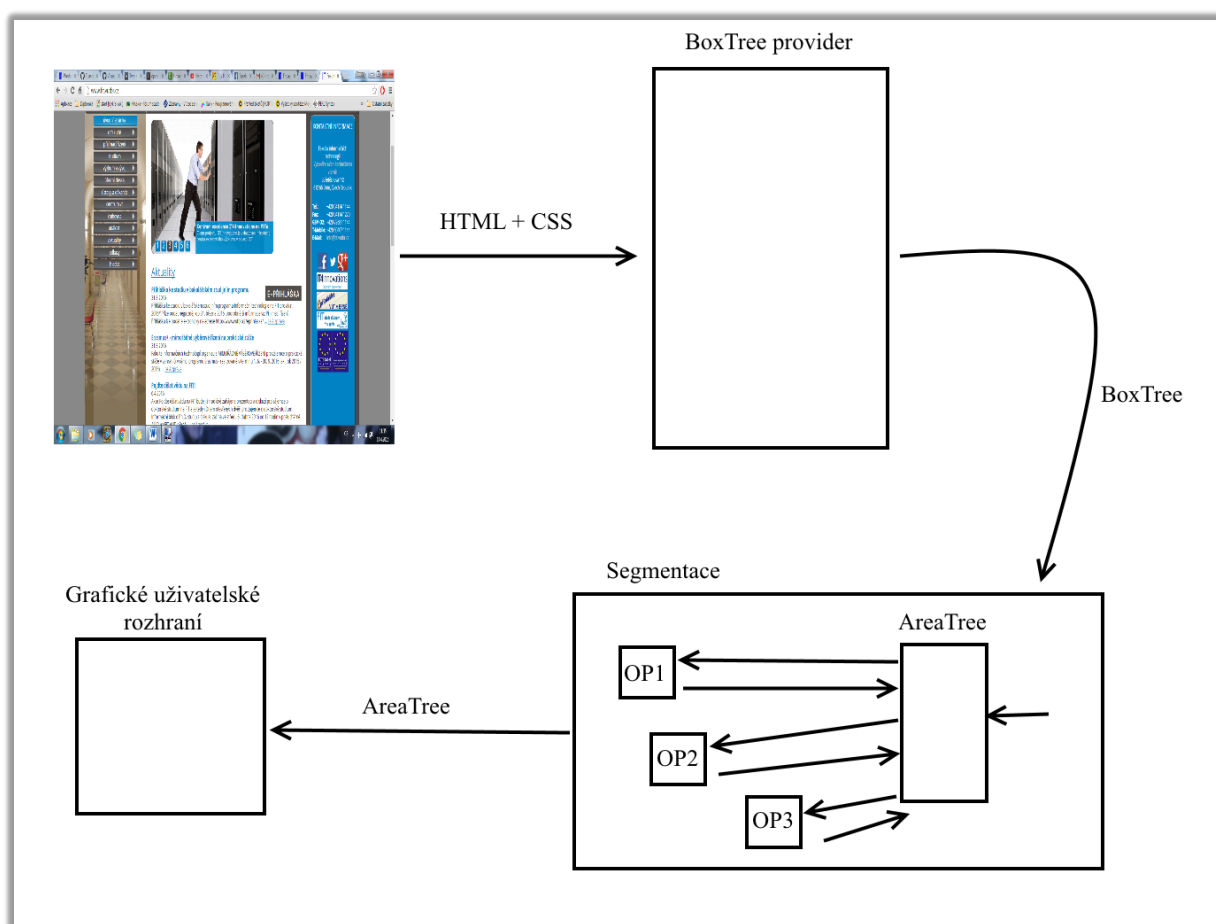
4.2 Práce s Frameworkem

Než se pustíme do samotné implementace algoritmu, vhodně se seznámíme také s ovládáním Frameworku FITLayout. Příklad jak vypadá grafické uživatelské rozhraní, jsme si ukázali na obrázku 4.3. Avšak měli bychom si také sdělit, jak Framework funguje při segmentaci. Důležitými dvěma prvky při segmentaci, za použití Frameworku, jsou `BoxTree` a `AreaTree`.

Prvně jmenovaný prvek vzniká přímo ze vstupní stránky. Odpovídá tak DOM stromu stránky. `BoxTree` získáme ze vstupní webové stránky za pomoci renderovacího nástroje. Ten je v případě Frameworku CSSBox, který byl popsán výše v kapitole 4.1.1. Když to shrneme, můžeme říct, že `BoxTree` je strom boxů, které odpovídají prvkům v DOM stromu vstupní webové stránky.

Naopak *AreaTree* je strom vizuálních oblastí, které získáme převodem z *BoxTree*. *AreaTree* se používá jako vstupní parametr do všech algoritmů pro segmentaci webových stránek ve Frameworku FITLayout. Je taktéž výstupem těchto algoritmů. Na obrázku 4.4 můžeme vidět ilustraci, jak Framework funguje při segmentaci webových stránek.

V tuto chvíli se dostáváme ke zmíněnému ovládání Frameworku. Na obrázku 4.3 vidíme grafické uživatelské rozhraní a je patrné, že jeho ovládání bude velmi jednoduché. Uživatel zde může vložit libovolnou webovou stránku. Poté spustí tlačítkem *Renderer* renderovací nástroj a získá tak *BoxTree* této stránky. Poté si může celý strom vykreslit přímo na zobrazené stránce. Tlačítka mají jednoznačné popisky, tudíž ovládání zvládne každý uživatel. Více informací, jak rozšířit či použít FITLayout se dozvíte zde [16].



Obrázek 4.4: Ilustrace Frameworku FITLayout při provádění segmentace.

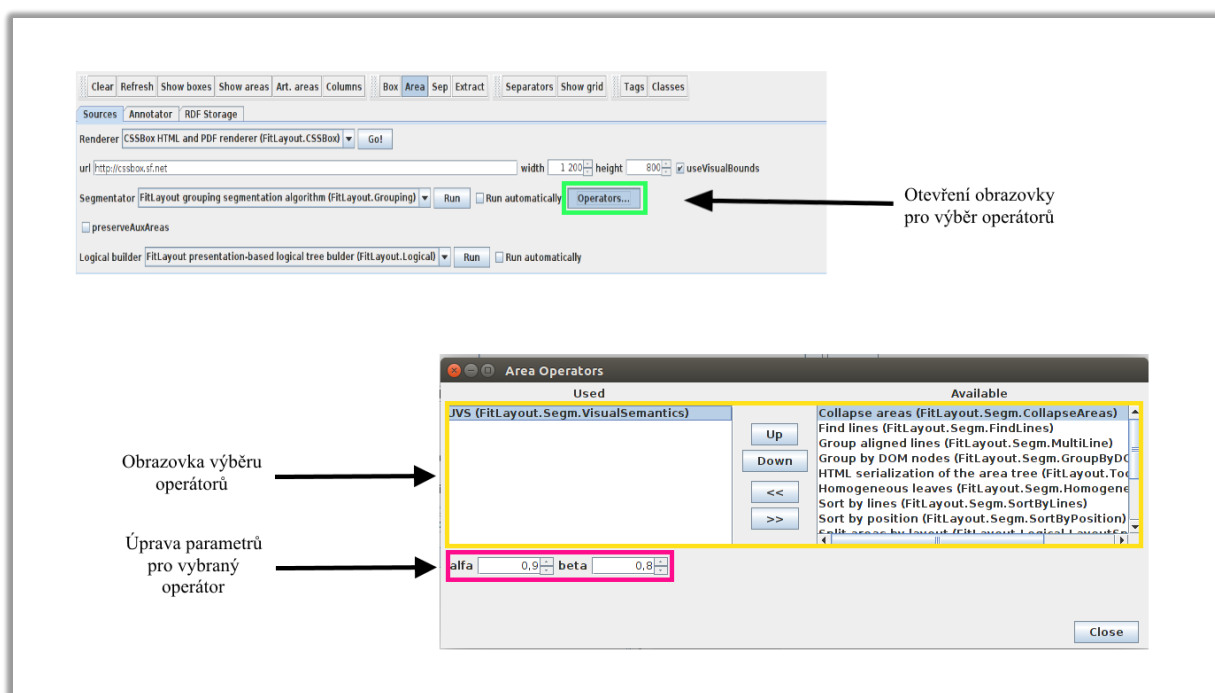
4.2.1 Operátory

Poněvadž rozšiřujeme Framework FITLayout o segmentační metodu, zmínímě, kam ji přesně v rámci Frameworku umístíme, a jakým způsobem bude volána při spuštění. Náš algoritmus implementujeme jako jeden z operátorů, které slouží pro specifikaci segmentace. To znamená, že při provádění segmentace nad vloženou webovou stránkou si můžeme vybrat,

které operátory na ni aplikujeme. Operátory můžeme vzájemně kombinovat a sledovat tak zajímavé výsledky segmentace.

Vstupem operátorů je opět zmíněný *AreaTree* a taktéž je jejich výstupem. Pokud vybereme více operátorů, vzájemně si předávají *AreaTree* a pokaždé na tento vstupní strom aplikují vlastní algoritmus. Poslední operátor tak vrací výstupní strom, který můžeme prozkoumat v grafickém uživatelském rozhraní. Na obrázku 4.5 můžeme vidět, jak vypadá výběr operátorů a zároveň, kde se provádí úprava vstupních parametrů aktuálně vybraného operátoru.

Základní třídou sloužící pro implementaci nového operátoru je *BaseOperator*. Ta zde slouží jako rozhraní, podle něž se tvoří operátory a dědí základní konstrukce metod, které operátory sdílejí. Při tvorbě nového operátoru zdědíme rozhraní a pouze reimplementujeme již předdefinované metody a můžeme jej tak použít v rámci segmentování webových stránek ve Frameworku FITLayout.



Obrázek 4.5: Ukázka výběru operátorů a taktéž úprava vstupních parametrů vybraného operátoru.

5 Implementace algoritmu

Doposud jsme se v práci soustředili pouze na teoretickou stránku použitého segmentačního algoritmu [8]. Nyní přejdeme k jeho implementaci. K tomu slouží tato kapitola, kde se zaměříme na popis implementace jednotlivých částí naší segmentační metody.

Implementace byla vytvořena jako samostatný projekt, který spolupracuje s Frameworkem FITLayout. Skládá se z hlavní třídy, která řídí celý algoritmus, a provádí jej podle uvedených postupů v kapitole 3. Algoritmus byl ve FITLayoutu implementován jako jeden z operátorů. Rozšiřuje rozhraní `BaseOperator` a je zahrnut v konfiguračních souborech. To nám umožňuje v grafickém uživatelském rozhraní Frameworku selektovat námi implementovaný algoritmus mezi ostatními operátory.

Rozebereme si jednotlivé části algoritmu z hlediska implementační části. Zaměříme se především na záludnosti, se kterými jsme se museli vypořádat, a zároveň jak jsme očekávané problémy vyřešili.

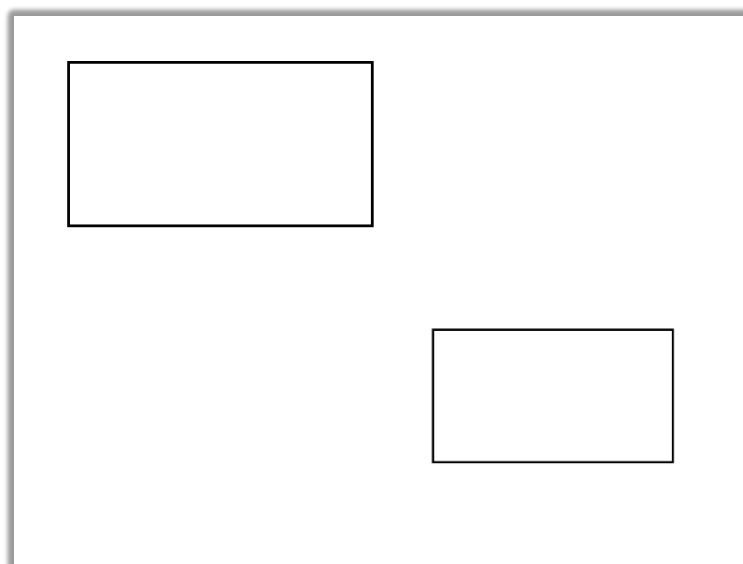
5.1 Sousednost bloků

Důležitou částí implementace segmentačního algoritmu podle vizuálních vlastností stránky je výpočet průměrného Seam Degree. Sousedností bloků rozumíme jejich vzájemné uspořádání. V kapitole 3.2.2 jsme si definovali sousedící a nesousedící bloky a taktéž Seam Degree. Při programování této části vyvstaly určité problémy, se kterými jsme se museli potýkat. Vyvstalé problémy si dále přiblížíme i s postupem jejich řešení.

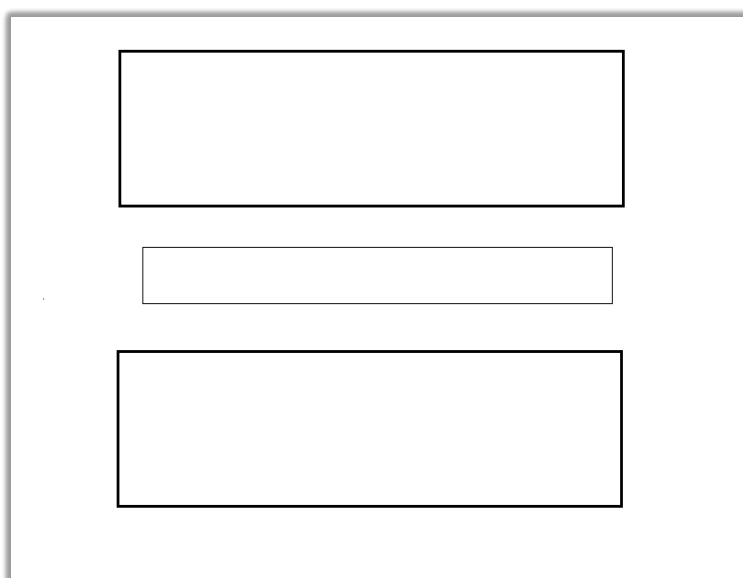
5.1.1 Nesousedící bloky

Dříve než začneme počítat Seam Degree pro dva dané bloky, potřebujeme určit, zda jsou bloky sousedící či nikoliv. To můžeme určit několika způsoby, avšak zde si ukážeme pouze ta řešení, na které jsme se při implementaci zaměřili. Taktéž musíme říci, že mohou být dva druhy navzájem nesousedících bloků:

- Dva bloky vzájemně se nepřekrývající žádnou částí vertikálně, ani horizontálně. Ukázku takových bloků lze vidět na obrázku č. 5.1.
- Dva bloky, mezi kterými se nachází další blok/bloky. Ukázku tohoto typu nesousedících bloků můžeme vidět na obrázku č. 5.2. Uvažované nesousedící bloky jsou na obrázku zvýrazněny tučně.



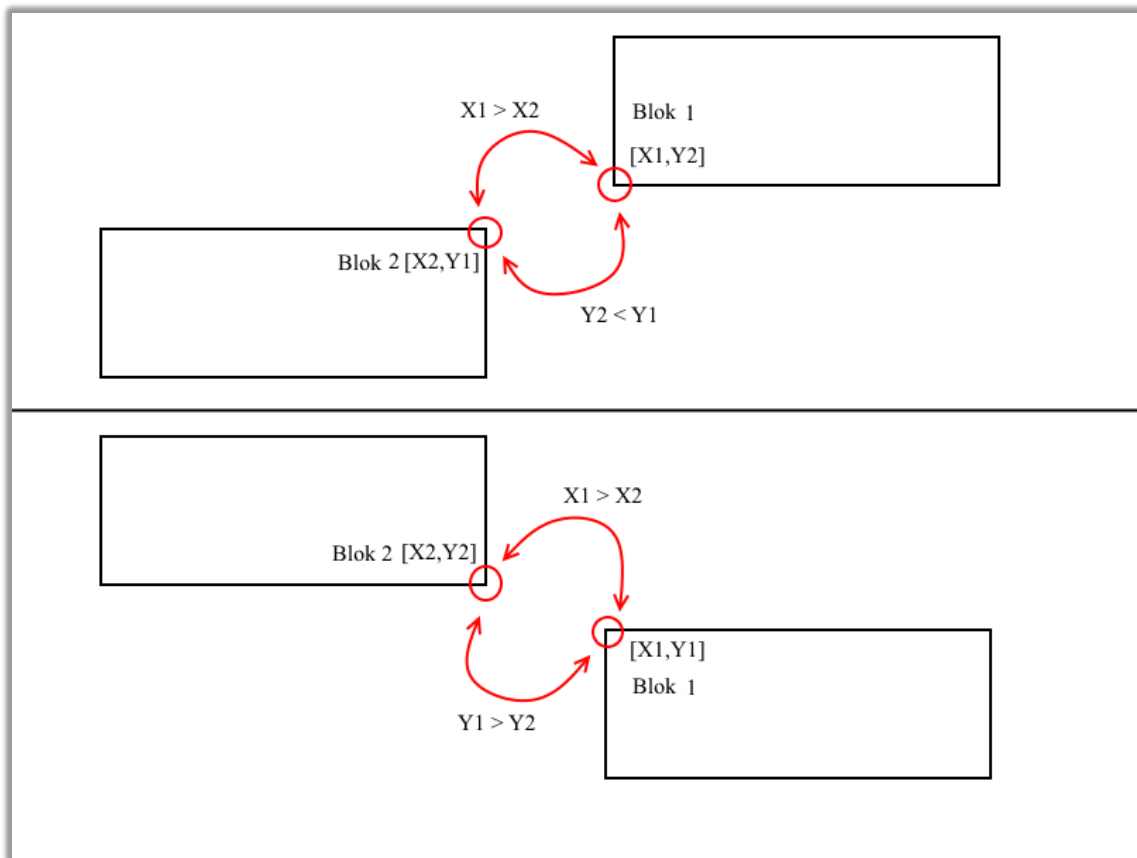
Obrázek 5.1: Ukázka nepřekrývajících se bloků.



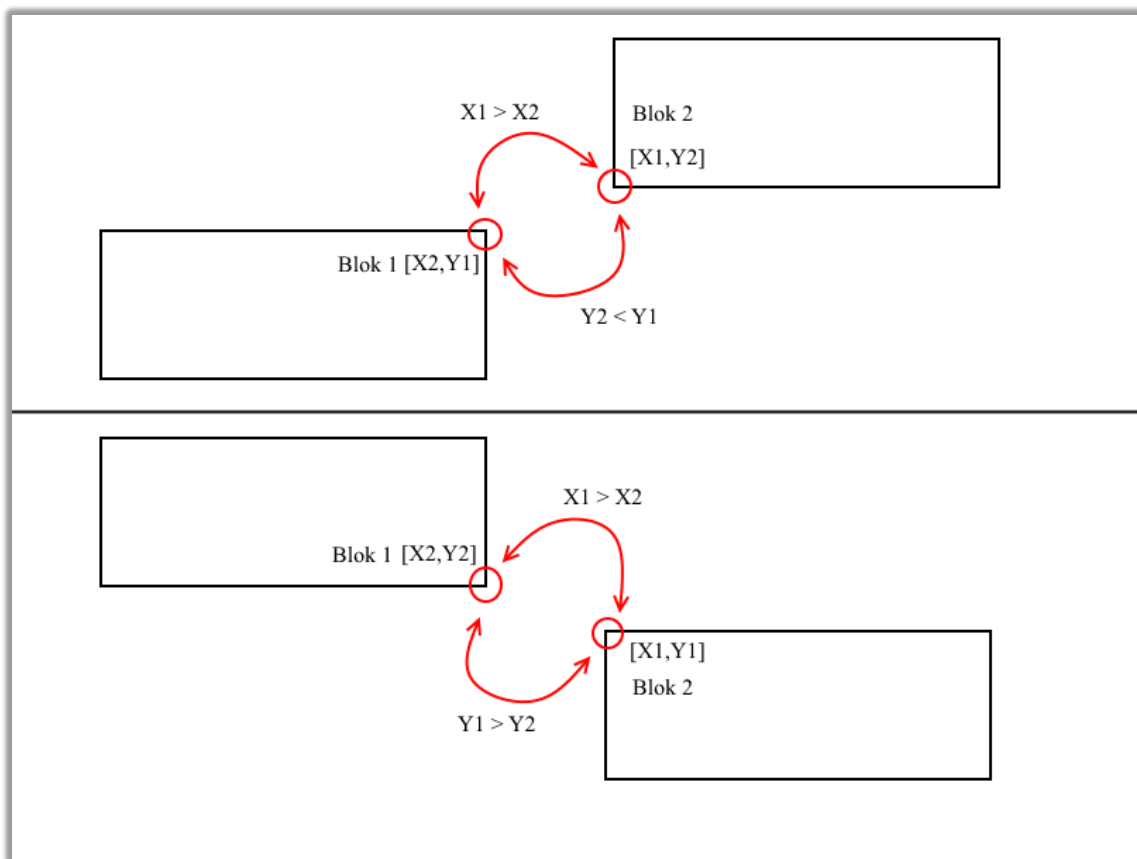
Obrázek 5.2: Ukázka nesousedících bloků, mezi nimiž je jiný blok.

První typ nesousedících bloků vyřešíme za pomoci souřadnic krajních bodů obou bloků. Musíme porovnat jak souřadnice pro vertikální sousednost, tak i pro horizontální. Na obrázcích 5.3 a 5.4 vidíme, za pomoci šipek, které souřadnice daných bloků se porovnávají. A taktéž, která z nich musí být větší nebo menší, aby byly dva bloky nesousedící.

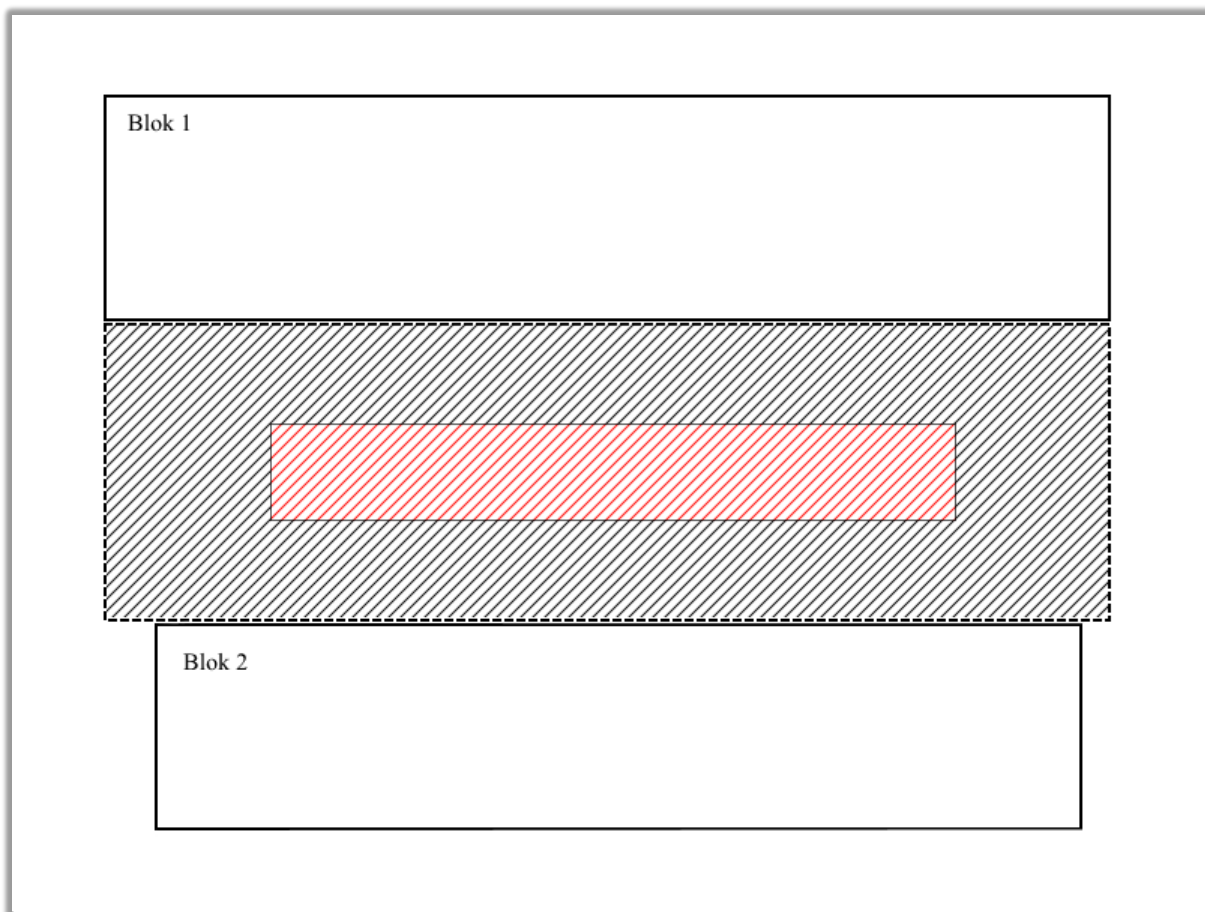
Pokud se mezi dvěma porovnávanými bloky nachází jiný blok, jsou tyto bloky nesousedící. Implementačně jsme to vyřešili za pomoci obdélníku. Ten je vytvořen jako oblast mezi porovnávanými bloky. Pokud se nachází v tomto obdélníku jiný blok, znamená to již zmíněnou situaci a bloky jsou určeny jako nesousedící. Celou tuto situaci můžeme vidět na obrázku č. 5.5. Černě vyšrafovaný obdélník znázorňuje náskres pomocného obdélníku a červeně vyšrafovaný obdélník zobrazuje nalezený blok mezi dvěma danými bloky.



Obrázek 5.3: Ukázka nesousedících bloků. Blok 2 se nachází vlevo a blok 1 vpravo.



Obrázek 5.4: Ukázka nesousedících bloků. Blok 2 se nachází vpravo a blok vlevo.



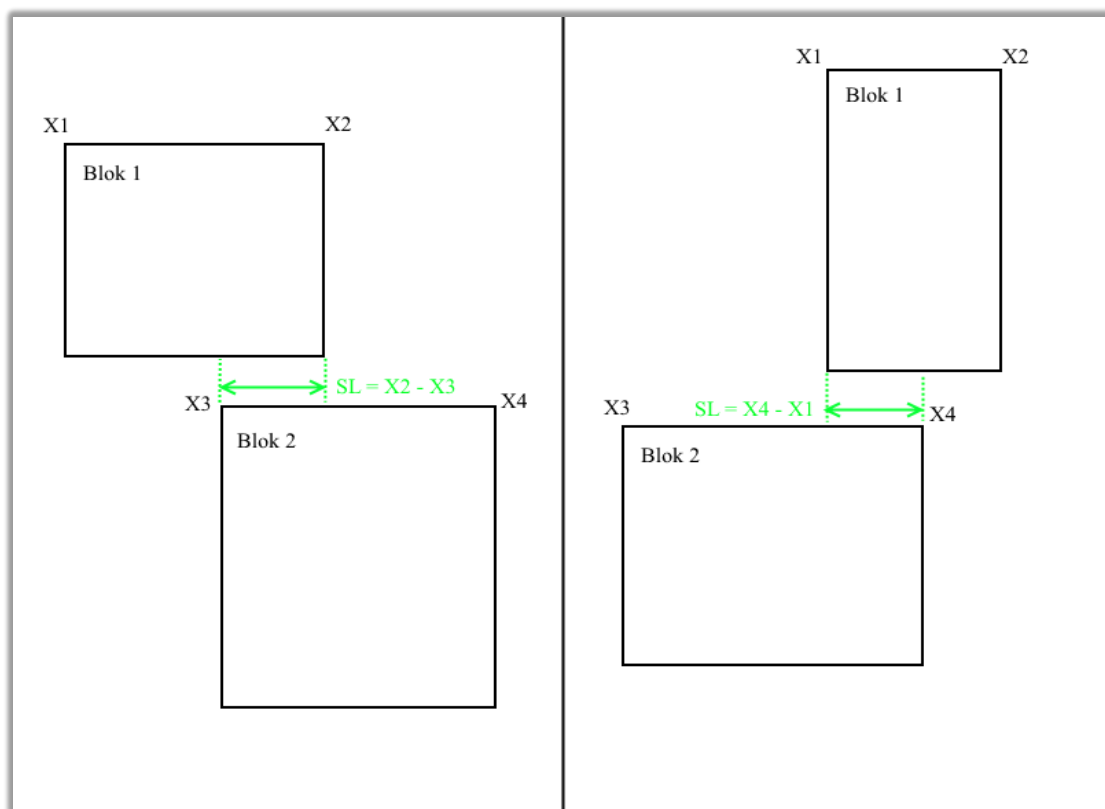
Obrázek 5.5: Ukázka hledání bloku mezi danými dvěma bloky.

5.1.2 Sousedící bloky

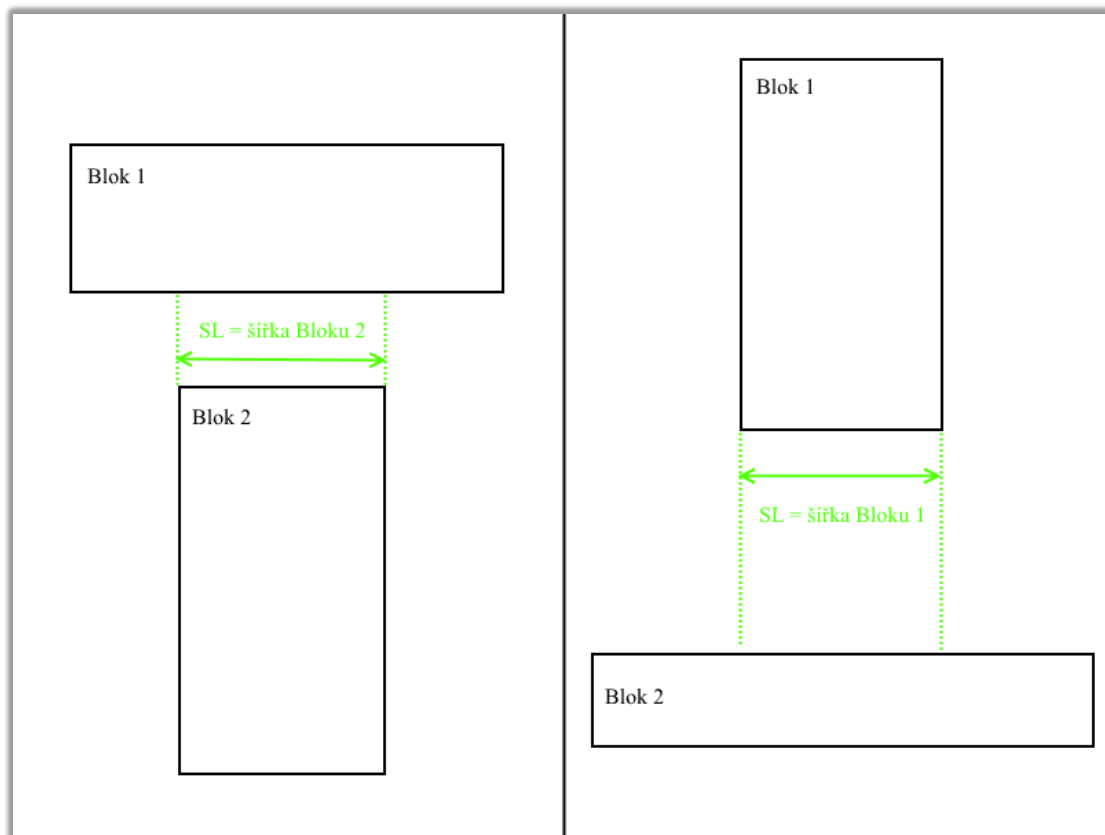
Pokud jsou dva bloky sousedící, musíme zajistit informaci o tom, jak spolu navzájem sousedí. Z toho vyplývá, že musíme zjistit, zda spolu sousedí vertikálně nebo horizontálně. Určení, zda jsou bloky sousedící, získáme z informace, že nejsou nesousedící.

5.1.2.1 Horizontální sousednost

Horizontálně sousedící bloky jsou uspořádány vedle sebe. Při horizontálně sousedících blocích používáme vzorec č. 1. Problém, který musíme vyřešit, se nachází při výpočtu hodnoty *SeamLength*, značící délku překrytí. Při výpočtu hodnoty, která udává délku horizontálního překrytí bloků, musíme vzít v potaz vzájemnou polohu dvou daných bloků. Všechny možné způsoby, jak mohou být dva bloky vzájemně horizontálně uspořádány, můžeme vidět na následujících obrázcích č. 5.6 a č. 5.7. U každého obrázku taktéž můžeme vidět, jak se vypočítá délka překrytí, pro dané vzájemné umístění dvou bloků. Délku překrytí označujeme na obrázcích, jako *SL*. Doplňme, že písmena X_i značí x-ové souřadnice bodu v daném místě bloku.



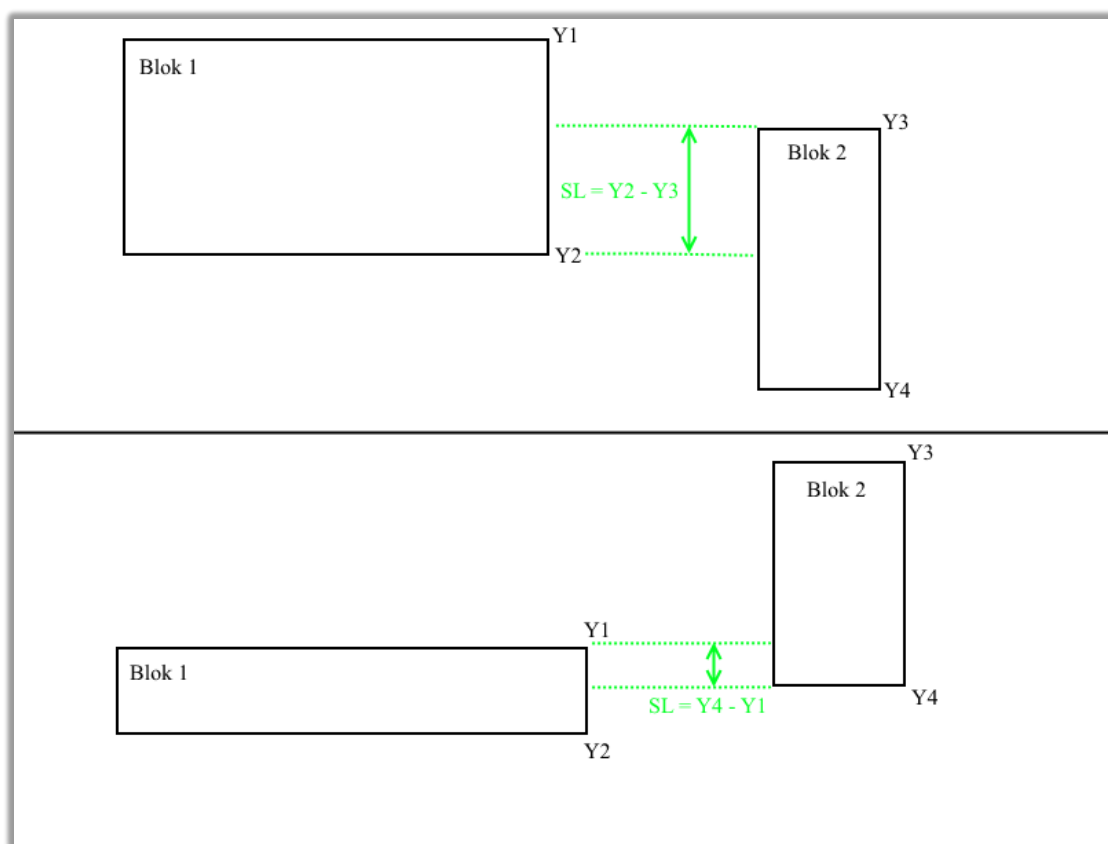
Obrázek 5.6: Ukázka výpočtu horizontální délky překrytí, při částečném překrytí.



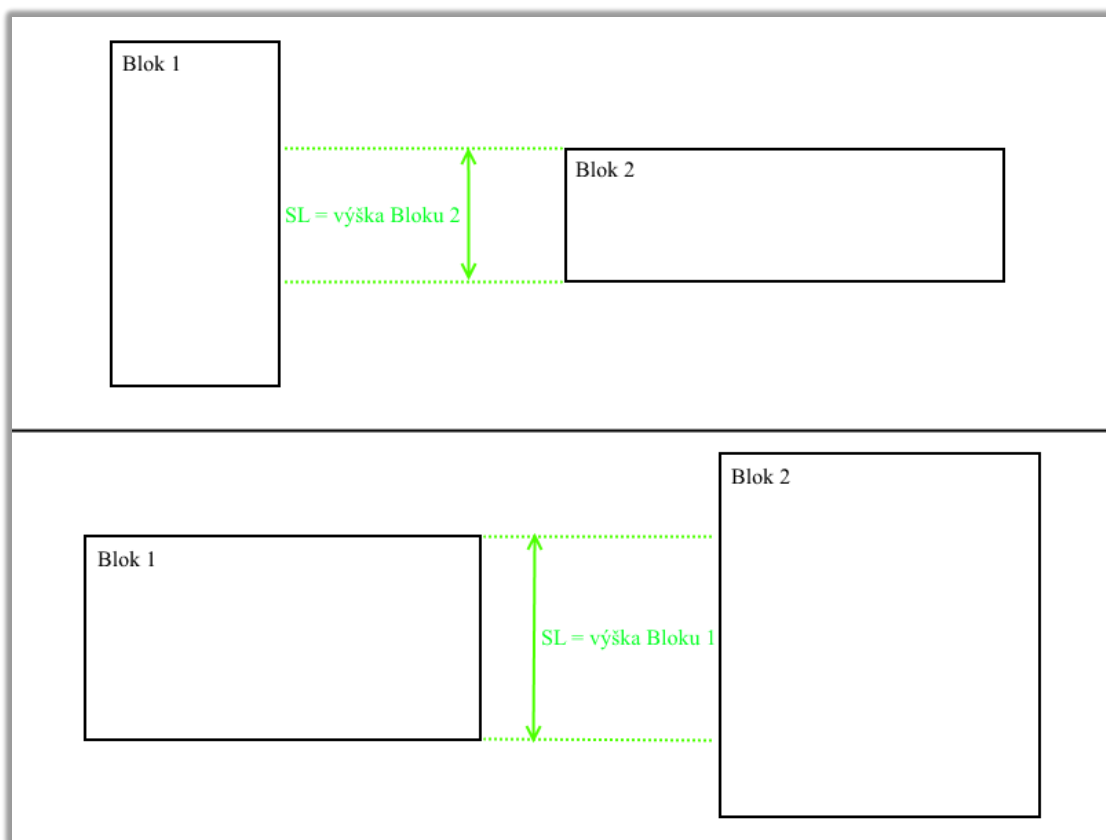
Obrázek 5.7: Ukázka výpočtu horizontální délky překrytí, při úplném překrytí.

5.1.2.2 Vertikální sousednost

Vertikálně sousedící bloky jsou uspořádány pod sebou. Při vertikálně sousedících blocích používáme vzorec č. 2. Problém, který musíme vyřešit, se nachází opět při výpočtu hodnoty *SeamLength*. Počítáme-li hodnotu, která udává délku vertikálního překrytí bloků, musíme vzít v potaz vzájemnou polohu dvou bloků. Všechny možné způsoby, jak mohou být dva bloky vzájemně vertikálně uspořádány, můžeme vidět na následujících obrázcích č. 5.8 a č. 5.9. U každého obrázku taktéž můžeme vidět, jak se vypočítá délka překrytí, pro dané vzájemné umístění dvou bloků. Doplňme, že písmena Y_i značí y-nové souřadnice bodu v daném místě bloku.



Obrázek 5.8: Ukázka výpočtu vertikální délky překrytí, při částečném překrytí.



Obrázek 5.9: Ukázka výpočtu vertikální délky překrytí, při úplném překrytí.

Jakmile zjistíme, zda jsou bloky sousedící či nikoliv, můžeme při kladné odpovědi vypočítat délku překrytí daných dvou bloků. Avšak, než ji vypočítáme, musíme zjistit, jaká je jejich vzájemná poloha. Jakmile máme vypočítanou hodnotu délky překrytí, můžeme vypočítat hodnotu Seam Degree pro aktuálně dané dva bloky. Dle algoritmu popsaneho v kapitole 3. můžeme dále spočítat průměrnou hodnotu Seam Degree pro všechny bloky uvnitř aktuálně zpracovávaného bloku. V tabulce 3.1 lze vidět, kterého kroku se tento výpočet týká. Konkrétně se jedná o krok pět. Pokud vypočítaná hodnota nesplňuje zadanou podmínku, přechází se na následující krok algoritmu. Splňující podmínkou v tomto případě myslíme porovnání vypočítané hodnoty s parametrem *alfa*.

5.2 Vizuální obsah bloků

Další nedílnou částí námi popisovaného algoritmu je výpočet podobnosti obsahu bloků. Obsah vizuálního bloku může být různý a při rozhodování zda daný blok dále dělit či nikoliv, potřebujeme vědět, co obsahuje. Podle toho, pak můžeme hodnotu vypočítat. Dále si přiblížíme, jakým způsobem jsme tento výpočet implementovali.

5.2.1 Vektory obsahu

Jak jsme si již sdělili, obsah vizuálního bloku může být různý. Jedná se například o obrázkový obsah, textový obsah atd. Máme-li daný blok, potřebujeme pro výpočet jednotlivých vzorců, uvedených kapitole 3. znát tzv. vektory obsahu. Vektorem se v tomto případě myslí seznam. Tento seznam obsahuje vizuální oblasti, které jsou pro něj určené. Tudiž to může být vektor textového obsahu, obrázkového obsahu atd. Potřebujeme tak vzít všechny části, které daný blok obsahuje a zjistit, jaký obsah má každá dílčí část a naskládat je do jednotlivých vektorů. Na obrázku č. 5.10 můžeme vidět příklady, jak může takový obsah daného bloku vypadat.

Pokud chceme zjistit obsah bloku přímo ve Frameworku FITLayout, musíme přistoupit k Boxu daného bloku. To znamená, že si vezmeme DOM uzel toho bloku za pomoci *BoxTree*, který byl popsán v kapitole 4. a můžeme pak pomocí metod přistoupit přímo k hodnotě určující typ obsahu toho bloku.



Obrázek 5.10: Ukázka vizuálních prvků na webové stránce. Žlutě jsou označené bloky obsahující převážně textový obsah a modře je označen blok, jak s textovým obsahem, tak i s obrázkovým.

5.2.2 Podobnost obsahu

Jakmile získáme vektory obsahu pro dané dva bloky, můžeme přejít k dalším částem výpočtu podobnosti obsahu. Ten jich obsahuje několik, které jsme si již definovali v kapitole 3. K výpočtu podobnosti obsahu potřebujeme další dvě pomocné hodnoty. Jedná se o kosínovou podobnost a váhu.

V první řadě chceme zjistit podobnost jednotlivých typů vektorů pro dva dané vstupní bloky. Podobnost dvou vektorů můžeme spočítat několika způsoby, avšak my jsme z důvodu jednoduchosti zvolili kosínovou podobnost. Ta předpokládá stejnou délku počítaných vektorů. Při neshodě se kratší vektor doplní nulami, aby se velikosti obou vektorů rovnaly. Jakmile splníme tento požadavek, můžeme přejít na výpočet. Ten se provádí, dle rovnice č. 5. Nastane-li situace, kdy jsou oba vektory prázdné, výsledná hodnota kosinové podobnosti je nulová.

Další hodnotu, kterou potřebujeme, je váha daného typu vektoru. Je to dáno tím, že každý blok, může mít například víc textového obsahu než obrázkového atd. Váha se počítá jako součet sum daného typu vektoru, pro který počítáme váhu, poděleno součtem celkových oblastí zabíraných danými dvěma bloky.

Máme-li hodnotu kosínové podobnosti a váhy pro daný typ vektoru, můžeme vypočítat podobnost obsahu. Získáme-li průměrnou podobnost obsahu daného bloku, porovnáme jej s definovanou hodnotou, v našem případě je to parametr *beta*, a tím rozhodneme, zda budeme opět blok dělit či nikoliv.

5.3 Řízení algoritmu

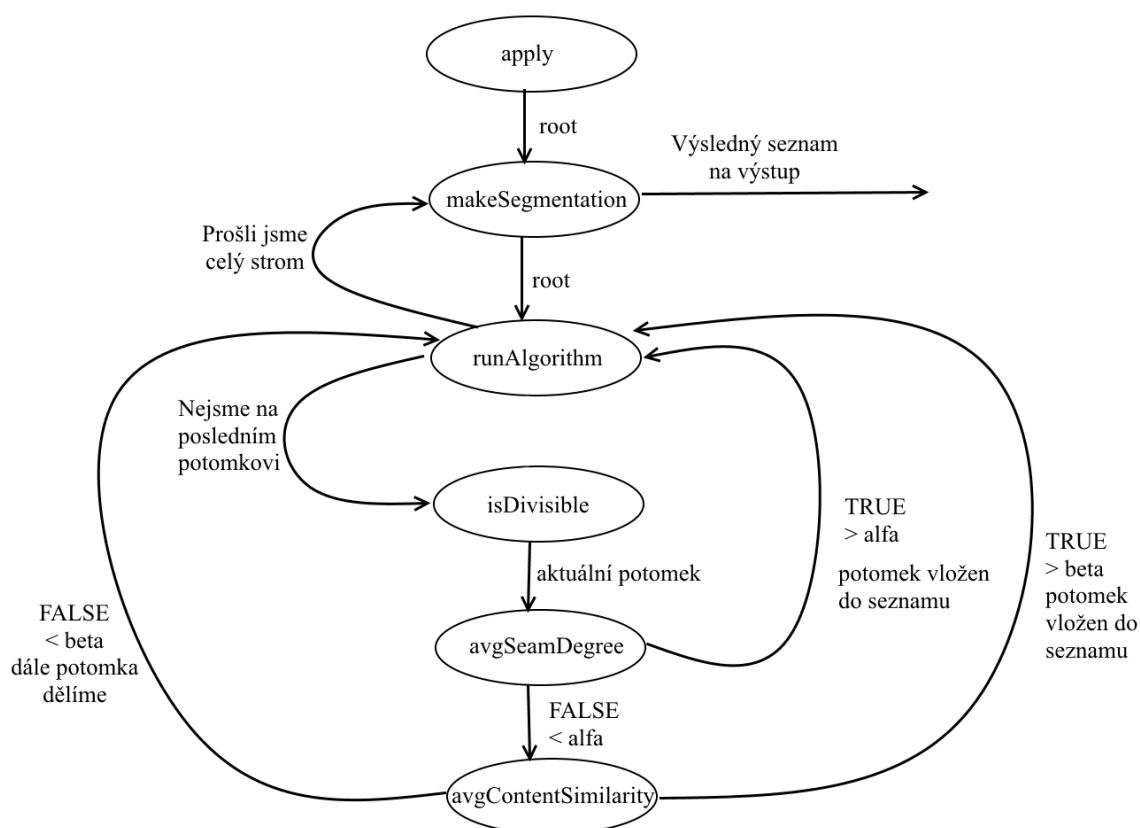
Je pravdou, že k tomu, abychom mohli provádět segmentaci webových stránek, nám nestačí si říct, jak vypočítat několik vzorců. Musíme si sdělit, jakým způsobem celý použitý algoritmus řídíme, a s tím taktéž souvisí tvorba výstupní posloupnosti.

Celý algoritmus je řízen především z jedné třídy, ve které jsou implementovány všechny metody a atributy důležité pro vykonání metody. Třída se nazývá `VisualSemanticsOperator` a rozšiřuje rozhraní `BaseOperator`. To znamená, že musí obsahovat několik pevně daných metod, které musí být implementovány. Ty slouží především pro správnou funkčnost z hlediska grafického rozhraní a taktéž spuštění námi implementovaného algoritmu.

Po spuštění algoritmu se jako první volá metoda `Apply()`. Této metodě je předán jako vstupní parametr `AreaTree`, který je velmi blízký DOM stromu, jenž je brán jako vstupní prvek našeho algoritmu. Poté můžeme začít s algoritmem. Ten řídí metoda `makeSegmentation()`. Ta obdrží kořenový uzel stromu a nad ním spustí rekurzivní metodu `runAlgorithm()`. Metoda prochází celý strom a podle pravidel v tabulce 3.1 řeší

dělení jednotlivých prvků vstupního stromu. Prvkem vstupního stromu myslíme vizuální blok.

Pokaždé, když se blok nedělí, vložíme jej do seznamu. Jakmile je seznam hotový, končí také rekurzivní funkce. Může tak být tvořen výstupní strom, který má stejný formát jako vstupní. Opět se jedná o *AreaTree*. Bereme postupně uzly z našeho seznamu a vytváříme výsledný strom. U každého uzlu zachováváme jeho potomky, aby byl strom hierarchicky uspořádán. Na obrázku č. 5.11 vidíme přehledný graf postupně volaných metod celého algoritmu.

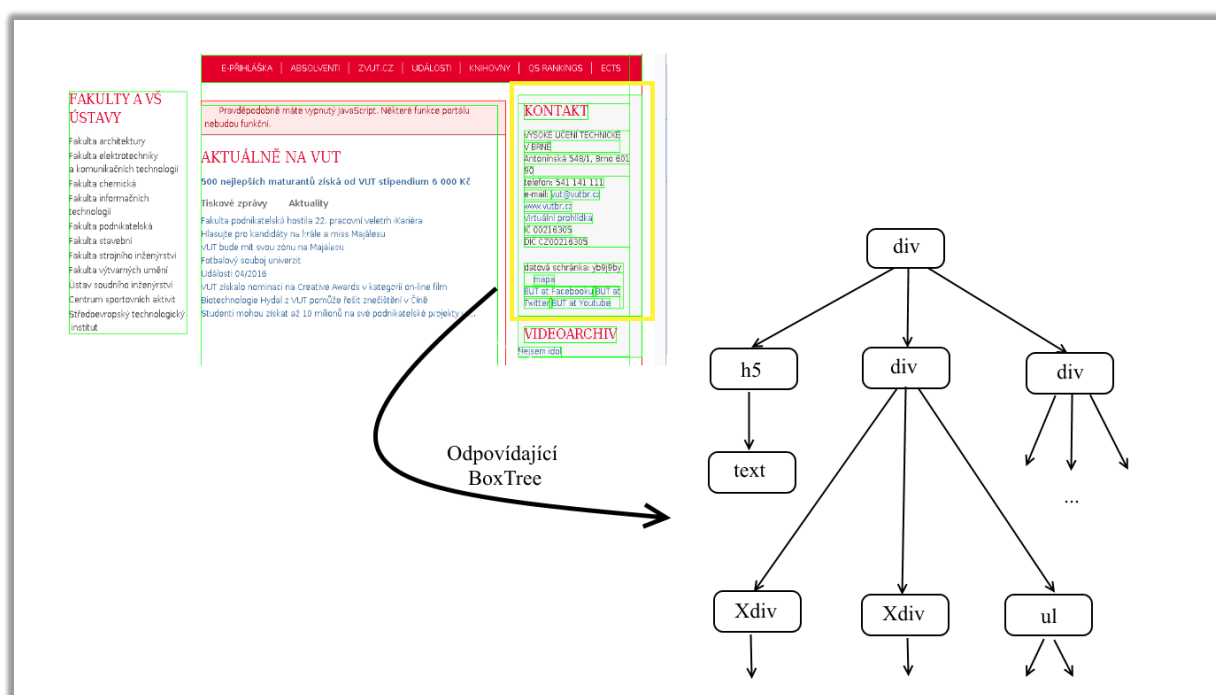


Obrázek 5.11: Zjednodušený obrázek grafu volaných metod při provádění algoritmu.

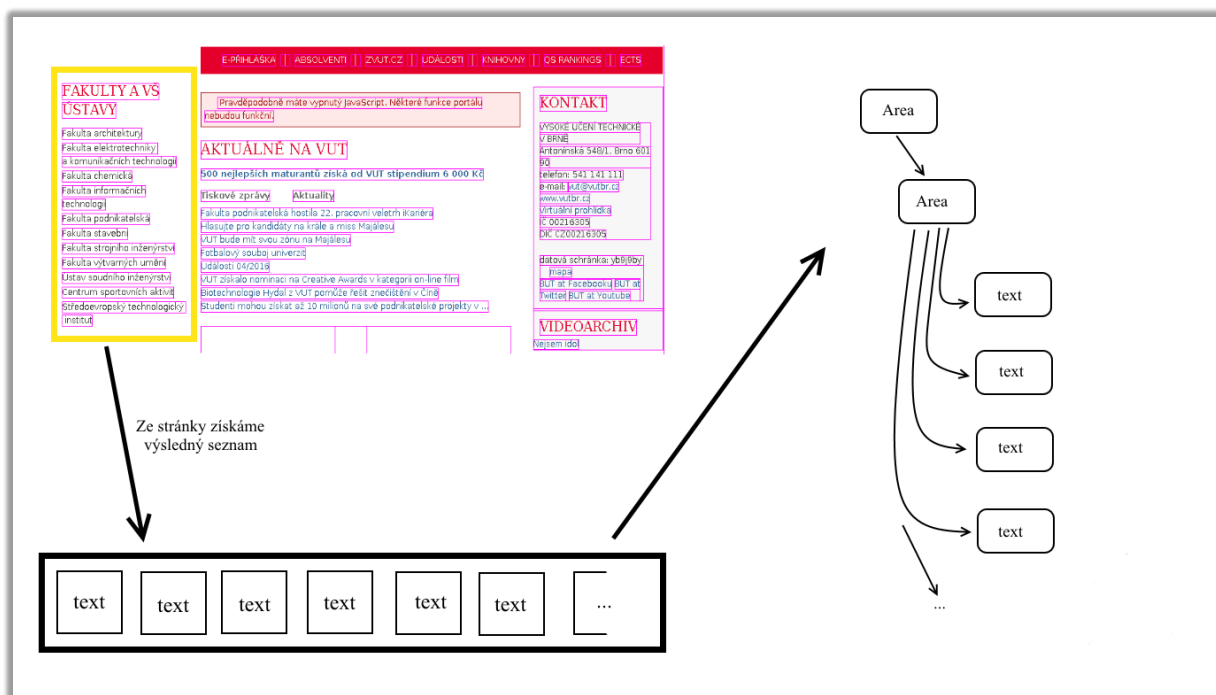
5.4 Vstup a výstup algoritmu

Nyní už víme, jakým způsobem celý algoritmus funguje a je vhodné si předvést, jak vypadá vstupní a výstupní stránka. Na vstupu máme webovou stránku upravenou CSS boxem, což je renderovací nástroj. Vstupní webová stránka se převede na *BoxTree*, který odpovídá DOM stromu vstupní stránky. *BoxTree* se poté převede na totožný *AreaTree*, který přijímá na vstup náš algoritmus. Na následujících obrázcích můžeme vidět, jak vypadají vstupní a výstupní prvky a taktéž jejich odpovídající stromy. *BoxTree* má na vstupu webovou stránku a výstupním prvkem je strom. Ukázku najdeme na obrázku 5.12, kde se nachází část webové

stránky a zeleně vyznačené nalezené prvky v *BoxTree*. Dále zde můžeme najít část *BoxTree*, který odpovídá žlutě vyznačené části. Zatímco u *AreaTree* se jedná o seznam jednotlivých prvků webové stránky, což jsou naše již nedělitelné vizuální bloky, a na výstupu opět vzniká strom. Ukázku máme na obrázku 5.13. Lze zde najít část webové stránky, na které jsou růžově vyznačené nalezené vizuální oblasti a dále seznam, ze kterého byly oblasti skládány do výsledného stromu. Ten je zde taktéž zaznačen. Můžeme si všimnout i ze stručné ukázky, že výsledný strom našeho segmentačního algoritmu je velmi plytký. Za pomoci vstupních parametrů můžeme rozložení výstupního stromu změnit. Změníme je v uživatelském rozhraní. Jakým způsobem vstupní parametry ovlivňují výstup, se dozvíme v následující kapitole.



Obrázek 5.12: Ukázka *BoxTree*, který odpovídá bloku na webové stránce.



Obrázek 5.13: Ukázka *AreaTree*, který odpovídá seznamu nalezeného ze stránky.

6 Dosažené výsledky

Po každé implementaci bychom měli ověřit její validitu. Ověřování vykonáváme za pomoci kvalitních testů. V aktuální kapitole si ukážeme, jakým způsobem jsme testování prováděli, a taktéž si ukážeme dosažené výsledky při segmentaci vstupní webové stránky. Jelikož se jedná především o experimentální odvětví, zvolení správné testovací sady je klíčové.

Seznámíme se zde se zvolenou testovací sadou. Ukážeme si výsledné výstupy po provedení našeho segmentačního algoritmu. Testování je důležité i z toho hlediska, že nám může pomoci odhalit některé chyby v implementaci. Pokud se nějaké naleznou, budou zde uvedeny i s jejich řešením.

Dalšími důležitými faktory, které velmi ovlivňují výsledek segmentace, jsou vstupní parametry. Již jsme se o nich lehce zmínili v předešlých kapitolách a jedná se o parametry *alfa* a *beta*. Jaký vliv mají na výsledek segmentace, se dozvíme v níže uvedené podkapitole 6.2.

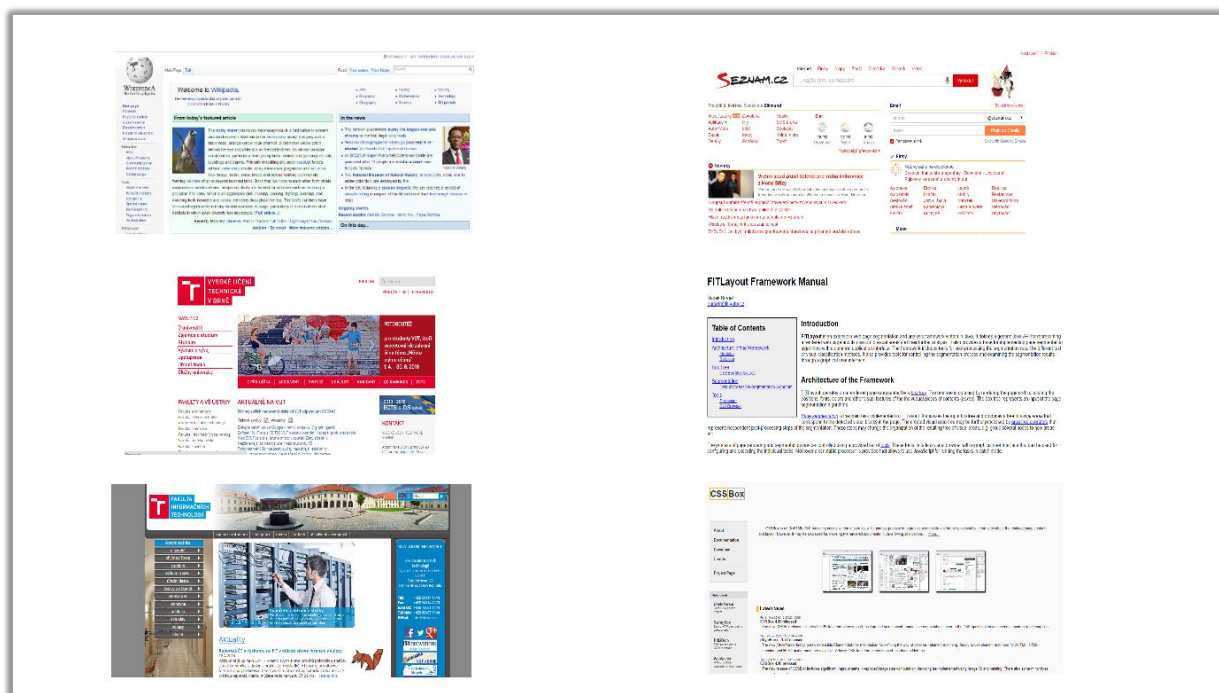
6.1 Testování

Správné a pečlivé testování je klíč k úspěchu každé implementace. Pomáhá nám ověřit jednotlivé části, a poté i kompletní část implementace. Taktéž nám napomáhá při odhalení mnoha chyb, kterých jsme mohli při implementaci dopustit.

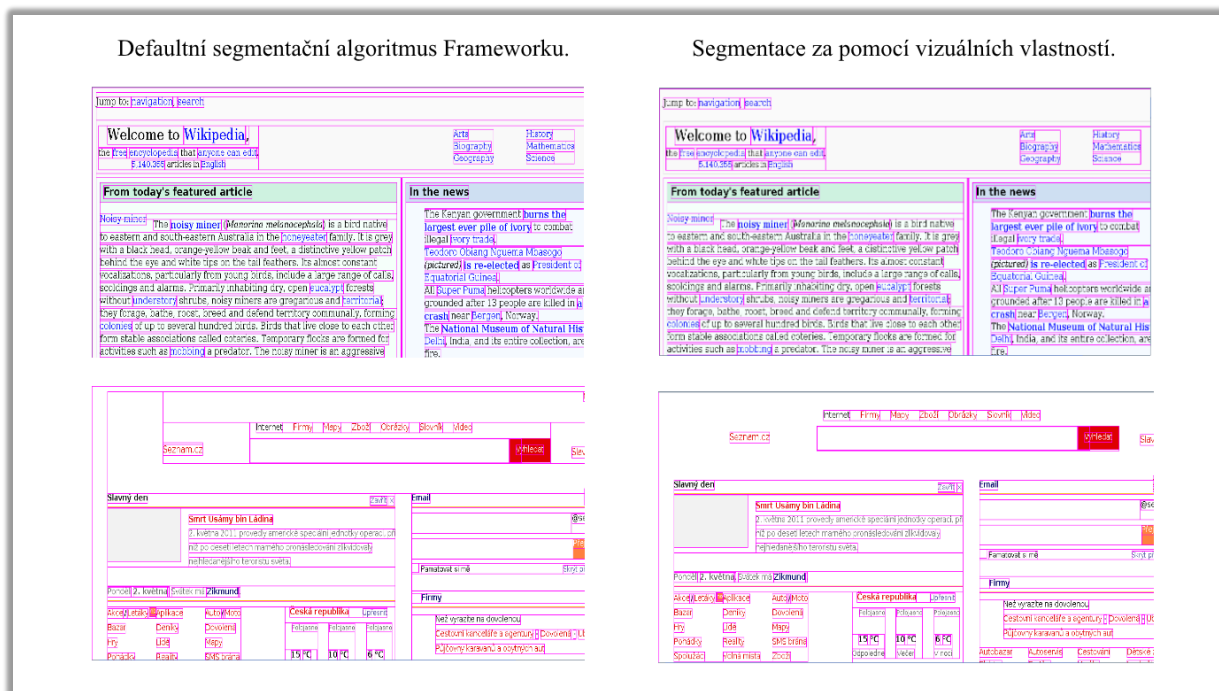
Vezmeme to z hlediska našeho algoritmu. Protože se jedná o experimentální tvorbu, musí tomu odpovídat i testy. Ty musíme volit správně a účelně, abychom dosáhli správných výsledků a odhalili chyby specifické pro náš algoritmus. Především musíme vybrat správnou sadu vybraných webových stránek. Jakmile vybereme správnou sadu, můžeme přejít k samotnému testování. To spočívá ve spouštění našeho segmentačního algoritmu s různými webovými stránkami ze vstupní sady při různé kombinaci vstupních parametrů. Jak tyto parametry ovlivňují naši segmentační metodu, se dočteme níže. Ukázku sady webových stránek můžeme vidět na obrázku 6.1.

Po otestování na vybrané sadě můžeme prozkoumat výsledky a zhodnotit tak, zda se nám podařilo správně implementovat námi popisovanou segmentační metodu. Po otestování jsme zjistili pouze drobnost, jako je například chyba v podmínce u konstrukce algoritmu podle tabulky 3.

Protože v době testování jsme neměli k dispozici další tvořené algoritmy segmentace, použili jsme pro srovnání výsledku výchozí metodu segmentace implementovanou ve Frameworku. Na obrázku 6.2 toto srovnání můžeme vidět. Pro ilustraci jsme použili stránky seznam.cz a wikipedii.org. Vidíme, že zatímco základní algoritmus, ohraničuje bloky celé, jak se nacházejí na stránce, námi implementovaný algoritmus ohraničuje bloky především podle vizuálních vlastností a rozdíl můžeme vidět již na první pohled.



Obrázek 6.1: Ukázka webových stránek použitých při testování.



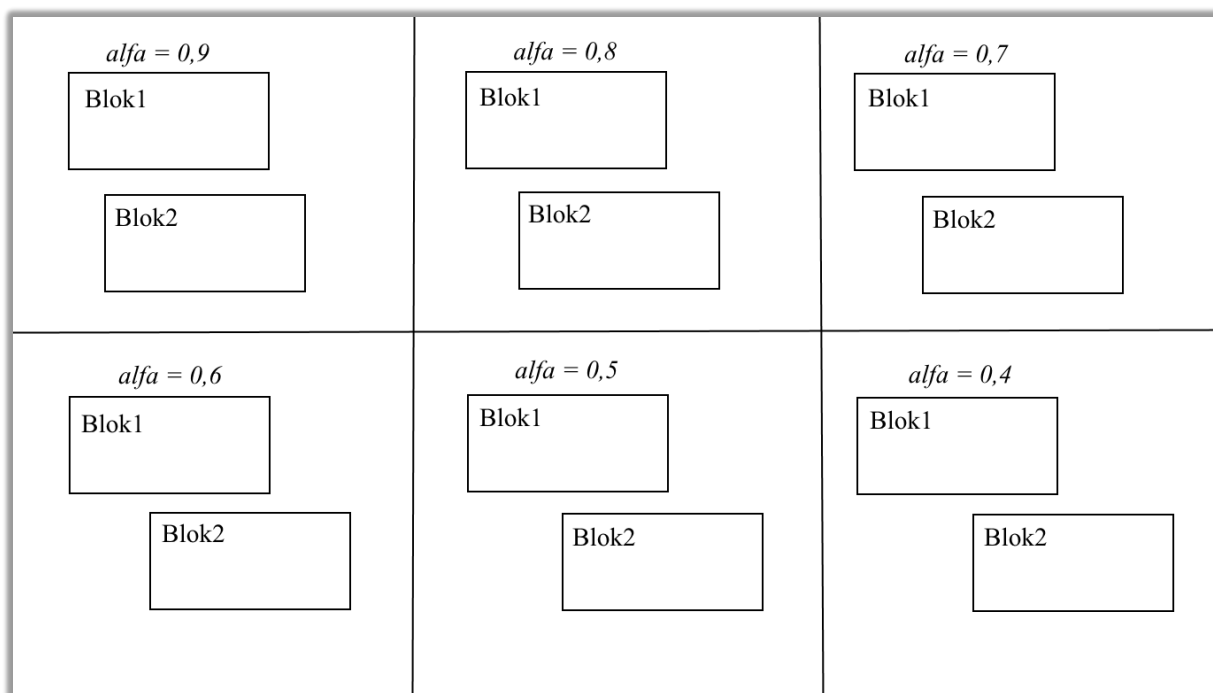
Obrázek 6.2: Srovnání segmentačních algoritmů ve Frameworku FITLayout. Nalevo vidíme výsledek segmentace defaultní segmentační metodou ve Frameworku a vpravo vidíme stejnou segmentovanou stránku za pomoci námi implementovaného algoritmu.

6.2 Změna vstupních parametrů

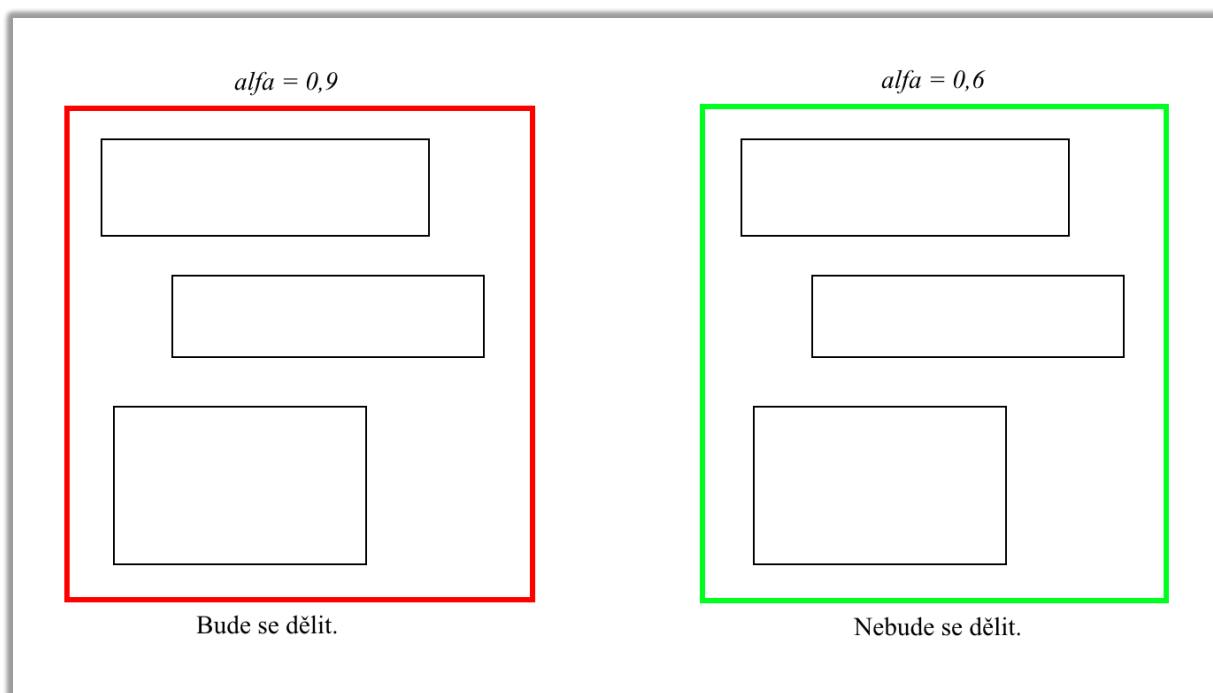
V našem algoritmu se nachází dvě důležité hodnoty, které chápeme jako vstupní parametry, ovlivňující výsledek segmentace. Jedná se o *alfu* (α) a *betu* (β). Jsou to tzv. prahové hodnoty, s kterými porovnáváme výpočty uvedené v kapitole 3. Konkrétně pro parametr α se jedná o výpočet č. 3 a parametru β patří výpočet č. 8. Při výběru naší implementace jako algoritmu, který bude ve Frameworku FITLayout provádět aktuální segmentaci, můžeme tyto parametry různě nastavit. V počátečním nastavení je $\alpha = 0.9$ a $\beta = 0.8$. Tyto hodnoty jsme empiricky zvolili podle [8].

Přejděme přímo k tomu, jakým způsobem vstupní parametry ovlivňují výsledek segmentace. Konkrétně parametr α porovnáváme s průměrným vypočítaným Seam Degree. Z toho vyplývá, že ovlivňuje porovnávání sousednosti jednotlivých bloků. Necháme-li *alfu* nastavenou na počáteční hodnotu, sousední bloky se budou muset nacházet ve větší blízkosti, aby se od sebe nedělily. Avšak jakmile hodnotu parametru *alfa* snížíme, tak se zvětší tolerance vzdálenosti sousedních bloků. Nastavení *alfy* ovlivní strukturu výsledného stromu. Bude-li se *alfa* blížit nebo bude rovna počáteční hodnotě 0.9, výsledný *AreaTree* bude plytký. Kdežto bude-li se *alfa* snižovat, tento výsledný strom bude více strukturovaný. Ostatně se můžeme podívat na obrázky č. 6.3 a č. 6.4, kde jsou příklady, co vše ovlivňuje parametr *alfa*, ukázány. Na prvním z nich můžeme spatřit, jakou polohu mohou mít vzájemně dva sousedící bloky, dle zadané *alfy*. Na druhém z nich ilustrujeme, že červeně ohraničený blok se bude muset dělit na menší bloky, protože *alfa* je větší, kdežto zelený blok se dělit nebude, poněvadž *alfa* je nižší, a je tak povolena větší tolerance odsazení při vzájemně sousedících dvou blocích. Taktéž zde máme ukázkou výsledku po segmentaci při různé změně parametru *alfa*, na konkrétní vstupní webové stránce a to na obrázku 6.5.

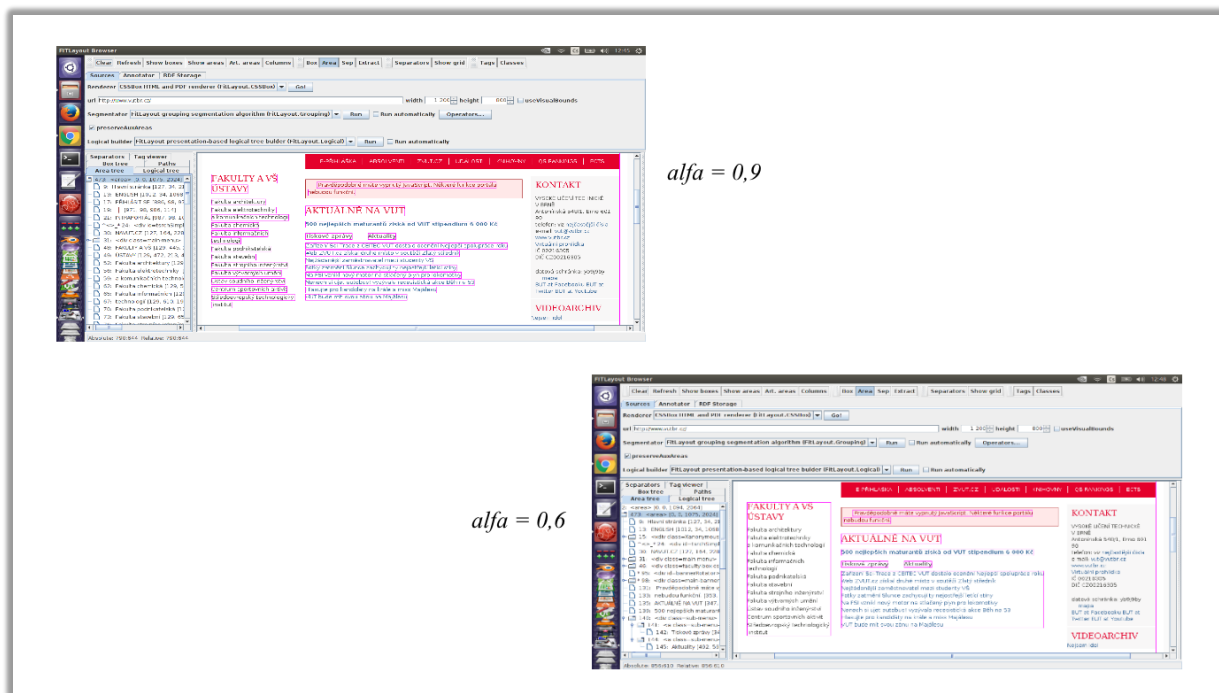
Poněvadž máme dva vstupní parametry, je vhodné si vysvětlit i druhý parametr. Hodnota parametru *beta* je porovnávána s průměrnou podobností obsahu. Na rozdíl od *alfy*, která ovlivňuje vzájemnou polohu při porovnávání dvou bloků, *beta* ovlivňuje porovnávání obsahu v daném bloku. Spouštíme-li naši segmentační metodu s nastavením *bety*, tak jak jsme si zmínili výše (prahovou hodnotou), blok se bude dělit na dílčí, již při menší různorodosti obsahu. Pokud budeme parametr *beta* snižovat, zvětší se tolerance obsahu, a blok bude moci obsahovat různé prvky, například z 80% text a zbytek obrázky. Nejvýstižněji to můžeme vysvětlit opět za pomoci obrázků a konkrétně ilustrujeme zmíněnou situaci na obrázcích č. 6.6 a 6.7. Lze zde spatřit situaci, kdy porovnáváme podobnost obsahu daného bloku, a jakou toleranci v této chvíli hraje vstupní parametr *beta*.



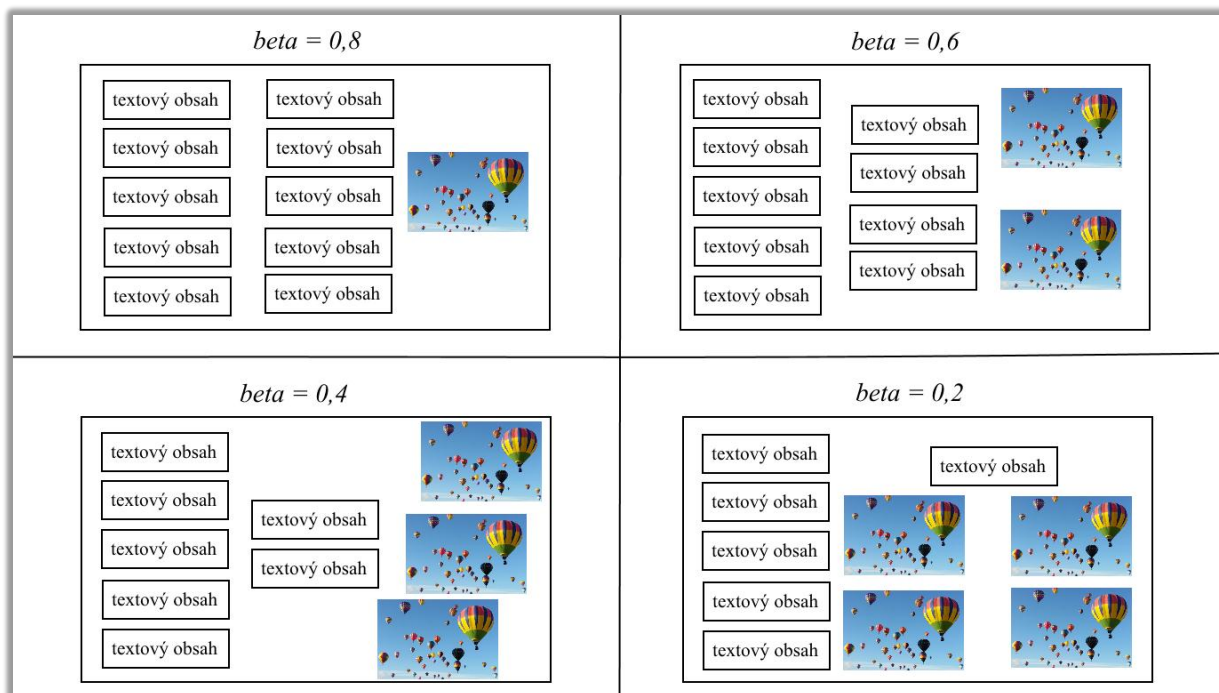
Obrázek 6.3: Ilustrace, jak α ovlivňuje toleranci, při porovnávání sousednosti dvou bloků. Všechny ilustrované bloky se nebudou dále dělit, podle zadané α fy nad nimi.



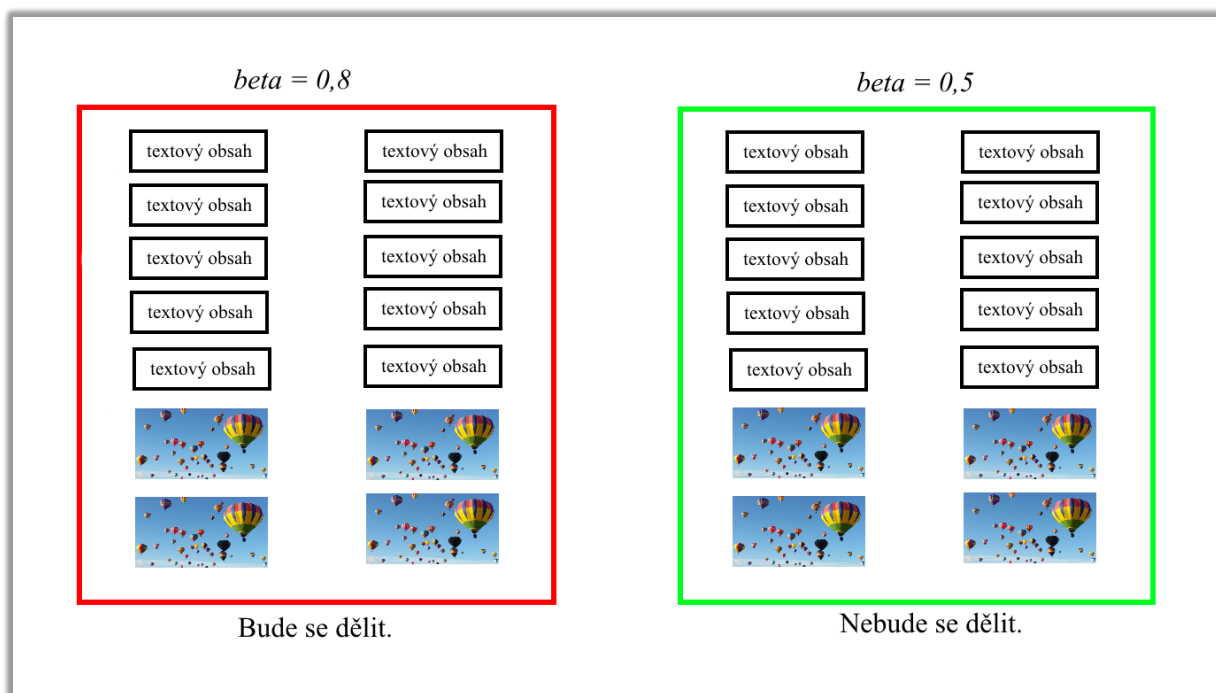
Obrázek 6.4: Ukázka bloků, které se při dané α fě budou dělit či nikoliv. Červeně je označen blok, který se bude dělit na nižší a zeleně blok, který se dělit bude.



Obrázek 6.5: Konkrétní ukázka po segmentaci námi implementovanou metodou.



Obrázek 6.6: Ilustrace, jak β ovlivňuje toleranci při porovnání obsahu v daném bloku. Všechny ilustrované bloky se nebudou dále dělit, podle zadané β nad nimi.



Obrázek 6.7: Ukázka bloků, které se při dané *betě* budou dělit či nikoliv. Červeně je označen blok, který se bude dělit na nižší a zeleně blok, který se dělit bude.

7 Závěr

V této práci jsme se seznámili se segmentací webových stránek. Uvedli jsme stručně některé existující algoritmy pro segmentaci webových stránek. Pro připomenutí to byly metody DOM-based Segmentation, VIPS a Box Clustering Segmentation.

Hlavním cílem práce bylo rozšířit Framework FITLayout o segmentační metodu. V našem případě se jednalo o metodu využívající vizuálních vlastností prvků na webových stránkách. Nejprve jsme se seznámili s algoritmem pouze teoreticky. Taktéž bylo za potřebí, seznámit se samotným Frameworkem. Podrobněji se o tom můžeme dočíst v kapitole 3. kde je teoretický popis námi vybraného algoritmu a ve 4. kapitole nalezneme seznámení s FITLayoutem.

Jakmile jsme se seznámili s potřebnou teorií a Frameworkem mohli jsme přejít na implementaci. Ta byla tvořena jako samostatný projekt, komunikující s FITLayoutem. Implementace byla dokončena v předstihu, abychom se mohli dále zaměřit na kvalitní testování a sepsání této práce.

Protože se jednalo o experimentální vývoj, museli jsme k tomu přizpůsobit i samotné testování. Vybrali jsme pro to specifickou sadu webových stránek, na kterých jsme testování provedli. Testování prokázalo správnost implementace i se spouštěním s různými parametry. Závažné problémy jsme nenalezli a to i díky včasnému odhalení již některých chyb při tvorbě implementace.

Jako nedostatek bychom mohli uvést to, že testování parametru *beta* nemohlo být dokonalé z důvodu využití výchozího renderovacího nástroje. CSSBox totiž nevykresluje obrázkový obsah, a tak nešlo vizuálně odhalit nějakou závažnější chybu. Avšak můžeme konstatovat správnou funkčnost našeho algoritmu, poněvadž jsme při implementaci dodrželi všechny teoreticky definované postupy. Tudiž můžeme říct, že zadání diplomové práce bylo splněno.

Protože segmentačních algoritmů existuje celá řada a my jsme se seznámili pouze s omezeným počtem, navázáním na tuto práci bychom mohli rozšířit Framework FITLayout o další implementace segmentačních algoritmů. Dále bychom mohli poznamenat, že Framework je kvalitní a velmi lehce rozšiřitelný nástroj. Tím pádem může být rozšiřován o celou řadu implementací a může být dále zdokonalován. Například přidáním vylepšeného renderovacího nástroje by mohl lépe fungovat i námi zvolený algoritmus.

Literatura

- [1] MILIČKA, M.; BURGET, R. Information Extraction from Web Sources based on Multi-aspect Content Analysis. In: *Semantic Web Evaluation Challenges, SemWebEval 2015 at ESWC 2015*. Portorož: Springer International Publishing, 2015, str. 81-92. ISBN 978-3-319-25517-0.
- [2] MILIČKA, M.; BURGET, R. Web Document Description Based on Ontologies. In: *Proceedings of the 2nd annual conference ICIA 2013*. Łódź: The Society of Digital Information and Wireless Communications, 2013, str. 288-293. ISBN 978-1-4673-5255-0.
- [3] BURGET, R. Layout Based Information Extraction from HTML Documents. In: *9th International Conference on Document Analysis and Recognition ICDAR 2007*. Curitiba: IEEE Computer Society, 2007, str. 624-629. ISBN 0-7695-2822-8.
- [4] BURGET, R.; RUDOLFOVÁ, I. Web Page Element Classification Based on Visual Features. In: *1st Asian Conference on Intelligent Information and Database Systems ACIIDS 2009*. Dong Hoi: IEEE Computer Society, 2009, str. 67-72. ISBN 978-0-7695-3580-7.
- [5] DENG, C.; SHIPENG, Y.; JI-RONG, W. *VIPS: a Vision-based Page Segmentation Algorithm* [online]. 2003 [cit. 2015-12-30].
Dostupné z: <https://research.microsoft.com/pubs/70027/tr-2003-79.pdf>
- [6] POPELA, T. *Implementace algoritmu pro vizuální segmentaci www stránek*. Brno, 2012. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Zelený. Dostupné z:
https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=118449
- [7] ZELENÝ, J.; BURGET R. Cluster-based Page Segmentation - a fast and precise method for web page pre-processing. In: *The Third International Conference on Web Intelligence, Mining and Semantics* [online]. Madrid: Association for Computing Machinery, 2013 [cit. 2016-04-11], str. 1-12. ISBN 978-1-4503-1850-1. Dostupné z:
<http://www.fit.vutbr.cz/research/pubs/conpa.php?file=%2Fpub%2F10252%2Fjzeleny.pdf&id=10252>
- [8] ZENG, J.; FLANAGAN, B.; HIROKAWA, S. A Web Page Segmentation Approach Using Visual Semantics. *IEICE TRANSACTIONS on Information and Systems*, 2014, str 223-230. ISSN 1745-1361.

- [9] ZENG, J.; FLANAGAN, B.; HIROKAWA, S. Layout tree-based approach for identifying visual similar blocks from web pages. In: *Computer and Information Science (ICIS), 2013 IEEE/ACIS 12th International Conference on*. IEEE, 2013. str. 66-70.
- [10] ZHANG, K.; SHASHA, D. Simple fast algorithms for the editing distance between trees and related problems. In: *SIAM Journal on Computing* [online]. 1989 [cit. 2016-04-15]. str. 1245-1262. Dostupné z: <http://dl.acm.org/citation.cfm?id=76082>
- [11] AMIT, S. Modern information retrieval: A Brief Overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 2001. str. 35-43.
- [12] DENG, C.; SHIPENG, Y.; JI-RONG, W. *Block-based Web Search* [online]. 2004 [cit. 2016-03-07]. Dostupné z: http://www.zjucadcg.cn/dengcai/VIPS/VIPS_July-2004.pdf
- [13] ZELENÝ, J.; BURGET R. *FITLayout Web Page Analysis Framework* [online]. 2014-2015 [cit. 2016-03-14]. Dostupné z: <http://www.fit.vutbr.cz/~burgetr/FITLayout/>
- [14] ZELENÝ, J. *Web page segmentation and classification* [online]. 2011 [cit. 2016-03-15]. Dostupné z: <http://www.feec.vutbr.cz/EEICT/2011/sbornik/03-Doktorske%20projekty/08-Informacni%20systemy/10-xzelen11.pdf>
- [15] BURGET, R. *CSSBox* [online]. 2007-2016 [cit. 2016-04-25]. Dostupné z: <http://cssbox.sourceforge.net/>
- [16] BURGET, R. *FITLayout Web Page Analysis Framework* [online]. 2014-2015 [cit. 2016-04-25]. Dostupné z: <http://www.fit.vutbr.cz/~burgetr/FITLayout/manual/>
- [17] HONG, J. L., SIEW, E. G., EGERTON, S. Information extraction for search engines using fast heuristic techniques. In *Journal of Data and Knowledge Engineering*, 2010. str. 169-196.

Seznam příloh

- Příloha A – obsah CD.

Obsah CD

Na přiloženém CD se nachází:

- Framework FITLayout s rozšířením o implementaci diplomové práce.
- Návod na instalaci a spuštění implementace, který se jmenuje „Čti mě“.
- Elektronická podoba diplomové práce.