VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Lukáš Kořínek

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

DETEKTOR PLAGIÁTŮ TEXTOVÝCH DOKUMENTŮ
TEXT DOCUMENT PLAGIARISM DETECTOR

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Lukáš Kořínek**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Petr Petyovský, Ph.D.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. Lukáš Kořínek ***ID:*** 195359
Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Detektor plagiátů textových dokumentů

POKYNY PRO VYPRACOVÁNÍ:

1. Nastudujte a prezentujte v současnosti používané algoritmy pro detekci plagiátů. Definujte úlohu strojového odhalování plagiátů textových dokumentů. Seznamte se s aktuálním stavem systémů automatizované detekce plagiátů používaných na VUT a na ostatních vysokých školách.
2. Navrhněte vylepšení koncepce systému automatizované detekce plagiátů používaného na VUT. Případně navrhněte nové vlastní řešení. Zaměřte se na automatizovanou detekci významných částí textu, jejich předzpracování a časově efektivní vyhodnocení míry shody.
3. Sestavte testovací databázi obsahující anotace určené pro vyhodnocení úspěšnosti detekce plagiovaných částí textu.
4. Implementujte navržený systém detekce plagiátů. Odlaďte jeho chování na testovací databázi.
5. Na rozšířené databázi demonstrujte funkčnost celého systému a definujte výpočetní požadavky pro HW řešení systému.
6. Prezentujte a diskutujte dosažené výsledky. Popište a zapracujte připomínky k fungování systému dodané zadavatelem práce.
7. Zhodnoťte funkčnost celého systému a navrhněte jeho další možná rozšíření.

DOPORUČENÁ LITERATURA:

[1] Potthast, M. : Overview of the 6th International Competition on Plagiarism Detection, [online] <http://ceur-ws.org/Vol-1180/CLEF2014wn-Pan-PotthastEt2014.pdf>.

[2] KOBATH, M. Detekce plagiátů [online]. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. 2015, <https://dspace.vutbr.cz/bitstream/handle/11012/39097/final-thesis.pdf>.

Termín zadání: 8.2.2021 ***Termín odevzdání:*** 17.5.2021

Vedoucí práce: Ing. Petr Petyovský, Ph.D.
Konzultant: Ing. Rudolf Musil

doc. Ing. Petr Fiedler, Ph.D.
předseda rady studijního programu

ABSTRAKT

Diplomová práce se zabývá rešerší metod detekce plagiátů v textových dokumentech a následným návrhem a implementací nového detektoru plagiátů, jehož primárním účelem je odhalování plagiátů v akademických pracích VUT v Brně. Vytvořené řešení aplikuje vícekrokové algoritmy předzpracování na cílové dokumenty, jejichž zpracovaná data jsou následně uložena do vlastního korpusu (báze dokumentů). Úloha hledání shod (možných plagiátů) porovnává vybraný dokument vůči zbytku korpusu, přičemž využívá paralelních výpočtů na grafické kartě. Cílem je dosáhnout co nejrychlejšího srovnání při zachování přijatelné kvality výstupu.

KLÍČOVÁ SLOVA

zpracování textu, textový korpus, paralelizace, n-gramy, CUDA, NoSQL, C++

ABSTRACT

This diploma thesis is concerned with research on available methods of plagiarism detection and then with design and implementation of such detector. Primary aim is to detect plagiarism within academic works or theses issued at BUT. The detector uses sophisticated preprocessing algorithms to store documents in its own corpus (document database). Implemented comparison algorithms are designed for parallel execution on graphical processing units and they compare a single subject document against all other documents within the corpus in the shortest time possible, enabling near real-time detection while maintaining acceptable quality of output.

KEYWORDS

text processing, text corpus, parallelism, n-grams, CUDA, NoSQL, C++

Vysázeno pomocí balíčku `thesis` verze 4.03; `http://latex.feec.vutbr.cz`

KOŘÍNEK, Lukáš. *Detektor plagiátů textových dokumentů*. Brno: Brno University of Technology, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2021, 115 s. Diplomová práce. Vedoucí práce: Ing. Petr Petyovský, PhD.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Lukáš Kořínek
VUT ID autora:	195359
Typ práce:	Diplomová práce
Akademický rok:	2020/21
Téma závěrečné práce:	Detektor plagiátů textových dokumentů

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu práce panu Ing. Petru Petyovskému, Ph.D. a také konzultantovi panu Ing. Rudolfu Musilovi za odborné vedení a náměty, které pomohly zvýšit kvalitu této práce.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Společně s rozvojem internetu došlo ke zpřístupnění podstatné části lidských znalostí a vyprodukovaných textů na webu. Lidé myšlenky na webu nejen čtou, tvoří a publikují, ale hlavně je také sdílejí mezi sebou. Sdílení textů však v některých případech ústí až ke kopírování dokumentů i myšlenek jiných autorů. Použít text napsaný někým jiným může být přínosné, ale je třeba dbát na citování autora textu, o to více, když účel díla přesahuje rámec osobních potřeb. Pokud čerpaný text není opatřen citací, pak se jedná o plagiát.

Zvláště v akademické sféře je plagiátorství považováno za jeden z nejhorších zásahů do akademické etiky a i při pozdějším zjištění může pro tvůrce plagiátu znamenat až ztrátu diplomu či kariéry. Vzniku plagiátů je třeba na vysokých školách předcházet a včas na nedostatky v citování zdrojů autora upozornit. K tomuto účelu mohou pomoci specializované softwary, detektory plagiátů. Ty dokáží vyhledat podobné texty a vyčíslit, zda a do jaké míry se nalezené texty shodují.

Je nutno poznamenat, že úloha detekce plagiátů není zcela exaktní a vždy je nutné volit kompromis mezi praktickou realizovatelností, rychlostí zpracování, kvalitou rozhodování a flexibilitou použití v různých systémech. Lze pouze doporučit, aby se automatizovaný detekční systém specializoval jen na jednu určitou oblast, neboť úloha zcela univerzálního detektoru plagiátů, jenž by fungoval na veškerých dostupných textech světových jazyků a rozhodoval dokonale v konečném čase, je současnými technickými prostředky neproveditelná. Nálezy získané prostřednictvím automatizovaného detekčního software je vždy vhodné vyhodnotit člověkem, aby se předcházelo neoprávněným obviněním na základě falešných nálezů, ke kterým jsou tyto systémy náchylné.

Navzdory uvedeným omezením vytvářejí systémy pro detekci plagiátů přínos tím, že na člověku ponechávají pouze finální rozhodnutí. Náročný a zdlouhavý proces prohledávání rozsáhlých databází textů je proveden počítačem.

Předložená práce se zabývá návrhem a vývojem nového detektoru plagiátů určeného primárně pro porovnávání vysokoškolských závěrečných prací. Jedná se o systém vytvořený nezávisle na existujícím řešení, které je nyní pro odhalování plagiátů v rámci VUT využíváno. Návrh detektoru staví na poznatcích z některých dřívějších prací pro sestavení optimálního detekčního algoritmu, avšak s úmyslem být dodán jako součást reálně nasaditelné aplikace a nikoliv pouhého prototypu.

Detektor je navržen ve spolupráci se CVIS VUT (Centrem Výpočetních a Informačních Služeb) a očekává se jeho začlenění do informačního systému, kde by mohl v budoucnu nahradit nebo vhodně doplnit stávající řešení. Hlavními požadavky na systém jsou rychlost zpracování, přiměřená kvalita detekce a použitelnost

v rámci informačního systému tak, aby bylo možné jej v dlouhodobém horizontu zpřístupnit pedagogům i studentům.

Největší výzvou pro nový software je objem zpracovávaných dat, jelikož se předpokládá velikost korpusu řádově až ve statisících dokumentech o variabilní délce, přičemž by v optimálním případě mělo být možné provést kontrolu na požádání během desítek sekund až jednotek minut. Vzhledem k této velikosti je nezbytné využít paralelních výpočtů na grafické kartě i na procesoru a efektivně pracovat s operační pamětí a dalšími přidělenými prostředky. Při takových datových objemech začínají jindy spolehlivé aspekty software narážet na své limity, např. v omezení propustnosti hardwarových sběrnic, nedostatku volné paměti nebo v maximálních velikostech datových typů i běžně využívaných datových struktur.

Teoretická část práce začíná výkladem právních norem s plagiátorstvím souvisejících, vysvětlením pojmů z oblasti plagiátorství a seznámením čtenáře s používanými citačními styly i zásadami citování. Druhá kapitola uvádí systémy etablované na českých univerzitách. Dále ve třetí kapitole následuje rešerše dostupných metod detekce plagiátů, včetně doporučení a praktických postupů použitých v předchozích pracích. Zvláštní pozornost je věnována předzpracování dokumentů, které je klíčové pro práci s velkými datovými objemy. Čtvrtá teoretická kapitola nakonec poskytuje základní přehled o softwarových prostředcích pro užití v rámci praktické části, jako jsou například NoSQL databáze.

Dokument pokračuje praktickou částí. Pátá kapitola popisuje řešení nového detektoru plagiátů. Je rozebrán návrh s popisem jednotlivých komponent i konkrétní implementační detaily na úrovni kódu aplikace. V šesté kapitole o testování jsou popsána použitá testovací data a výsledky testů z hlediska kvality nálezů i doby trvání detekčního procesu. Další, sedmá, kapitola specifikuje požadavky na provozování aplikace a poskytuje návod k jejímu sestavení. V kapitole osmé následuje seznámení s rozhraním klienta a možnostmi grafického zobrazení výsledků. Kapitola devátá stručně popisuje zapracované připomínky zadavatele. Poslední kapitola uvádí náměty na možná rozšíření, která by dále vylepšila fungování detektoru z hlediska kvalitativního i uživatelského.

1 Teoretické podklady

V kapitole budou vymezeny základní pojmy z oblasti plagiátorství, možné varianty plagiátů a zásady citování.

1.1 Autorské právo

Problematika plagiátorství souvisí s autorským právem, které upravuje zákon 121/2000 Sb. Občanského zákoníku ([1]). Zákon definuje práva autorů na jejich autorská díla (duševní vlastnictví) a vymezuje podmínky, za kterých lze díla jiných autorů užívat, rozmnožovat, rozšiřovat, pronajímat, sdělovat atd.

"Předmětem práva autorského je dílo literární a jiné dílo umělecké a dílo vědecké, které je jedinečným výsledkem tvůrčí činnosti autora a je vyjádřeno v jakékoli objektivně vnímatelné podobě včetně podoby elektronické, trvale nebo dočasně, bez ohledu na jeho rozsah, účel nebo význam (dále jen "dílo")." [1]

Zákon se tedy vztahuje na díla různého charakteru, včetně textových dokumentů vědecké a literární povahy, které jsou předmětem této práce.

1.2 Vymezení pojmů

Textový úsek Pro potřeby této práce dále myšlen jako skupina vět, odstavec, kapitola textového dokumentu nebo i celý dokument.

Plagiátorství Dle normy ČSN ISO 5127 z roku 2003 definováno jako proces, kdy autor představuje část díla jiného autora půjčeného či napodobeného jako část svého vlastního. Může se jednat o celá díla i jejich části [2]. Plagiátorství je v rozporu s autorským zákonem ([1]).

Plagiát Autorské dílo, které se navenek nijak neliší od jiných děl obdobného charakteru, vyjma faktu, že vzniklo procesem plagiátorství [2]. Jeho části byly neoprávněně převzaty z jiných zdrojů spadajících do duševního vlastnictví svých autorů.

Citace Normovaným způsobem (nejčastěji odkazem do použité literatury) přiznává duševní vlastnictví textového úseku jeho skutečnému autorovi tak, aby citovaný textový úsek nebyl plagiátem. Citovaný text by měl být viditelně oddělen od vlastního textu autora dokumentu.

Problematiku citací zmiňuje i §31 autorského zákona:

"Do práva autorského nezasahuje ten, kdo

a) užije v odůvodněné míře výňatky ze zveřejněných děl jiných autorů ve svém díle,
b) užije výňatky z díla nebo drobná celá díla pro účely kritiky nebo recenze vztahující se k takovému dílu, vědecké či odborné tvorby a takové užití bude v souladu s poctivými zvyklostmi a v rozsahu vyžadovaném konkrétním účelem,
c) užije dílo při vyučování pro ilustrační účel nebo při vědeckém výzkumu, jejichž účelem není dosažení přímého nebo nepřímého hospodářského nebo obchodního prospěchu, a nepřesáhne rozsah odpovídající sledovanému účelu;

vždy je však nutno uvést, je-li to možné, jméno autora, nejde-li o dílo anonymní , nebo jméno osoby, pod jejímž jménem se dílo uvádí na veřejnost, a dále název díla a pramen." [1]

Bibliografická citace Formální údaj o dokumentu, ze kterého autor čerpá během tvorby textu. Při citování (použití cizího výroku či textu v rámci dokumentu) uvádí bibliografická citace jednoznačný odkaz na zdroj, ze kterého text či myšlenka pochází [3]. Zdrojový text je za pomocí této bibliografické citace dohledatelný.

Doslovná citace Neboli také citát, je přímá kopie textového úseku v nezměněné podobě ze zdrojového dokumentu do cílového dokumentu. Kopie je citována a odkazuje se na příslušnou bibliografickou citaci.

Přímý plagiát Je plagiát, jenž vzniká přímou kopií textového úseku v nezměněné podobě ze zdrojového dokumentu do cílového dokumentu bez uvedení zdroje.

Parafráze Výklad jiného textu vlastními slovy. Může se jednat i o shrnutí, rozvinutí či zestručnění původní myšlenky. Parafráze musí být řádně opatřena citací, aby nebyla plagiátem. Vzhledem k použití vlastních slov může být obtížné určit, zda je daný textový úsek vlastní myšlenkou autora nebo parafrází (a tedy plagiátem v případě, kdy není úsek citován).

Kompilace Vzniká složením několika textů od různých autorů do jednoho nového textu, zpravidla bez nových poznatků. Často slouží k poskytnutí uceleného pohledu na danou problematiku [2]. Nemusí se jednat o plagiátorství, pokud jsou použité prameny uvedeny a text je jako kompilace vydán.

Autoplagiát Vznikne, pokud se autor odkazuje na jiný text, kterého je rovněž autorem, avšak bez toho, aby tento text uvedl jako zdroj. Příkladem může být použití autorova středoškolského textu jako části jeho vysokoškolské práce. K autoplagiátu může dojít nechtěným opomenutím, ale i úmyslně (snaha o vydání nového díla vytvořeného přepisem staršího) [2].

V kontextu vysokého školství je plagiátorství vážným zásahem do akademické etiky a jeho odhalení může vést kromě odsouzení ze strany autorovy almy mater či možné ztráty kariéry i k závažnějším důsledkům plynoucím z trestního zákoníku.

Autor se však plagiátorství může dopouštět i neúmyslně, a to hlavně v důsledku toho, že všeobecně známá fakta není třeba citovat. Hranice mezi tím, co je a co není v dané oblasti všeobecně známým faktem, může být subjektivní, zejména u novějších nebo vysoce odborných poznatků.

1.3 Zásady citování

Citování textových pramenů je nedílnou součástí každé rešerše a má i význam pro vědeckou komunitu. Uvedené bibliografické citace dokazují autorovo poznání daného tématu i jeho přehled v rámci písemných zdrojů z dané oblasti.

U delších textů se cituje kladením odkazů na příslušné bibliografické citace do čerpaných úseků (v místě citátů i parafrází). Texty kratšího charakteru, jako jsou články v časopisech, mohou uvádět zdroj i přímo za daný citát.

Bibliografické citace jsou nejčastěji sdružovány do samostatné sekce díla, seznamu literatury (v angličtině také *References*), která obsahuje všechny zdroje použité při psaní dotčeného textu. Zdroje jsou řazeny buďto abecedně, anebo podle pořadí, ve kterém z nich bylo v textu čerpáno. Některé citační styly umisťují bibliografické citace i do poznámek pod čarou.

Způsob citování musí být v rámci celého textu konzistentní a často platí i snaha o sjednocení na úrovně patřičné instituce, vydavatele či odvětví. Na začátku tvorby díla je třeba stanovit:

- formát (normu) pro zápis bibliografických citací;
- způsob odkazování v textu.

1.3.1 Norma ČSN ISO 690

Rozšířenou citační normou v odborné literatuře je v současnosti ČSN ISO 690 (platná od roku 2011, odvozeno z mezinárodní ISO 690:2010) ([3]). Norma definuje požadovaný zápis bibliografických citací tak, aby jednoznačně a přesně identifikovaly čerpané zdroje.

Povinné i volitelné položky každé bibliografické citace jsou dány povahou citovaného dokumentu. Je rozlišováno mezi knihami, články, elektronickými dokumenty, webovými příspěvky a stránkami, příspěvky ve sborníku a jinými. Základními údaji

jsou příjmení (hůlkovým písmem) a jména autorů, název a podnázev díla, rok vydání, počet stran, citované strany, u elektronických dokumentů také URL adresa a datum citace.

Příklad bibliografické citace podle normy ČSN ISO 690 [3]:

"EARLE, Richard. The art of cause marketing: how to use advertising to change personal behavior and public policy. Cover and interior design by Monica BAZIUK. New York: McGraw-Hill, ©2000. ISBN 0-07-138702-1."

Norma [3] umožňuje tři postupy odkazování:

Harvardský systém Odkazy se uvádí formou jména autora a roku vydání daného zdroje, vždy do kulatých závorek. Za rokem lze ještě uvést i rozsah stran výchozího textu, kterého se citace týká. Dvě a více děl tvůrců se stejným jménem a rokem vydání je třeba odlišit malým písmenem abecedy za rokem vydání. Seznam literatury je uspořádán abecedně podle příjmení autorů.

Příklad: *Systém pracuje s normou ČSN ISO 690 (KOŘÍNEK, 2021a).*

Forma číselného odkazu Bibliografické citace se číslují podle pořadí, v jakém jsou v dokumentu citovány poprvé. Tato čísla poté v textu slouží jako odkazy na danou bibliografickou citaci. Každý další odkaz na citovaný pramen již zachovává stejné číslo jako ten první. Čísla lze uvést do hranatých závorek, kulatých závorek nebo jako horní index.

Příklad: *Systém pracuje s normou ČSN ISO 690 [1].*

Průběžné poznámky Také využívají číselných odkazů a uvádění bibliografických citací v pořadí, v jakém jsou poprvé citovány. Každá citace však má nové číslo a samostatný záznam v seznamu literatury. Opakující se bibliografické citace mohou být zkráceny formou reference na jednu z předchozích. U rozsáhlých prací může být uveden i abecední seznam zdrojů.

Příklad: *Systém pracuje s normou ČSN ISO 690* [1].

V této práci je norma ISO 690 použita a využívá číselných odkazů.

1.3.2 Další citační styly

Vyjma zmíněné ČSN existují i další platné citační normy a styly, obvykle používané ve specifických oborech.

APA (American psychological association, Americká psychologická asociace) styl je užíván především v oblastech psychologie a medicíny [4].

MLA (Modern Language Association, Americká asociace moderního jazyka) má využití pro humanitní obory [4].

Chicago (Turabian/Chicago style) je často využíván v oblasti společenských věd, politologii a historických vědách. Původní manuál Chicagské univerzity byl přeformulován a zestručněn autorkou Kate Turabian, proto je tento styl známý pod dvěma různými názvy [4].

IEEE (Institute of Electrical and Electronics Engineers, Institut pro elektrotechnické a elektronické inženýrství) styl využívaný ve standardizační organizaci IEEE a nacházející uplatnění v rozličných technických oborech [5].

1.3.3 Kladení citací v textu

Doslovné citace se viditelně oddělují od autorova vlastního textu a dávají do uvozovek. Je vhodné využít i jiný typ písma, typicky kurzívu. Vždy musí být zřetelný kontext, doslovná citace je proto často umístěna za předchozí uvozovací větou. Při přejímání delších textových úseků se citát umístí do samostatného odstavce (odstavců) a odsadí od zbytku textu. Naproti tomu parafráze se od zbytku textu graficky neoddělují, mělo by však stále být zřejmé, kde převzatá myšlenka začíná a kde končí. Delší parafráze se umisťují do samostatných odstavců. Parafrázi je nutné opatřit citací [6].

Pokud je v rámci parafráze nebo v uvozovací větě citátu zmíněn autor myšlenky (příjmením i jinak), pak se uvádí citační odkaz ihned za tuto zmínku. Jestliže tomu tak není, uvede se odkaz na konec citátu či parafráze [6].

Ilustrační příklady citací (číselné indexy):

Záležitost podrobně vysvětluje Novák [1], a to následovně:

„Ve zmíněné situaci nastává jev známý jako . . . a dochází tak k přechodu dalších minoritních nosičů přes oblast prostorového náboje.“

. . .

„Fotoelektrický jev nastává při . . . , avšak pouze pokud energie dopadajícího fotonu odpovídá šířce zakázaného pásma použitého polovodiče.“ [1]

. . .

Podle Nováka [1] má energie dopadajících fotonů zásadní vliv na účinnost probíhajícího fotoelektrického jevu.

---

2 Detekce plagiátů na českých univerzitách

V následujících subkapitolách jsou stručně uvedeny systémy používané pro detekci plagiátů v českém univerzitním prostředí.

2.1 Systém používaný na VUT

VUT v Brně disponuje vlastním detektorem plagiátů, jenž je spravován na CVIS. Funkce detektoru jsou zabudovány do IS Apollo jako součást modulu závěrečných prací, kde mohou být využity ze strany pedagogů (např. vedoucím či oponentem). Závěrečné práce nahrané studenty do informačního systému jsou textově zpracovány a později porovnány na shody s ostatními pracemi z VUT.

Proces detekce plagiátů probíhá vzájemným srovnáním dokumentů na úrovni hledání podobných vět. Algoritmus je citlivý na přímé citáty i parafráze. Součástí řešení je i mechanismus pro vynechání formálních úseků textu formou ignorovaných vět.

Výstupy hledání shod jsou po jeho dokončení k dispozici v grafickém rozhraní, kde dojde k zobrazení zkoumané i nalezené podobné práce. Uživatel může nalezené shody vyhodnotit a buďto je ignorovat, nebo označit za plagiát.

Právě tento systém by mohl být v budoucnu doplněn či nahrazen novým řešením vyvinutým v rámci této práce. Doplněním je myšlena varianta, kdy budou oba systémy používány současně s tím, že jeden bude například rychlejší a druhý bude mít kvalitnější výstup.

2.2 Systémy vyvinuté Masarykovou univerzitou

Množství českých a slovenských univerzit je zapojeno do systému *Theses*[1] vyvíjeného a spravovaného Fakultou informatiky Masarykovy univerzity (FI MUNI). Tento systém slouží jako databáze pro ukládání vysokoškolských prací různé povahy (závěrečné práce, eseje, domácí úlohy). Uložené práce je poté možno zpřístupnit v rámci veřejně dostupného registru, ať už jen formou metadat nebo i plných textů. Systém zpracovává všechny běžné formáty (PDF, DOC, TXT, ...) a zprostředkovává také zálohování, archivaci nebo strojové rozpoznání textu. Součástí systému Theses je i detektor plagiátů pracující nejen napříč jeho databází dokumentů, ale i mezi dalšími repozitáři a zdroji na internetu [7].

S detektorem plagiátů a databází Theses pracuje také portál Odevzdej.cz [8], taktéž vyvinutý na FI MUNI, který zpřístupňuje detektor plagiátů i neautorizovaným

[1] `www.theses.cz`

uživatelům. Je možné do systému nahrát dokument a nechat si jej během několika dní v závislosti na dostupné výpočetní kapacitě systému zkontrolovat. Výsledky přijdou uživateli na email (pouze kvantitativní výsledky, kompletní výstup je k dispozici za poplatek) a vložená data jsou ze systému vymazána.

2.3 Zahraniční systém Turnitin

Od října 2019 využívá Univerzita Karlova kromě zmíněného systému Theses také zahraniční systém *Turnitin* vyvinutý v Kalifornii. Obdobně jako Theses umožňuje tento systém kontrolu vůči rozsáhlému korpusu dokumentů i včetně překladů z cizího jazyka [9].

Služeb Turnitin využívá přes 80 předních světových univerzit. Je podporováno 19 různých jazyků a denně je do systému odevzdáno kolem jednoho milionu prací. Opět se jedná o komplexní softwarové řešení s integrací na řadu nástrojů používaných v akademickém prostředí, které podporuje i funkce komentování, prohlížení nebo recenzování vložených prací [10, 9].

3 Metody strojové detekce plagiátů

Kapitola seznamuje čtenáře s používanými metodami strojové detekce plagiátů, a to primárně ve vysokoškolských dílech, v souladu s předmětem této práce. Je čerpáno zejména od autora Kobatha [2], na jehož přístup k detekčnímu algoritmu práce navazuje, a dále Přibila [11], který provedl v tomto směru důkladný teoretický rozbor.

3.1 Obecná detekční úloha

Odhalování plagiátů v obecných dokumentech (otevřených textech) spadá do oblasti zpracování přirozeného jazyka (natural language processing), což je široká vědní disciplína s přesahem do lingvistiky, informatiky, umělé inteligence a akustiky [11].

Je zřejmé, že obecná detekční úloha bude spočívat ve:

1. vytipování určitých úseků zkoumaného textu;
2. nalezení shod vytipovaných úseků s úseky z jiných dokumentů (srovnání);
3. vyloučení citovaných úseků z množiny nálezů;
4. posouzení nalezených shod matematickým kritériem nebo člověkem.

$$\mathrm{dfunc}(S) = \{[s, \vec{a_s}]; s \subseteq S \wedge \mathrm{crit}(s) > thr\} \tag{3.1}$$

kde:

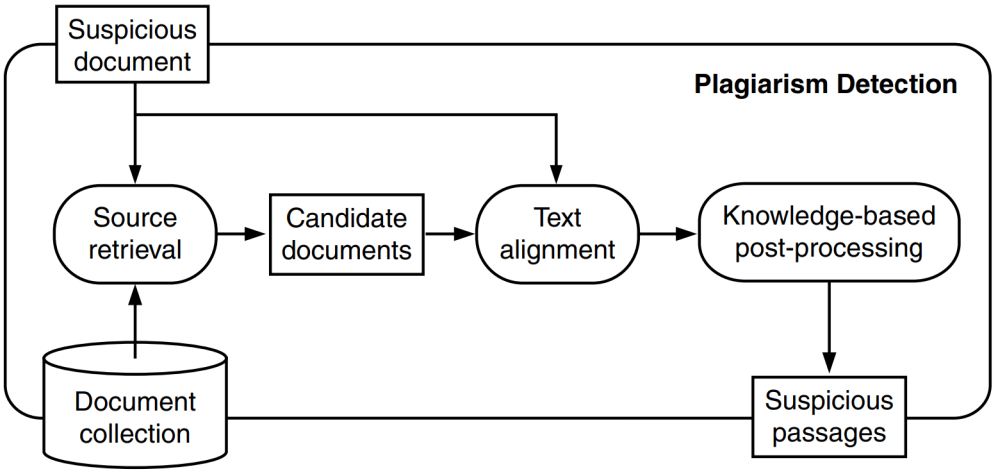
dfunc(S)	*detekční funkce nad dokumentem (řetězcem)* $\boldsymbol{S}$
s	*testovaný textový úsek (subřetězec) z dokumentu* $\boldsymbol{S}$
$\vec{a_s}$	*přidružené informace k danému nálezu*
crit(s) ..	*kriteriální funkce, např. měřící shodu* $\boldsymbol{s}$ *s jiným dokumentem*
thr	*empiricky stanovený práh, kdy* $\boldsymbol{s}$ *prohlásíme za plagiovaný*

Výstupem detekce je tedy množina textových úseků, u kterých existuje podezření z plagiátorství, a případné další přidružené informace pro každý z prvků množiny (např. zdroj plagiátu či hodnota kritéria shody).

Detekční systém nemůže být z principu zcela spolehlivý, neboť nikdy nebude možné v konečném čase zkoumaný dokument otestovat na shody oproti všem jiným existujícím dokumentům (byť těm tématicky relevantním). Stejně tak nastávají obtíže při plagiování textů, které jsou psané v jiných jazycích (autor tedy při plagiování provedl překlad, ať už strojového charakteru či vlastní). U parafrázovaných úseků se šance na nalezení shody výrazně snižuje a je závislá na povaze algoritmu i míře opisu původního textu při tvorbě parafráze.

Důležitým faktorem při hodnocení výstupu detektoru je výskyt falešných nálezů (false positives), např. vlivem nerozpoznání, že některá z citací se na daný textový úsek vztahuje. Nalezený plagiát také může být neúmyslný a nevědomý v případě, kdy se jiný autor zabýval podobným tématem a uvedl obdobné myšlenky a závěry. Po vyhodnocení případných matematických kritérií je vždy na místě nechat nálezy prověřit člověkem.

Definice obecné úlohy podle autora Potthasta [12] je zobrazena na obrázku níže. V překladu probíhá zleva - sběr dat zkoumaného dokumentu a dostupných dokumentů ke srovnání, vyhodnocení podobných dokumentů, srovnání textů zkoumaného a podobných dokumentů, vyhodnocení znalostními technikami, výstup v podobě podezřelých pasáží.



Obr. 3.1: Obecná úloha detekce plagiátů dle [12].

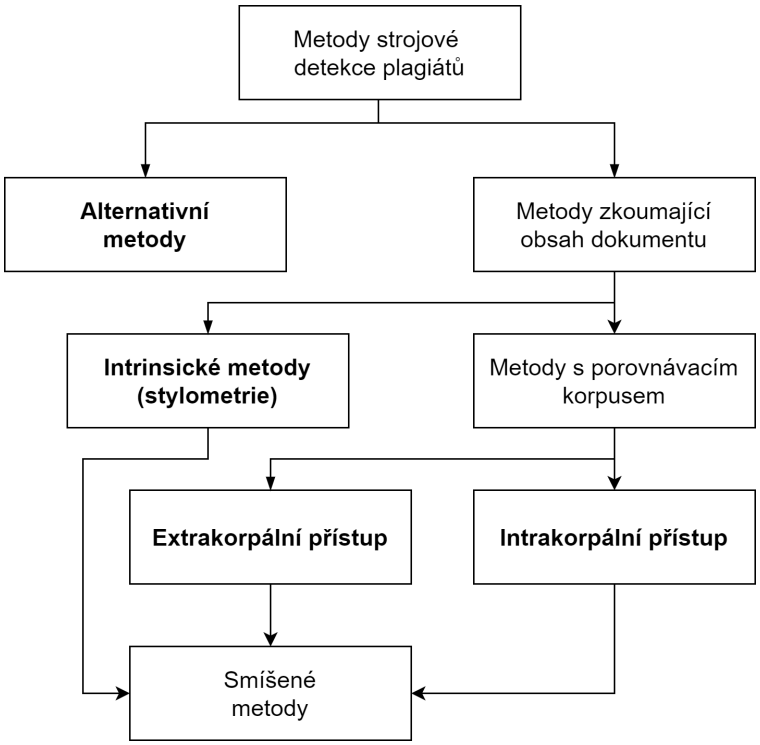
3.2 Varianty detekčních systémů

Pro detekci plagiátů lze vycházet ze čtyř různých přístupů založených na zkoumání obsahu analyzovaného dokumentu [11]:

- intrakorpální přístup;
- extrakorpální přístup;
- intrinsický přístup;
- smíšené/kombinované přístupy.

Mimo zkoumání samotného obsahu dokumentu je za některých okolností možné použít i alternativních metod, jako je neviditelné značkování (při vkládání textu

se ze zdrojového dokumentu okopíruje i skrytá značka) anebo ponechání autora vlastnoručně doplnit vynechanou část jeho textu (předpokládá se, že autor vyplní text přibližně stejnou informací a slovní zásobou, jako když práci psal). Více o těchto technikách ve zdroji [2].



Obr. 3.2: Běžné přístupy používané pro strojovou detekci plagiátů.

3.2.1 Systémy intrakorpální

Intrakorpální (intra corpal) systémy předpokládají existenci korpusu, tedy databáze dokumentů, se kterými bude analyzovaný dokument porovnán při hledání shod. Zásadní nevýhodou tohoto přístupu je nemožnost nalézt plagiáty převzaté ze zdrojů, jež nejsou do korpusu zařazeny. Tento problém se snižuje s rostoucím počtem obsažených textů [11], závisí však také na zdrojích, ze kterých jsou nové dokumenty do korpusu přidávány. Ani databáze všech česky psaných akademických prací neodhalí skutečnost, že autor okopíroval první pasáž textu ze serveru Wikipedia.org, na kterou byl odkázán webovým vyhledávačem.

Korpus lze sestavit postupnou analýzou dokumentů, kdy je nejprve každý z nich předzpracován předepsaným postupem a po dokončení přidán mezi ostatní dokumenty v korpusu (uložen do databáze a zaindexován).

Vzhledem k omezené velikosti korpusu a zpravidla i možnosti si korpus předzpracovat do vhodné podoby a formátu, může být běh detekčního procesu této metody poměrně rychlý oproti ostatním přístupům. Je sice nutné počítat s časem potřebným pro indexaci dokumentů, tu však stačí pro každý dokument provést jen jednou.

Faktory klíčové pro intrakorpální detekci jsou podle [11] následující:

- způsob uložení dokumentů v korpusu a míra jejich předzpracování;
- výběr algoritmu hledajícího podezřelé (podobné) dokumenty v korpusu;
- výběr algoritmu vyhledávající konkrétní plagiované úseky textu.

Pokud se nepodaří vybrat z korpusu ty dokumenty, u kterých lze počítat jako s možným zdrojem plagiátů, je možné analyzovaný dokument porovnat se všemi ostatními dokumenty v korpusu [2]. Možnost provést takovéto prohledávání se poté odvíjí od velikosti korpusu, náročnosti použité metody a dostupného hardware.

3.2.2 Systémy extrakorpální

Extrakorpální metoda je založena na neexistenci vlastního korpusu. Dokumenty, oproti kterým je analyzovaný dokument porovnáván, jsou čerpány z nástrojů třetích stran, jako jsou internetové vyhledávače a katalogy knihoven. Ačkoliv se systém jeví být extrakorpálním, tak korpus v nějaké formě vždy existuje, je však spravován stranou, odkud jsou data čerpána [11]. Je vhodné proto volit takové datové zdroje, které mají k dispozici co největší korpus (co největší počet zaindexovaných stránek v případě internetového vyhledávače).

Při praktickém použití není možné porovnávat podezřelý dokument s každým dokumentem v databázi/korpusu, ale je nutné vybrat jen určitou část dokumentů, u kterých má smysl je porovnávat [2]. Strategie výběru nálezů k porovnání je poté jedním z klíčových aspektů úspěšnosti extrakorpálního systému [11]. Například u výsledků získaných z internetového vyhledávače se dá stanovit, že budou testovány pouze první tři nálezy získané na základě analýzy klíčových slov zkoumaného dokumentu.

Při použití internetových vyhledávačů bude systém typicky fungovat na principu generování velkého množství dotazů na daný vyhledávač, a sice ve snaze najít shodu s vybranými krátkými úseky textu (řádově jednotky slov). Kvalita a funkčnost extrakorpálního systému je pak zcela odkázána na možnosti poskytovatele dat. Přenos dat po síti dále zpomaluje funkci systému a může jej činit až řádově pomalejším a méně spolehlivým oproti přímému přístupu do vlastního korpusu [11].

V praxi bývají největším problémem podmínky využití služeb těchto vyhledávačů, kdy některé servery umožní pro nekomerční účely pouze položení několika tisíc

dotazů denně[1]. Výhodou je také, když vyhledávač umožňuje vyhledávat pomocí webového API[2] [11], aby se předešlo nutnosti provádět náročnou operaci zpracování HTML dokumentů již u výsledků hledání. Nicméně u nalezených zdrojů je nakonec stejně nutné získané HTML zpracovat.

Při hledání shod nad anglicky psanými dokumenty se jako možná alternativa k tradičním vyhledávačům nabízí i databáze přes půl miliardy webových stránek zvaná **ClueWeb**[3]. K přístupu do ClueWeb je možné použít i vyhledávací systém **Chat Noir**[4] [12]. Kromě absence českých textů je však nevýhodou i zastaralost databáze, jelikož její sběr byl ukončen roku 2012.

Každý text nalezený vyhledávačem je nutné před hledáním shod předzpracovat do podoby vhodné k porovnání. Je na místě, aby již nalezené a zpracované zdroje byly ukládány do dočasné paměti (cache) a při dalším nálezu byly pouze vyvolány z této cache. Další možností je zvolit smíšený přístup a z takhle zpracovaných dokumentů sestavovat vlastní korpus.

3.2.3 Systémy intrinsické

Intrinsický přístup se od předchozích odlišuje tím, že nevyžaduje žádný korpus. Namísto porovnávání textových úseků analyzovaného dokumentu s jinými dokumenty se zaměřuje na stylistické vlastnosti textu (stylometrie). Předpokládá se, že plagiovaný úsek bude mít lingvistické charakterstiky odlišné od autorovy vlastní práce. Původní autor plagiovaného úseku má totiž své vlastní vyjadřování, formátování i efektivní slovní zásobu [11].

Zjevnou nevýhodou tohoto přístupu je však nemožnost určit zdroj, ze kterého bylo čerpáno. Zároveň může často docházet k falešným nálezům, jelikož autor mohl část textu napsat například ve spěchu a již neprovedl následnou revizi a korekturu [11].

Mezi zkoumané lingvistické charakteristiky patří například [11]:

- relativní frekvence podstatných nebo přídavných jmen;
- průměrná délka vět a souvětí;
- použitá slovní zásoba a fráze.

[1] U vyhledávače Google, s největším potenciálním korpusem, se nepodařilo v době psaní práce nalézt informaci o kvótě pro provádění běžných dotazů na www.google.com, které vrací výsledky ve formátu HTML. Existuje však služba Google Custom Search API určená převážně pro vlastní vyhledávání v rámci malého webu. Tato služba umožňuje zdarma pouze 100 dotazů denně.

[2] Application Programming Interface, je formalismus nad protokolem HTTP/HTTPS sloužící ke vzájemné komunikaci mezi webovými službami v předepsaném formátu.

[3] `https://lemurproject.org/clueweb12/`

[4] `https://www.chatnoir.eu/`

Intrinsické metody se v praxi spíše než pro detekci plagiátů používají například pro zjištění autorství či datace neznámých děl [11]. Díky relativní rychlosti zpracování je vhodné použití intrinsických metod ve smíšených systémech, kde mohou sloužit jako pomocná kritéria před anebo po využití korpálních metod [2].

Základní intrinsický algoritmus může dle [13] pracovat na principu posuvného okna, kdy se:

1. určí lingvistické charakteristiky pro celý dokument;
2. stanoví vhodná velikost posuvného okna;
3. určí lingvistické charakteristiky pro každý úsek daný posouvaným oknem;
4. porovnají charakteristiky pro jednotlivá okna s charakteristikou dokumentu.

Okna, jejichž charakteristiky se významně odlišují od charakteristiky celého dokumentu, nasvědčují odlišnost autora, a tedy plagiovaný úsek. Právě vhodná volba velikosti posuvného okna je pro intrinsický detektor klíčová, jelikož není apriori známa informace o poloze a délce plagiovaných úseků. Alternativou je i srovnání získaných oken navzájem vůči sobě, jelikož dokument s velkým obsahem plagiátů může mít znatelně zkreslenou referenční hodnotu [13].

Na základě praktických experimentů na anglicky psaných ručně plagiovaných textech autoři [13] doporučují velikost okna 100 slov (bez předchozího odstranění stop-slov[5]) nebo 200 slov (s odstraněnými stop-slovy).

3.3 Předzpracování dokumentů

Předzpracování dokumentů (ať už před vložením do databázového korpusu nebo za běhu extrakorpální metody) je jedním z klíčových faktorů pro úspěšnou detekci plagiátů v analyzovaném dokumentu. Kvalita a způsob předzpracování ovlivňuje aspekty rychlosti i správnosti rozhodování detekčního systému. V následujících podkapitolách budou podrobněji představeny možné techniky.

3.3.1 Převod znakové sady

Znaky digitalizovaného textu jsou určeny svým celočíselným kódem, jehož hodnota je dána použitou znakovou sadou. Nejčastější znakové sady používané v českojazyčných dokumentech jsou *UTF-8* (zejména na webu, dle standardu *Unicode*[6]) a *ISO-8859-2* (středoevropská). Aby bylo možné dva dokumenty mezi sebou porovnat, je nutné

[5]Definice stop-slov v kapitole 3.3.5.

[6]Vyvíjeno mezinárodním konsorciem organizací za účelem vytvoření jednotné znakové sady, která pojme znaky všech používaných jazyků, např. včetně těch čínských. Znaky jsou kódovány do 32bitového rozsahu.

je nejprve převést do stejné znakové sady, aby dvě stejná slova byla v obou dokumentech vyjádřena stejnou sekvencí číselných kódů.

Jistou výhodu má v tomto ohledu angličtina, jelikož neakcentované znaky velké a malé abecedy (znaky ASCII tabulky[7]) jsou v drtivé většině znakových sad shodné. Oproti tomu čeština obsahuje celkem patnáct akcentovaných znaků[8], jejichž kódování se i ve dvou zmíněných znakových sadách liší.

Autoři plagiovaných textů mohou cíleně znakovou sadu měnit, aby zamezili svému dopadení [2]. Proces převodu znakové sady může dále ztížit fakt, že ne u všech dokumentů je informace o použité znakové sadě uvedena.

3.3.2 Syntaktická analýza textu (parsing)

Textové dokumenty zpravidla nejsou uloženy ve formě prostého textu, ale v jednom ze standardizovaných formátů, prostřednictvím kterých jsou k textu přidružena dodatečná metadata, informace o formátování, grafické prvky a jiné.

Pro další použití v rámci detekčního systému je třeba znaky převést do formy prostého textu, ze kterého budou odstraněny všechny speciální značky pro daný formát, obrázky, grafy a další nežádoucí elementy.

Nejpoužívanější formáty pro ukládání a přenos dokumentů jsou:

PDF Formát vytvořený společností Adobe primárně pro účely tisku a webového přenosu dokumentů. Není ve výchozím stavu určen pro přímou editaci, a proto se využívá k distribuci dokumentů, u kterých není žádoucí, aby je příjemce upravoval.

DOCX Formáty pro dokumenty vytvořené v aplikaci Microsoft Word a obdobných pokročilých textových editorech. Používá kompresních algoritmů a stromové struktury s XML[9] značkováním.

HTML Standardní formát pro webové stránky. Existuje ve vícero standardech, z nichž nejnovějším je HTML5. Vychází z XML a je s ním z velké části syntakticky kompatibilní.

Čistý text Nejčastěji s příponou *.txt*. Obsahuje pouze textové znaky bez jakéhokoliv formátování a může být přímo použit v detekčním systému.

[7](American Standard Code for Information Interchange) Tabulka kódující znaky anglické abecedy, číslice, základní interpunkci a některé speciální znaky do 7bitového celočíselného rozsahu. Tabulka se stala historicky základem pro různé novější znakové sady, které však využívají větších rozsahů pro zakódování většího množství znaků.

[8]Konkrétně á,é,ě,í,ý,ó,ú,ů,š,č,ř,ž,ď,ň,ť.

[9](Extensible markup language) Značkovací jazyk pro popis libovolných datových struktur za pomocí stromu elementů a atributů k těmto elementům náležejícím.

Proces syntaktické analýzy je často výpočetně náročný, a to právě kvůli tomu, že užívané formáty používají komplexní stromové struktury, kompresní algoritmy, mohou být kódovány za pomocí kryptografických klíčů atd.

Jelikož se jedná o častý problém počítačových věd, existuje řada knihoven, které umí optimalizovaným způsobem výše zmíněné formáty na prostý text převádět. Knihovny je však třeba používat opatrně, jelikož mohou mít problémy s převodem českých znaků [2].

3.3.3 Odstranění formálních částí a citací

Ve specifických typech dokumentů, jako jsou akademické práce, normy a vědecké články, se vyskytují části, které není žádoucí při detekčním procesu brát v potaz.

Patří mezi ně [2]:

- standardní části dokumentu, např. titulní strana nebo zadání práce;
- části bez významných informací (obsah, seznam obrázků);
- jasně rozeznatelný citovaný text.

Popsané textové úseky je třeba před detekcí plagiátů odstranit, neboť by mohly být označeny za plagiované, ačkoliv je zřejmé, že k žádnému plagiátorství nedochází. Při odstraňování citací je na místě opatrnost, aby byla vždy správně určena část textu náležející k dané citaci [2]. Je lepší zvolit pesimistický přístup, kdy se předpokládá, že se daná citace váže spíše k menšímu, nežli většímu textovému úseku.

Odstraněním formálních a citovaných úseků zároveň dojde ke zmenšení velikosti analyzovaného dokumentu, a tím pádem i ke snížení náročnosti následného procesu hledání shod [2].

3.3.4 Tokenizace

V rámci tokenizace je text rozdělen na jednotlivá slova (tokeny), ze kterých je odstraněna veškerá interpunkce, číselné údaje, převedena velká písmena na malá, odebrány tzv. bílé znaky (mezery, konce řádků, tabulátory) tak, aby text měl charakter jedné dlouhé věty [11].

Převodem velkých písmen na malá dojde k urychlení a zkvalitnění detekce, avšak za cenu toho, že systém nebude rozlišovat mezi běžnými slovy a zkratkami názvů [2]. Možným řešením je nepřevádět velká písmena na malá u těch slov, která jsou psána pouze velkými písmeny, a tedy je nelze považovat za zkratku.

Zároveň při procesu tokenizace dochází ke ztrátě některých údajů důležitých pro intrinsickou detekci, proto je vhodné tento krok v takovém případě omezit či vynechat [2].

3.3.5 Filtrace

V rámci filtrace se z textu odstraní tzv. stop-slova. Jedná se o v daném jazyce často opakující se slova, která postrádají přínos pro detekční proces a zbytečně zvětšují daný dokument. V některých případech by kombinace stop-slov mohla vést i k falešným nálezům. Během výzkumu provedeného v práci [11] odstranění stop-slov znatelně navýšilo podobnostní metriky (udávající míru podobnosti dokumentů) na testovaném korpusu složeném z akademických prací.

V češtině se jedná o předložky, spojky, zájmena nebo i některá přídavná jména. Jako vhodný zdroj stop-slov se jeví frekvenční lingvistická analýza většího textového vzorku. Podle provedených analýz z 80. let minulého století mezi nejpoužívanější slova v českém jazyce patří: „a“, „že“, „v“, „na“, „s“, „z“, „být“ (ve všech tvarech), „on“, „ten“ a „který“ [11].

3.3.6 Lemmatizace

„Lemma“ v lingvistickém oboru znamená základní podobu slova či fráze. Jedná se o proces nahrazení slov v dokumentu jejich výchozími slovníkovými tvary, nejčastěji 1. pádem jednotného čísla u podstatných jmen a infinitivem u sloves. Cílem je snížit počet unikátních slov v textu tak, aby bylo snazší odhalit parafráze a zároveň snížit náročnost výpočtu [11] (to už závisí na použité metodě).

U některých jazyků lze lemmatizovat algoritmicky, pro češtinu je však vhodnější užít slovníkových metod (vzhledem k množství tvarů slov a jejich nepravidelnostem).

3.3.7 Vážení

Pro potřeby výpočtů podobnostních metrik přiřazuje jednotlivým slovům určitou váhu, zpravidla dle míry jejich odbornosti a četnosti výskytu. Váhy lze alokovat na základě principů stanovených pro úroveň jednoho dokumentu, globálně celého korpusu i kombinovanými postupy [11].

3.3.8 Synonymizace

Při tvorbě parafráze mohou být určitá slova vhodně zaměněna svými synonymy či antonymy, což při detekci způsobí nenalezení shody.

Tento nedostatek je možné potlačit, pokud je pro požadovaný jazyk k dispozici slovník synonym a případně antonym. Lze projít všechny tokeny textu a nahradit je za vybraný tvar ze slovníku (např. první tvar v pořadí).

Podobně jako v případě lemmatizace dochází synonymizací ke snížení počtu unikátních slov, a tedy i k pravděpodobnému zrychlení hledání shod. Potlačení synonym/antonym má smysl pouze v případě, kdy je aplikováno na celý korpus. Použití

pouze na analyzovaný dokument by naopak mohlo být kontraproduktivní, jelikož by zvýšilo rozdíl mezi plagiovaným úsekem a zdrojem plagiátu.

Kromě pravděpodobné nedostupnosti slovníku ve formátu vhodném pro použití v rámci detekčního systému vyvstává problém, kdy určitá synonyma dávají smysl (a tudíž mají význam shodný s vybraným tvarem) jen ve specifickém kontextu.

Příklad použití synonymizace:

„Statný hřebec poklusával v ohradě.“ - se předzpracuje s lemmatizací
„statný hřebec poklusávat ohrada“ - provede se synonymizace
„velký kůň běhat plot“ - synonymizovaný text

A dále na parafrázi:

„Obrovský oř běhal po výběhu.“
„obrovský oř běhat výběh“
„velký kůň běhat plot“

Je zjevné, že v ilustrovaném případě by tento postup pomohl při odhalení plagiátu.

3.3.9 Strojový překlad

Autor plagiátu zběhlý v jazycích může se zanedbatelnou námahou parafrázovat dokument napsaný v cizím jazyce, kvůli čemuž bude problém takový plagiát odhalit pomocí přístupů intrinsických i korpálních.

Aby detekční systém dokázal najít plagiáty i pro zdroje v cizích jazycích, je možné analyzovaný dokument nejprve přeložit do cílového jazyka (nejčastěji do angličtiny) a následně jej již porovnávat, jako by byl v daném jazyce napsaný (porovnat jej s anglickými dokumenty v interním korpusu nebo extrakorpálně).

Praktická použitelnost tohoto úkonu předzpracování je vzhledem k aktuálním možnostem strojového překladu a spontánnímu vzniku parafrází během překladu omezená. Pro získání kvalitnějších překladů je vhodné využít webových služeb, u kterých lze však stejně jako u extrakorpálních metod narazit na finanční a jiné limity vyplývající z podmínek jejich používání.

Strojový překlad jde použít i zároveň se synonymizací, kdy se nahradí vybraným tvarem nejen slova z českého jazyka, ale i z jiných jazyků (mnohojazyčná synonymizace) [11].

3.3.10 Převod dokumentu na vektor n-gramů

V závislosti na použité detekční metodě a povaze dokumentového skladu (korpusu) může být výhodné dokument reprezentovat pomocí vektoru tzv. n-gramů. N-gramy

jsou sekvence slov o právě n prvcích. Každý dokument o délce m slov je rozdělen na $(m - n + 1)$ n-gramů, kde platí $m \gg n$ [11].

Hodnota n se volí tak, aby nebyla příliš nízká (falešné nálezy), ani příliš vysoká (nenalezení shod tam, kde jsou očekávány). Vhodné hodnoty parametru n jsou podle rešerše v práci [2] hodnoty nízké, a to $\{2, 3\}$.

Je však nutné mít na paměti, že čeština obsahuje velké množství dvouslovných a tříslovných slovních spojení[10] bez významu pro detekci, které mohou vést na falešné nálezy [11].

Příklad tvorby n-gramů o $n = 3$ pro větu:

„Řízení navržené metodou Ziegler-Nichols vyniká při regulaci poruch.“

Vzniká vektor (po předchozí tokenizaci):

$\vec{u} = ($
{ řízení, navržené, metodou };
{ navržené, metodou, zieglernichols };
{ metodou, zieglernichols, vyniká };
{ zieglernichols, vyniká, při };
{ vyniká, při, regulaci };
{ při, regulaci, poruch };
)

Takto vytvořený vektor n-gramů lze posléze použít ke hledání shod v textech.

Vyhotovené n-gramy mohou sloužit i přímo jako formát, ve kterém bude dokument uložen do korpusu, a to namísto ukládání předzpracované fulltextové[11] podoby dokumentu. Formát uložení n-gramů má poté vliv na rychlost běhu algoritmu hledání shod a množství potřebné paměti [11].

Základní způsoby uložení n-gramového vektoru jsou [11]:

Fulltextový formát Hodnoty n-gramů se ukládají v přímé (slovní) podobě vhodné pro danou databázi. Tento způsob klade největší nároky na hardware použitý ke hledání shod.

Alfanumerické formáty Slova v n-gramech jsou zkrácena na několik písmen doplněných o numerický údaj pro zajištění unikátnosti n-gramu. Například pro n-gram { řízení, navržené, metodou } vzniká řetězec „riname15“. Zkrácení řetězců umožní zrychlení detekčního algoritmu.

[10] Jako například „a tak dále“, „v současné době“, „na tomto místě“.

[11] Text se uloží „tak jak je“, tedy v podobě jedné dlouhé věty.

Numerické formáty Unikátním n-gramům se přidělují unikátní celočíselné identifikátory. Úloha se tím mění z porovnávání řetězců na porovnávání číselných hodnot, čímž dojde k dalšímu znatelnému urychlení oproti alfanumerickému formátu. Pro zajištění vyhledatelnosti n-gramu bez převodní tabulky je vhodné číselné hodnoty přiřazovat tak, aby je bylo možné ze slovního vyjádření vypočíst (např. použitím hashovacích funkcí[12]).

3.4 Kritéria podobnosti a shody

Další funkcí detekčního software po předzpracování dokumentů je výběr vhodných dokumentů ke srovnání a následné stanovení matematických kritérií, pomocí nichž se dá kvantitativně posoudit, do jaké míry se zkoumané dokumenty shodují.

3.4.1 Podobnostní kritéria

„Podobnost dvou dokumentů vyjadřuje míru obsahové (tématické) příbuznosti obou dokumentů.“ [11]

Kritéria založená na podobnosti slouží pro vytipování těch dokumentů či specifických textových úseků, u kterých má další smysl použití výpočetně náročnějších metod pro detekci plagiátů. Neměla by však sloužit jako výsledek, na základě kterého je daný textový úsek za plagiát prohlášen.

Extrakce klíčových slov

Základní metodou určení podobnosti dokumentů je porovnání jejich klíčových slov. Tento údaj je většinou snadno dostupný i v případě korpusů získaných od třetích stran (extrakorpální metody).

Nejsou-li klíčová slova pro daný dokument přímo k dispozici, lze je z dokumentu extrahovat. Nejčastěji se provádí extrakce statistickými metodami, kdy například dochází k výběru stanoveného procenta nejvíce se vyskytujících podstatných jmen v daném dokumentu [11].

Na získané vektory klíčových slov lze aplikovat libovolnou metriku. Typicky jde o podíl klíčových slov společných pro oba dokumenty vůči celkovému počtu unikátních klíčových slov z těchto dokumentů [11].

[12] Hashovací funkce fungují jako jednosměrná šifra, která převede vstupní řetězec na jiný (číselný) dle použitého algoritmu.

Kosinová podobnost

(Cosine similarity) Pracuje na základě četnosti výskytu unikátních slov v obou porovnávaných dokumentech. Měří, kolikrát je jedno a to samé slovo ve srovnávaných textech použito [11]. Dá se použít nejen na celé dokumenty, ale i libovolně krátké úseky textu.

Postup získání kosinové podobnosti pro textové úseky A a B sestává z [11]:

1. určení pomocného vektoru $\vec{c}$, sestávajícího z unikátních slov v obou textech (v libovolném pořadí), vektor bude mít délku n;
2. určení vektorů $\vec{a}$ a $\vec{b}$, jejichž hodnotami jsou počty výskytů slov z $\vec{c}$ v textových úsecích A a B;
3. určení kosinu úhlu svíraného vektory $\vec{a}$ a $\vec{b}$.

Vzorec pro určení kosinu úhlu [2]:

$$\text{cosim}(A, B) = \frac{A \cdot B}{|A| \cdot |B|} = \frac{\sum_{i=1}^{n} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{n} a_i^2 \cdot \sum_{i=1}^{n} b_i^2}} \tag{3.2}$$

Hodnota podobnosti se pohybuje v rozmezí $< 0; 1 >$, kde 0 znamená dokumenty zcela odlišné (zpravidla v jiném jazyce) a 1 naopak dokumenty shodné (doslovné kopie). Výsledek blízký horní hranici intervalu však nemusí nutně znamenat plagiátorství a je vhodné v takovém případě dále aplikovat metriky shodnosti [11].

Podobnost na základě kompresních technik

K určení podobnosti dokumentů lze použít i kompresních algoritmů pracujících na principu vyhledávání opakujících se sekvencí bajtů. Aplikací kompresního algoritmu na dva textové úseky A a B k porovnání a následně na úsek vzniklý jejich sloučením $A+B=C$ vzniknou tři soubory A_c, B_c, C_c.

Z povahy kompresního algoritmu a velikostí získaných souborů lze vyvodit [11]:

Pokud $|C_c| \rightarrow (|A_c| + |B_c|)$, pak jsou úseky zcela odlišné.
Naopak když $|C_c| \rightarrow \max(|A_c|, |B_c|)$, pak jsou úseky shodné.

Jako výsledné procentuální podobnostní kritérium lze poté použít [11]:

$$A_c = \text{compress}(A)$$
$$B_c = \text{compress}(B)$$
$$C_c = \text{compress}(C)$$
$$\text{sim}_{\text{com}}(A_c, B_c, C_c) = \frac{A_c + B_c}{C_c} - 1 \tag{3.3}$$

Zástupcem zmíněného typu algoritmů může být například kompresní metoda LZ77 [11].

Podobnost dle algoritmu Nilsimsa

Nilsimsa je obdobou hashovací funkce. Oproti běžným hashovacím funkcím se však odlišuje tím, že pro velmi podobné dokumenty generuje velmi podobné řetězce (hashe)[13] [11].

Aplikací funkce *nilsimsa* na dva textové úseky se získají 2 hashe o délce 256 bitů, jejichž počet vzájemně shodných znaků (na odpovídajících pozicích) lze dále použít jako podobnostní kritérium. Hodnota kritéria inicializovaná na nulu se s každým shodným bitem navýší a s rozdílným sníží. Výsledná hodnota podobnosti bude v rozsahu $< -128, 128 >$, kdy dolní hranice intervalu odpovídá zcela různým dokumentům a horní hranice zcela shodným dokumentům [11].

3.4.2 Hledání shod v dokumentech

Po nalezení podobných dokumentů je třeba je srovnat se zkoumaným dokumentem, pro který byly v předchozím kroku tyto podobnosti nalezeny. Výstupem procesu hledání shod jsou páry textových úseků s nalezenou shodou, vždy jeden náležející zkoumanému dokumentu a jeden podobnému dokumentu.

Při selekci vhodného algoritmu je nezbytné vycházet ze zvolené reprezentace dokumentů v korpusu [11].

Měření shody fulltextů vyhledáváním

Vyhledávací algoritmy (string searching algorithms) hledají přesné shody v řetězcích, a jsou proto vhodné pouze pro detekci přímého plagiátorství. Oproti tomu vynikají svojí rychlostí a použitelností na libovolný (i nepředzpracovaný) korpus. Výstupem

[13] Tato vlastnost je naopak u běžného použití hashovacích funkcí (kryptografie a zabezpečení) nežádoucí.

algoritmu je zpravidla číselná hodnota indikující pozici hledaného vzoru v testovaném textovém úseku [11].

Práce [11] uvádí nejpoužívanější vyhledávací algoritmy:

Naivní vyhledávání Využívá hrubé síly. Plovoucí okno o délce hledaného vzoru je znak po znaku posouváno přes všechny textové úseky ke srovnání.

Knuth-Morris-Pratt Vylepšuje algoritmus naivního vyhledávání tím, že posouvá plovoucí okno o více než jeden znak zároveň. Informace o možné délce posunu je získána speciálním předzpracováním.

Boyer-Moore Oproti předchozím metodám aplikuje inverzní přístup, kdy namísto hledání shodných řetězců dochází k hledání řetězců, které se neshodují. Tento přístup umožní další urychlení posunu okna. Text je procházen zprava doleva (od konce).

Rabin-Karp Využívá hashovací funkce pro stanovení podobných textových úseků dokumentu. Vyhledávání samotné probíhá pouze na podobných úsecích. Existuje i modifikovaná verze umožňující hledání více vzorů zároveň.

Závěry v práci [11] uvedené na základě měření na testovacím korpusu doporučují pro nepředzpracovaný text použití algoritmu Boyer-Moore.

Měření shody fulltextů vzdálenostní metrikou

Užití vzdálenostních metrik namísto přímého vyhledávání umožní i nalezení textů mírně parafrázovaných. Jejich užití je však výpočetně náročnější a při nesprávně nastavené citlivosti může vést k falešným nálezům [11].

Používanou vzdálenostní metrikou pro texty je Levenshteinova vzdálenost. Měří vzdálenost dvou textových řetězců jako množství operací (záměn, přidání a odebrání znaku), které je nezbytné provést k transformaci jednoho řetězce na druhý [2].

Příklad:

$$\text{Káva} \rightarrow \text{K}\mathbf{a}\text{va} \rightarrow \text{Ka}\mathbf{m}\text{a} \rightarrow \text{Kam}\mathbf{n}\text{a} \implies \text{dist}_{\text{Lvsh}}(\text{"Káva"}, \text{"Kamna"}) = 3$$

Pro přechod z řetězce „Káva“ na „Kamna“ bylo třeba provést tři operace (dvě záměny, jedno přidání).

Měření shod pro vektory n-gramů

U dokumentů reprezentovaných pomocí vektoru n-gramů lze jednoduchým způsobem kvantifikovat míru shody s jiným dokumentem porovnáním počtu n-gramů, které jsou pro oba dokumenty společné. Za předpokladu použití neuspořádaných n-gramů

(je ignorováno pořadí slov v n-gramu) bude tato metoda citlivá i v případě změny pořadí slov při parafrázování [2].

Tento přístup jde aplikovat na textové úseky libovolné velikosti.

3.4.3 Metriky pro stanovení shody dokumentů

Po získání shodných úseků v obou srovnávaných dokumentech je třeba stanovit metriku pro kvantifikování míry jejich vzájemné shody. Následné práhování získané hodnoty umožňuje oddělit dokumenty s malým množstvím shod (pravděpodobně falešné nálezy) od dokumentů, ve kterých jsou kopie celých odstavců.

Výhodné je užití takové metriky, která není závislá na velikosti porovnávaných textů [2].

Asymetrická metrika

Asymetrická metrika říká, do jaké míry je jeden dokument obsažen v druhém. Vrací odlišné hodnoty podle toho, zda je srovnán dokument A vůči B nebo B vůči A [2].

Platí [2]:

$$\mathrm{con}(A, B) = \frac{|W(A) \cap W(B)|}{|W(A)|} \tag{3.4}$$

kde:

$|W(A) \cap W(B)|$ *vyjadřuje počet slov shodných v obou dokumentech*
$|W(A)| \ldots\ldots\ldots$ *vyjadřuje celkový počet slov v analyzovaném dokumentu*

Symetrizovaná asymetrická metrika

Symetrizací asymetrické metriky dochází k odstranění zmíněné nevýhody v podobě rozdílných výsledků při záměně pořadí zkoumaných dokumentů. Základním přístupem je výběr maxima z obou variant pořadí, ale lze použít i jiné přístupy, jako aritmetický průměr.

Při použití maxima platí [2]:

$$\mathrm{maxcon}(A, B) = \max(\mathrm{con}(A, B), \mathrm{con}(B, A)) \tag{3.5}$$

3.5 Rešerše algoritmů a doporučení

V subkapitole budou shrnuty algoritmy a doporučení z čerpaných zdrojů. Shrnutí umožní následný návrh optimálního detektoru plagiátů na cílovém korpusu.

3.5.1 Obecná doporučení pro intrakorpální systémy

Autor disertační práce [11] uvádí doporučený postup pro implementaci intrakorpálního detektoru plagiátů, který by byl vhodný pro rozsáhlé dokumentové sklady (korpusy). Vychází z četných praktických experimentů na testovacích korpusech. Jeho závěry jsou shrnuty v nadcházejících odstavcích.

Pro oblast **předzpracování dokumentu** autor vyzdvihuje především tu vlastnost, kdy i náročné metody předzpracování mají ve výsledku jen malý vliv na výkonnost systému jako celku, jelikož dokumenty procházejí předzpracováním ještě před vložením do korpusu. Naopak hloubka provedeného předzpracování (lemmatizace, synonymizace) může mít zcela zásadní vliv na účinnost systému. Jako nejdůležitější krok se jeví filtrace stop-slov, díky které dochází ke zhruba třetinovému navýšení citlivosti algoritmu.

Při **ukládání dokumentů do korpusu** je zavrženo fulltextové uložení textů kvůli celkové náročnosti na výpočetní výkon (v oblasti rychlosti algoritmu a dále i nároků na propustnost architektury systému). Autor doporučuje ukládat dokumenty jako neuspořádané n-gramy, u kterých byla měřením prokázána úspěšnost nálezu až o 50% vyšší než u uspořádané varianty. Zároveň ukládáním neuspořádaných n-gramů dojde ke snížení celkového počtu uložených n-gramů, a tím pádem ke zvýšení efektivity detekce. Z měření dále vyplývá, že pro česky psané dokumenty jsou nejvhodnější n-gramy o délce čtyř slov.

U **identifikace podobných dokumentů ke srovnání** jsou zásadní metadata vložená do korpusu během předzpracování. Množina ukládaných údajů by měla být vždy promyšlena v souladu s vybraným podobnostním algoritmem. Jako nejvhodnější se na základě měření ukazují být metody postavené na kosinové podobnosti a analýze klíčových slov. V případě reprezentace dokumentů pomocí n-gramů však autor uvádí nízkou efektivitu identifikace podobných dokumentů a navrhuje tento proces zcela vynechat.

Výpočet míry shody musí cílit na odhalení zdrojů plagiátů napříč celým korpusem. Je žádoucí, aby s rostoucí velikostí korpusu rostla výpočetní náročnost metriky co nejméně. Metrika se volí podle očekávaného účelu systému.

Systém vytvořený na základě předchozích doporučení by měl dle autora vykazovat optimální rovnováhu mezi kvalitou detekce (přesností) a rychlostí.

3.5.2 Extrakorpální detekce na neuspořádaných n-gramech

Autor [2] navrhl v rámci své bakalářské práce detektor plagiátů vyhledávající zdroje extrakorpálně za pomocí vyhledávače Google. Primárním účelem detektoru je taktéž detekce plagiátů v akademických pracích. Výsledná aplikace má uživatelsky přívětivé rozhraní, ve kterém lze většinu parametrů detektoru upravovat.

Implementovaný proces sestává z kroků:

- předzpracování dokumentu;
- zpracování textu pro vyhledání online zdrojů;
- vyhledání zdrojů v databázi;
- vyhledání zdrojů na síti;
- zpracování nalezených zdrojů;
- hledání shod;
- zobrazení výsledků.

Předzpracování souborů začíná vyhodnocením znakové sady, která může být i vynucena uživatelem. K převodu nalezených online zdrojů na prostý text autor využívá odstranění HTML značek a nástrojů třetích stran pro formáty DOCX[14] a PDF[15].

Interní korpus složený z akademických prací využívá filtrování na základě ignorace prvních či posledních N stran a odstranění stop-slov, jejichž seznam je uložen v konfiguračním souboru. Detektor provádí i lemmatizaci za pomoci nástroje třetí strany LemmaGen[16].

Za účelem vyhledávání online zdrojů je v dokumentu nebo jeho částech provedena extrakce klíčových slov (nejčastěji se vyskytující slova) a poté je položen dotaz vyhledáváči *Google*. Srovnání je provedeno s prvními M nalezenými zdroji.

Dokumenty jsou pro účely srovnání dále převedeny na neuspořádané n-gramy ukládané formou číselných hodnot do převodní tabulky v interní paměti.

Shody napříč dokumenty se vyhledávají na základě porovnávání jejich n-gramů. Ze získaného vektoru shod mezi analyzovaným dokumentem a možným zdrojem plagiátu probíhá výpočet kosinové podobnosti (vzorec 3.2) a míry shluků nalezených shod (další matematické kritérium). Práhováním vypočtených hodnot se určí, zda je zkoumaný zdroj podezřelý.

[14] Code7248.WordReader - `http://sourceforge.net/projects/word-reader/?source=dlp`
[15] XPDF - `http://www.foolabs.com/xpdf/home.html`
[16] `http://lemmatise.ijs.si/`

4 Softwarové prostředky pro implementaci

Kapitola uvádí krátký přehled softwarových technologií, které budou dále použity v praktických částech pro implementaci detektoru.

4.1 Kontejnerizace aplikací

Kontejnerizace je jedním z moderních trendů ve vývoji software. Jedná se o druh virtualizace s nízkým výkonnostním dopadem na hostující operační systém. Účelem je zajištění přesně definovaných stabilních podmínek pro běh cílové aplikace či služby, a to bez ohledu na cílovou hostující platformu. Díky tomuto charakteru se bude aplikace v kontejneru umístěná chovat naprosto stejně i pro přenesení na jiný počítač s jiným operačním systémem. Využití kontejnerů však s sebou nese kromě benefitů i nové výzvy v oblasti zabezpečení a komplexní úlohy v podobě jejich správy.

Pro cílovou aplikaci se definuje konfigurační soubor (*Dockerfile*, *Containerfile*), na jehož základě kontejnerizační služba vytvoří tzv. obraz (*image*). Tvorba obrazu zahrnuje například instalaci operačního systému (nejčastěji s jádrem OS Linux), nutných knihoven v požadovaných verzích, nastavení práv a proměnných prostředí apod. Hotový obraz se umístí do webového repozitáře nebo na lokální disk.

Získaný obraz lze prostřednictvím kontejnerizační služby spustit obdobně jako jiný virtuální stroj. Vzhledem k nízké náročnosti mohou běžet na jednom hostujícím stroji jednotky až desítky různých kontejnerů. S hostujícím operačním systémem kontejner komunikuje pomocí zpřístupněných síťových portů (*exposed ports*) anebo sdílených částí souborového systému (*volumes*).

Hlavní využití kontejnerizace je dnes pro rozvíjející se cloudové technologie. S těmi úzce souvisí i problematika *clusterů* (organizačních jednotek pro skupiny kontejnerů) a orchestrace (deklarativní konfigurace podoby běžícího clusteru). Uplatnění však kontejnerizace nalézá i pro lokální vývojářská prostředí a nově také i v základních strukturách některých systémů na jádru Linux.

Používaná řešení kontejnerizačních služeb jsou:

Docker Služba, díky které došlo k popularizaci kontejnerizace jako takové. Dnes mírně na ústupu. [1]

Podman Evolučně novější řešení, které odstraňuje některé nedostatky Dockeru. Zejména se jedná o oblasti zabezpečení a přenositelnosti konfigurace na cloudové orchestrační systémy. [2]

[1] `https://www.docker.com/`

[2] `https://podman.io/`

4.2 Databáze NoSQL

NoSQL databáze tvoří alternativu k tradičním relačním databázím. Jsou založeny na flexibilním modelu ukládaných dat bez pevně stanovené struktury, což umožňuje vyvíjený software v průběhu času přizpůsobovat aktuálním požadavkům a potřebám. Kromě flexibility je cílem těchto systémů také škálovatelnost a uspokojivý výkon při vysoké zátěži [14].

Existuje několik druhů NoSQL databází [14]:

Databáze dokumentů Data jsou organizována do oddělených dokumentů zapsaných ve formátu *JSON*[3]. Vnitřní struktura dokumentů je modelována přímo na základě objektů aplikace, tak aby byly operace nad databází co nejvíce přímočaré bez dalších transformací dat. Dokumentové databáze jsou vhodné na obecná použití. Patří mezi ně například *MongoDB* nebo *CouchDB*.

Databáze klíčů a hodnot Jednodušší databáze na principu asociativního pole. Řetězcové klíče odkazují na jednotlivé hodnoty. Jsou vhodné pro aplikace nevyžadující komplexní dotazování, jako je ukládání dočasných dat nebo uživatelských preferencí. Typickými představiteli může být *Redis* nebo *DynamoDB*.

Databáze širokosloupcové Obdobně jako u relačních databází organizují data do tabulek, řádků a sloupců. Podstatný rozdíl oproti SQL databázím však je, že jednotlivé řádky nemusí mít shodný počet a velikost sloupců. Jedná se o kompromis mezi výše uvedenými druhy. Představiteli jsou například *Cassandra* nebo *HBase*.

Databáze založené na grafech Dle teorie grafů uchovávají data formou vrcholů a větví. Vrcholy ukládají data o objektech reálného světa, zatímco větve určují vztahy mezi vrcholy. Jsou vhodné pro speciální aplikace s potřebou prohledávat a procházet relace mezi uloženými objekty, jako jsou sociální sítě.

Systémy NoSQL zpravidla podporují horizontální škálování. Databáze je tak tvořena decentralizovanou sadou replik obsahujících stejná data, které si mezi sebou navzájem sdílejí jednotlivé transakce. Repliky mohou být umístěny v různých fyzických a virtuálních strojích, díky čemuž jsou data chráněna oproti ztrátě selháním hardware a zároveň je umožněno číst z vícera replik zároveň. Tato vlastnost souběžného čtení je výhodná pro využití v rámci detektoru plagiátů, kde platí požadavek na co nejrychlejší běh a vysokou datovou propustnost systému.

[3] JavaScript Object Notation - je široce rozšířený formát pro přenos dat napříč webovými službami.

4.3 Výpočty akcelerované využitím grafické karty

Úloha detekce plagiátů je vzhledem k velikosti korpusu i dokumentů výpočetně náročná. Je vhodné výpočty urychlit za pomoci tzv. technik výpočtů o vysokém výkonu (*High performance computing*), především ve smyslu paralelizace co největší části detekční úlohy.

Základní metodou je využití grafické karty (*GPU*) jako alternativní výpočetní jednotky. Ta má oproti procesoru nižší rychlost vykonávání instrukcí, umožňuje však spouštění tisíců pracovních vláken zároveň.

Mezi nevýhody kódu vykonávaného na grafické kartě patří nutnost alokovat paměť ve *VRAM* (operační paměti grafické karty) a také nepřístupnost standardních komponent operačního systému, jako je souborový subsystém nebo síťový subsystém. Z toho zároveň vyplývá, že veškeré vstupně výstupní operace nad databázemi nebo operace zprostředkované aplikacemi třetích stran musí být vykonávány pouze na procesoru. Paralelizace pomocí grafické karty je tedy vhodná jen a pouze pro výpočetně náročné úlohy nad daty předem připravenými procesorem.

Známé technologie pro výpočty na grafické kartě jsou:

Nvidia CUDA Uzavřená technologie společnosti Nvidia určená pro grafické karty téže firmy. Vyznačuje se snadnou integrací kódu pro grafickou kartu do zbytku aplikace.

OpenCL Novější otevřený standard pro propojení činností procesorů, grafických karet, ale i DSP. Funguje i na kartách jiných výrobců.

V případě Nvidia CUDA jsou výpočty probíhající na grafické kartě organizovány do funkcí výpočetních jader (*kernel functions*). Na procesoru je třeba naalokovat paměť a zavolat funkci jádra s modifikátorem počtu výpočetních bloků a vláken v každém bloku. Počet bloků odpovídá počtu použitých grafických multiprocesorů. Prováděné výpočty probíhají vůči procesoru asynchronně, proto je třeba po jejich spuštění provést synchronizaci, čili vyčkat na dokončení všech operací před započetím dalších [15].

Výpočetní jádra pro Nvidia CUDA pracují s šířkou slova o délce *32 bitů*, je vhodné tedy veškeré výpočetní operace na tuto délku optimalizovat. Lze použít i 64bitové hodnoty, avšak příslušné aritmetické operace budou kompilátorem přeloženy jako série více instrukcí [16].

Funkce určené pro grafickou kartu je nezbytné překládat odděleně od zbytku aplikace, například jako statickou knihovnu. CUDA umožňuje zápis v jazycích C/C++ a dodává vlastní kompilátor *nvcc*.

5 Návrh a implementace detektoru

V kapitole je rozebrán výstup návrhu řešení nového detektoru plagiátů a specifika implementace, jež byla podle tohoto návrhu následně vytvořena. Jsou zde popsány obecné cíle pro detektor plagiátů, použité softwarové prostředky, součásti aplikace a jejich princip fungování.

5.1 Požadavky na řešení

Na základě provedených konzultací a s ohledem na potenciální využití detektoru plagiátů v rámci informačního systému VUT byly autorem práce stanoveny následující cíle a požadavky na provedení:

- aplikace musí primárně pracovat s akademickými pracemi VUT v Brně (dále uváděny jako dokumenty interní), měla by však podporovat i přidávání textů z jiných zdrojů, jako jsou lokální PDF soubory (dokumenty externí);
- musí být kladen důraz na kvalitní předzpracování dokumentů (indexaci), které zpracuje dokumenty do formátu umožňujícího co nejrychlejší hledání shod;
- detektor musí být intrakorpálního charakteru s možným rozšířením o hybridní přístupy do budoucna;
- aplikace by měla dokázat srovnat požadovaný dokument se zbytkem korpusu v časovém horizontu desítek sekund (rozšířený testovací korpus na testovacím hardware [1]), aby bylo možné ji využít ke kontrole textu na požádání;
- procesované dokumenty musí být psány latinkou a kódovány v UTF-8, ISO-8859-1 (západní latinka) nebo ISO-8859-2 (středoevropská latinka), detektor musí být schopen pracovat s texty psanými česky, slovensky a anglicky, přičemž kvalita výstupu v případě hledání shod napříč jazyky není zaručena;
- součásti řešení musí být sestavitelné pro systémy na bázi UNIX, přičemž by tyto součásti měly být kontejnerizovány tak, aby mohlo být řešení dodáno v „krabicové“ podobě;
- mělo by být využito dvojí paralelizace, a sice vlákny na procesoru (pro paralelní běh úloh aplikace) a vlákny na grafické kartě (pro hledání shod i případné jiné úlohy);
- zdrojový kód by měl být verzován, čitelný a vhodně strukturován pro snadnou možnost rozšíření.

[1] Konkrétněji v kapitole 6.

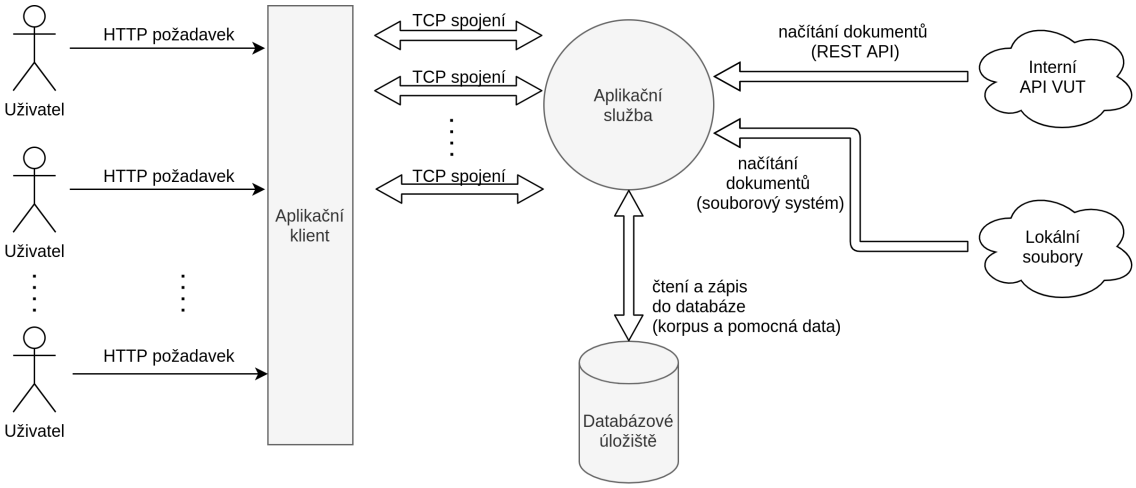
5.2 Přehled řešení

Nový detektor plagiátů navazuje na princip použitý v rámci práce [2], který je založen na reprezentaci dokumentů pomocí neuspořádaných n-gramů a následném hledáním shod v těchto reprezentacích s výpočtem metrik.

Vytvořené řešení *TDPD* (Text Document Plagiarism Detector) se skládá z trojice implementačně nezávislých komponent:

- aplikační služby (*daemon*);
- aplikačního klienta (*client*);
- databáze pro dokumentový korpus a data aplikace.

Centrální komponentou řešení je aplikační služba, která provádí indexaci dokumentů a hledání shod v jejich textech. Úlohy jsou službě předávány tenkým aplikačním klientem, který poskytuje uživateli nebo nadřazenému systému rozhraní typu REST API [2]. Služba má k dispozici úložiště v podobě databázového serveru. Textové dokumenty jsou načítány buďto z interní API (obsahující práce z VUT) anebo z externích zdrojů (lokální disk, po možném rozšíření v budoucnu i web). Komunikace mezi službou a klientem probíhá prostřednictvím TCP socketů.



Obr. 5.1: Přehled řešení detektoru plagiátů.

Provedení aplikace v podobě trvale běžící služby a tenkého klienta v roli uživatelského rozhraní umožňuje vzájemnou koordinaci úloh, efektivní nakládání s dostupnými systémovými prostředky i předchystání klíčových dat do operační paměti tak, aby byla poté k dispozici všem úlohám v co nejkratším čase.

[2]Nádstavba HTTP založená na komunikaci předáváním JSON objektů.

Vývoj řešení probíhal ve skupině čtyř kontejnerů provozovaných kontejnerizační službou *podman*. Každá z komponent aplikace tak běží v odděleném prostředí a komunikuje s ostatními výhradně síťovými protokoly. Předpokládá se nasazení kontejnerů i pro reálný provoz, zejména z důvodů zjednodušení procesu instalace a správy systému.

V diagramu přílohy B jsou znázorněny scénáře užití řešení.

5.2.1 Aplikační služba

Služba naslouchá na příchozí úlohy od klientů a poté je vykonává, přičemž si drží důležité prostředky alokované v operační paměti, aby nemusely být znova vyvolány pro každou novou úlohu.

Jedná se o aplikaci napsanou v jazyce C++ (ISO standard c++17) pro UNIXové systémy. Sestavení probíhá pomocí nástrojů *gcc*, *nvcc*, *make* a *CMake*. Zdrojový kód obsahuje soubory *.cpp* určené pro překlad pomocí gcc (běžné C++) a soubory *.cu* s funkcemi pro grafickou kartu překládané kompilátorem nvcc. Hlavičkové soubory *.h* jsou společné oběma. Výstupem nvcc je knihovna *libcuda.a*, která je ke spustitelnému souboru aplikace *tdpddaemon* staticky linkována.

Ve výkonově nekritických částech kódu je maximálně využíváno moderních funkcí jazyka C++ a standardní knihovny (*std*), jako například automatické správy paměti. Oproti tomu na prostředky náročný proces hledání shod pracuje s nízkoúrovňovou logikou, ruční správou paměti a převážně jednoduchými datovými typy.

Kód služby je objektově orientován a skládá se z modulů tříd (modul tvořen hlavičkovým a zdrojovým souborem) seskupených do struktury jmenných prostorů. Kořenovým jmenným prostorem je *TDPD*. Hierarchie jmenných prostorů služby je v příloze C.2.

Během vývoje služby bylo dbáno i na zpracování výjimek. Aplikace rozeznává mezi zachytitelnými výjimkami a výjimkami neočekávanými. Zachytitelné výjimky (*TaskException*) vznikají na úrovni úloh aplikace při např. špatném zadání úlohy nebo selhání připojení k externím zdrojům. Výjimky neočekávané způsobí pád aplikace a musí být řešeny dodatečným vývojem.

Vzhledem k rozsáhlému portfoliu činností aplikační služby jsou podrobnější informace uvedeny v dalších subkapitolách počínaje 5.3.

5.2.2 Tenký aplikační klient

Klient povyšuje interní rozhraní služby v podobě TCP serveru na spojení protokolem HTTP a vytváří tak z řešení webovou službu snadno použitelnou v rámci libovolného nadřazeného systému.

Implementace klienta je provedena na knihovnách *NodeJS*[3] v jazyce JavaScript. Spuštěná instance klienta zpracovává příchozí HTTP požadavky v asynchronním režimu, díky čemuž zvládne obsloužit více požadavků najednou. Pro každý příchozí požadavek se z předaných parametrů zkonstruuje bajtové pole definující strukturu úlohy ve formátu srozumitelném aplikační službě. Následně dojde k navázání spojení TCP socketem, zadání úlohy se odešle službě a výstup úlohy je poté postupně předán zpět do klienta.

Každé úloze se přiřadí unikátní ID skládající se z časové značky v milisekundách a pseudonáhodné složky. Výstup vykonávaných úloh je interně předáván rourou (*pipe*) do souborů dočasně vytvořených pod ID úlohy. Výstup úloh je uživateli poté prezentován z obsahu těchto souborů.

Výpis 5.1: Ukázková část kódu klienta odesílající data o úloze službě.

```
const buffers = [];

// char taskName[32];
buffers.push(Buffer.alloc(32));
buffers[buffers.length - 1].write(task.substr(0,31)+'\0','utf8');

// char output[8];
buffers.push(Buffer.alloc(8));
buffers[buffers.length - 1].write(format.substr(0,7)+'\0','utf8');

// uint16_t argc;
buffers.push(Buffer.alloc(2));
buffers[buffers.length - 1].writeUInt16LE(args.length);

// char argv[512];
buffers.push(Buffer.alloc(512));
let offset = 0;
for(arg of args)
{
    const str = arg + '\0';
    buffers[buffers.length - 1].write(str, offset, 'utf8');
    offset += str.length;
}

const data = Buffer.concat(buffers);
const size = data.byteLength;
client.write(data);
```

Návod k použití klienta je uveden v pozdější kapitole 8.

[3] https://nodejs.org/en/

5.2.3 Databázové úložiště

Za účelem ukládání dat aplikace využívá dokumentovou NoSQL *MongoDB*[4] databázi (verze 4.4.1). Důvodem pro její volbu byla jednoduchá povaha potřebných dotazů, absence relací v datovém modelu a potřeba flexibility umožňující diverzifikaci obsahu v závislosti na původu dokumentu.

Databáze je dále organizována do menších položek, tzv. kolekcí. Největší kolekce *documents* obsahuje dokumentový korpus (mapování mezi dokumenty v aplikaci a v databázi je 1:1), další kolekce slouží k perzistentnímu uložení pomocných dat užitých při indexaci. Z položek dokumentů určených k vyhledávacímu dotazování jsou vyhotoveny indexy.

Integrace MongoDB do C++ aplikace je zprostředkována ovladačem *mongocxx* a s ním dodávanou knihovnu *bsoncxx*[5].

V rámci aplikační služby je s touto databází komunikováno skrze singleton[6] třídu *Corpus*, která při vytvoření objektu inicializuje zásobník na tvorbu komunikačních klientů sdíleného mezi vlákny (tzv. *poolu*). Volání metod korpusu z pracovního vlákna získá z tohoto poolu klienta pro konkrétní vlákno a poté provádí potřebnou komunikaci. Metody singletonu komunikaci s databází plně standardizují a odstiňují od ní zbytek aplikace.

5.3 Úlohy a paralelizace

Každý požadavek na službu má tvar jedné ze známých typů úloh (*Task*). Třídy typů úloh dědí z abstraktní třídy *AbstractTask* definující společnou logiku. Úloze je vždy přiřazen výstupní proud (*stream*), do kterého odesílá svůj výstup (např. se odkazuje na otevřené TCP spojení ke klientovi).

Základem aplikační služby je singleton objekt třídy *DetectorService* vytvořený funkcí main. Ten zajišťuje vedení fronty úloh, jejich plánování a spouštění.

Výchozí vlákno aplikace naslouchá příchozím úlohám od klientů a řídí frontu úloh. Úlohy z fronty jsou spouštěny na samostatných pracovních vláknech, avšak s omezením množství paralelně běžících úloh, které je limitováno počtem dostupných procesorových jader. Další úlohy, pro které již není výpočetní kapacita, čekají ve frontě, dokud nedojde k ukončení některé z předchozích. Čekající klienti jsou pravidelně informováni zprávou o pozici své úlohy ve frontě.

Běh vícera procesorových vláken vyžaduje, aby singleton i statické třídy aplikace byly provedeny jako bezpečné pro použití s více vlákny (thread-safe provedení).

[4] `https://www.mongodb.com/`

[5] BSON = binary JSON, formát pro ukládání datových struktur užívaný v MongoDB.

[6] Česky „jedináček“ je třída, u které nelze vytvořit více než jednu instanci.

K dosažení bezpečnosti se v aplikaci využívá zámků vzájemné výlučnosti (mutex) a atomických proměnných.

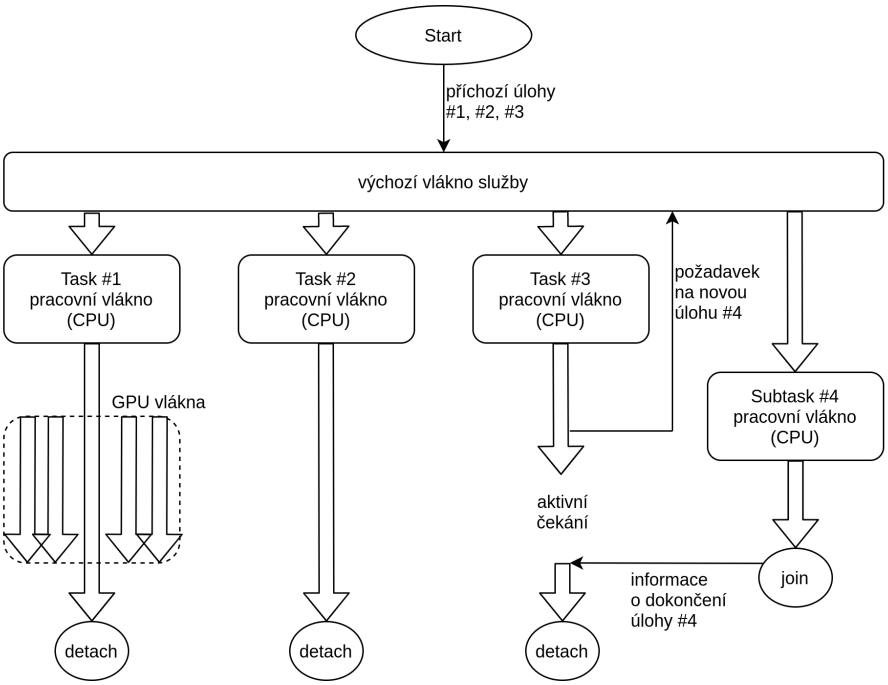
Podle pozice ve frontě se úlohy nacházejí v jednom z definovaných stavů:

TaskStatus::Pending - úloha čeká ve frontě na zpracování;
TaskStatus::Running - úloha se právě zpracovává;
TaskStatus::Finished - úloha již byla dokončena.

U spuštěné úlohy už se dále neočekává přímé spouštění dalších procesorových vláken. Nicméně je možné z již běžící úlohy zařadit do fronty novou podúlohu (*subtask*) a volitelně také aktivně čekat na její dokončení (volající pracovní vlákno je blokováno až do spuštění a dokončení novějšího pracovního vlákna). Podúlohy sdílejí výstupní proud se svým rodičem.

V závislosti na tom, zda se u dotčené úlohy čeká na dokončení nebo ne, dojde po dokončení buďto k odpojení jejího pracovního vlákna z aplikačního kontextu (*detach*), nebo k jeho připojení do volajícího pracovního vlákna (*join*). Hotové úlohy s ukončenou rutinou svého vlákna jsou poté z aplikace odstraněny.

Běžící pracovní vlákno má k dispozici možnost další paralelizace pomocí vláken spouštěných na grafické kartě (již nezávisle na výchozím vláknu služby). Výpočty na grafické kartě jsou v aplikaci realizovány pomocí technologie Nvidia CUDA.



Obr. 5.2: Hierarchie vláken v aplikaci a související signály.

CUDA byla zvolena kvůli nenáročné integraci do běžného kódu aplikace a velkému množství dostupných online materiálů. Za účelem zjednodušení častých operací nad grafickou kartou byl vytvořen i statický třídní adaptér (*Cuda*). Jeho zásadní funkcí je zpracování chybových kódů typických jazyku C na C++ výjimky.

5.3.1 TCP server

Server přijímající spojení od aplikačních klientů je realizován objektem *AppComm*. K implementaci komunikace bylo využito běžných POSIX funkcí a práce s deskriptory souborů (file descriptor). Kontrola na příchozí spojení probíhá asynchronně, aby nebylo výchozí vlákno aplikace blokováno (socket je nastaven jako neblokující).

V případě nově příchozího spojení dojde k vyčtení binárně zapsané struktury typu *AppCommTaskInfo* obsahující popis úlohy - název, očekávaný výstupní formát, počet argumentů a hodnotu argumentů. Struktura je předána objektu továrny *TaskFactory*. Dále dochází ke zpracování argumentů a vytvoření výstupního proudu z deskriptoru spojení. Případné chybné údaje v názvu nebo argumentech úlohy jsou vráceny klientovi, přičemž by došlo i k ukončení spojení. Továrna vrací připravenou úlohu, která se na úrovni *DetectorService* zařadí do fronty. Informace o příchozích spojeních a chybách jsou vypisovány do standardního výstupu služby.

Výpis 5.2: Část kódu k implementaci TCP serveru (vytvoření socketu).

```
this->sockfd = socket(AF_INET, SOCK_STREAM, 0);
if(this->sockfd == -1)
{
    auto err = std::runtime_error("Failed to create socket...");
    this->internal->WriteError(err);
    throw err;
}

fcntl(this->sockfd, F_SETFL, O_NONBLOCK);

timeval tv;tv.tv_sec = 60;tv.tv_usec = 0; // 60s
setsockopt(
this->sockfd, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv, sizeof tv
);
setsockopt(
this->sockfd, SOL_SOCKET, SO_SNDTIMEO, (const char*)&tv, sizeof tv
);

int val = 1;
setsockopt(this->sockfd, SOL_SOCKET, SO_KEEPALIVE,&val,sizeof val);
```

5.4 Formátování výstupu

Veškerý výstup aplikace je organizován do zpráv obsahujících strukturovaná data (typu *boost::property_tree*), informační textové hlášky nebo chybové hlášky. Každá zpráva má také atribut *type* indikující její účel, jenž umožňuje třídění výstupu na úrovni softwarového nadsystému.

Mezi známé typy zpráv patří:

queued - pro informace o pozici čekajících úloh ve frontě;
progress - oznamuje postup vykonávání úlohy (čistě informativní);
error - chybová hláška;
success - hlášení o úspěšném dokončení úlohy;
result - výpis výsledků zpracované úlohy;
result_detail - výpis dodatečných výsledků úlohy.

Výstupní proudy úloh používaných pro komunikaci služby s uživatelem jsou zabaleny do instancí tříd zajišťujících formátování (*Formatter*). Společným rodičem těchto tříd je abstraktní *AbstractFormatter* definující metody *WriteMessage*, *WriteError* a *WriteData*. Metody vytvoří zprávu z předaných argumentů, požadovaným způsobem ji zformátují a poté odešlou do výstupního proudu klientovi.

Ve službě je obsažen *JsonFormatter*, který je brán jako výchozí a odesílá zprávy jako sled řádkově oddělených JSON objektů (*line-delimited JSON*). Tento formát je vhodný pro strojové zpracování případným nadsystémem. Druhou variantou výstupu je *PlainFormatter* tvořící lépe čitelný strukturovaný text určený spíše pro přímé fyzické uživatele systému.

Výpis 5.3: Srovnání výstupu JSON formátování a strukturovaného textu.

```
// JSON formatter
{"type":"result","message":"Found documents","data":[{"uri":"
    samples\/sample_2.pdf","source":"external","updated":"19.03.2021
     12:07:22"}]}
{"type":"success","message":"Listed 1 documents"}

// Plain formatter
---
RESULT: Found documents

uri: samples/sample_2.pdf
source: external
updated: 19.03.2021 12:07:22
---
SUCCESS: Listed 1 documents
```

5.5 Reprezentace dokumentů

Aplikace využívá vícero reprezentací zpracovávaných dokumentů dle potřeb aktuální úlohy:

- modelový objekt třídy dědící z abstraktní *Document*;
- korpusová reprezentace (MongoDB BSON);
- zpracovaný obsah dokumentu formou ukazatele a identifikace pro hledání shod;
- záznam o existenci dokumentu s danou identifikací při procházení korpusem.

Stěžejní jsou objekty tříd odvozených z *Document*. Ty udržují k dokumentu jeho původní obsah, předzpracovaný obsah a veškerá známá metadata (údaje o původu dokumentu, časové značky, ...). Jsou také výstupem procesu indexace a umožňují aktualizaci obsahu. Aplikace rozeznává mezi interními dokumenty (*InternalDocument*) a externími dokumenty (*ExternalDocument*).

Primárním klíčem pro identifikaci dokumentů jsou unikátní řetězcové identifikátory (*URI*), které udávají cestu ke zdroji dokumentu. Položka identifikátoru je v korpusu vedena jako vyhledávací index.

Tvorba URI dokumentu:

Interní dokument - *internal://{rok obhajoby}:{id práce}*
Lokální dokument - *{úplná či relativní cesta v lokálním souborovém systému}*

Detailní přehled metadat uchovávaných o dokumentech s hierarchií souvisejících modelových tříd je znázorněna v přiloženém třídním diagramu pod B. Další příloha C.5 uvádí, jak jsou dokumenty uloženy v korpusu.

5.5.1 Formát obsahu dokumentu

Obsah zaindexovaného dokumentu je uchováván ve dvou formách:

- stránky originálního textu k zobrazení uživateli;
- předzpracovaná data pro hledání shod.

Předzpracovaná data dokumentu jsou tvořena neuspořádanými n-gramy o délce stanovené parametrem aplikace *ngrams.size*. Výchozí je délka dvou slov[7]. Neuspořádané n-gramy jsou uloženy jako celočíselné hodnoty (*NgramID_t*), které slouží jako indexy (adresy) do tabulek n-gramů uchovávaných v databázi. Tabulky přiřazují tuto číselnou hodnotu k odpovídajícímu řetězci. Oproti řetězcové reprezentaci vede

[7]Na základě předběžných testů vykázaly tříslovné n-gramy nižší citlivost na detekci parafrází. Délka tří slov byla použita jako výchozí v předcházející práci [2].

využití číselných n-gramů ke snížení objemu dat proudících mezi službou a databázovým korpusem při načítání dokumentů.

Hodnoty *NgramID_t* mají délku čtyř bajtů, což umožňuje uchování více než 4 miliard unikátních n-gramů[8]. Díky této délce lze porovnat dva n-gramy v jedné instrukci 32bitového grafického výpočetního jádra. V případě nutnosti by bylo možné délky rozšířit na 64 bitů, čímž by však došlo ke zpomalení procesu hledání shod.

Při tvorbě n-gramů z obsahu dokumentu dochází k postupnému průchodu jeho lemmatizovaných tokenů, které jsou za pomocí statické třídy *Ngram* skládány do tzv. n-gramových slov. Tato slova jsou vždy jeden řetězec obsahující tokeny daného n-gramu spojené svislicí „|“. Neuspořádanosti se docílí abecedním seřazením tokenů před jejich vložením do n-gramového slova. Získaná slova jsou převedena na čísla přes převodní tabulku „slovo → číselný index“ načtenou v operační paměti. V případě neexistence slova je do tabulky přidán nový záznam. Výsledný vektor n-gramů (indexů) je uložen do korpusu jako jedna položka binárních dat, aby bylo umožněno rychlé načtení a uložení dat[9].

Příklad tvorby n-gramů na textovém úseku:

Ukázkový text - *Dlouhý černý text na bílém.*
Lemmatizované n-gramy - *(dlouhý, černá) (černá, text) (text, bílá)*
Neuspořádané n-gramy - *(černá, dlouhý) (černá, text) (bílá, text)*
N-gramová slova - *„černá|dlouhý“ „černá|text“ „bílá|text“*
Číselné indexy - *0x00000010 0x00000011 0x00000012*

Převodní tabulky jsou do operační paměti načítány z databáze. Vzhledem k omezení velikosti MongoDB dokumentu na 16MB vyvstala nutnost rozdělit převodní tabulky do vícera databázových dokumentů. Na základě slova se proto před uložením n-gramu vypočítá umělý klíč identifikující cílový dokument (jeden ze sady 1024 dokumentů)[10]. Databázové dokumenty jsou umístěny do společné kolekce *n-grams*. Náhled na obsah jednoho z nich je přiložen v sekci C.4.

[8] Což umožňuje zaindexovat přes 350 000 dokumentů o průměrné délce 12 000 slov/n-gramů, a to i za nereálného předpokladu, že jsou všechny jejich n-gramy zcela unikátní a abecedně seřazené.

[9] Prvotní metoda uložení hodnot n-gramů v korpusu jako pole BSON (binary JSON) čísel vykazovala při testování cca o dva řády nižší výkon.

[10] Původně probíhal výpočet klíče na základě délky slova s rozdělením do maximálně 100 dokumentů. Při testování na reálných datech se však i toto rozdělení ukázalo být nedostačujícím, a proto byl navýšen počet dokumentů na 1024 a stanovena složitější funkce výpočtu.

Funkce pro výpočet umělého klíče:

$$fd(s) = (|s| \cdot s_0) \bmod 1024 \tag{5.1}$$

kde:

s *je n-gramové slovo (řetězec o délce $|s|$)*
s_0 *je číselná UTF-8 hodnota prvního znaku slova*

Do paměti se obsah dokumentů načítá přes autorem vytvořenou datovou strukturu typu *BidirectionalOrganizedMap*. Jedná se o abstrakci nad *std::map*[11] kopírující strukturu databázových dokumentů. Interně si objekt udržuje pole map, kdy každý prvek pole odpovídá jednomu databázovému dokumentu. Pro přidání či nalezení hodnoty podle klíče (slova) je třeba předat i umělý klíč. Míra složitosti operace vyhledání prvku přes tento objekt je tak $\mathcal{O}(\log(n)+1)$, přičemž při označení celkového počtu známých n-gramů v tabulkách jako N platí $n \ll N$.

Datová struktura je navíc obousměrná a umožňuje tak i rychlé vyhledání klíče (slova) podle hodnoty (indexu) s využitím obdobných umělých klíčů. Lze tak zpětně rekonstruovat textové úseky v n-gramech do čitelné podoby. Nevýhodou je vysoká paměťová náročnost struktury, jelikož jsou data uchována ve dvou kopiích.

Při návrhu byla zvažována i alternativní reprezentace n-gramů hashovacími algoritmy. Nebyla však nalezena zjevná výhoda tohoto přístupu, jelikož zvažované hashe by činily proces porovnání výpočetně náročnějším (MD5 tvoří 128 bitů, nutno porovnat v cyklu čtyři 32bitová čísla, Nilsimsa 256 bitů, nutno porovnat osm 32bitových čísel) a bez převodní tabulky by nebylo možné poskytnout funkci zpětného překladu n-gramových indexů na slova.

5.5.2 Značky se zvláštním významem

Indexy n-gramů jsou při ukládání dokumentů přiřazovány od hodnoty *0x0000000A* (10). Nižší hodnoty jsou vyčleněny jako rezerva pro značky se zvláštním významem, kterými jsou:

`NGRAM_MARKER_PAGE` - vodicí značka konce stránky ve vektoru n-gramů;
`NGRAM_MARKER_LINE` - vodicí značka konce řádku ve vektoru n-gramů;
`NGRAM_MARKER_IGNORED` - označuje úseky ignorované při hledání shod.

Zásadní nevýhodou uložení obsahu dokumentů číselnými n-gramy je absence fulltextové podoby, kvůli čemuž nelze obsah jednoznačně zpětně rekonstruovat. Právě

[11] Generické asociativní pole klíč → hodnota.

proto bylo přidáno ukládání stránek původního textu do korpusu (zmíněno v předchozí subkapitole 5.5.1). Aby však shoda mohla být uživateli na původním textu prezentována, je zapotřebí mít jako součást výstupu procesu hledání shod i informaci o pozici shody. Za tímto účelem byly vytvořeny vodicí značky stránek a řádků umožňující lokaci shody v původním textu. Přesná lokace na úrovni znaků či písmen není vzhledem k použitým technikám předzpracování možná.

Vodicí značky jsou vkládány do dokumentu během předzpracování, přičemž je důsledně dbáno na správnou propagaci umístění značek z původní fulltextové podoby až do podoby n-gramů. Nevýhodou zavedení značek je nežádoucí navýšení velikosti dokumentu [12].

5.6 Indexace dokumentů

Aby mohl být dokument použit při hledání shod, je třeba jej nejprve načíst a vložit do korpusu, tedy zaindexovat. Za proces indexace je zodpovědný objekt *DocumentIndexer* spouštěný indexovacími úlohami, který vykonává kroky:

1. získání textového obsahu požadovaného dokumentu;
2. předzpracování textu;
3. vytvoření modelové reprezentace dokumentu;
4. uložení dokumentu do korpusu či aktualizace existujícího.

Služba implementuje indexační úlohy určené pro jeden konkrétní dokument i úlohy pro dávkovou indexaci většího množství textů. Úlohy pro jeden dokument vždy uloží změny do korpusu ihned po indexaci dokumentu. Dávkové úlohy vytvářejí samostatné indexační podúlohy pro každý záznam nalezený v patřičném datovém zdroji. Změny jsou ukládány po každých 100 dokumentech, jelikož uložení změn do korpusu trvá až řádově déle, nežli indexace jednoho dokumentu. Podoba výstupu z aplikace při zpracování dávkové indexace je k dispozici pod D.3.

Při pokusu o indexaci již známého dokumentu je výchozím chováním vyhození výjimky a dokument není v takovém případě aktualizován. Aktualizace existujícího dokumentu vyžaduje předání explicitního požadavku indexační službě. Toto chování vyhovuje dávkovým úlohám indexujícím rozsáhlé datové zdroje (typicky interní práce VUT), protože je tak umožněno dodatečné doindexování nových dokumentů chybějících v korpusu.

[12] Za předpokladu průměrného počtu šesti n-gramů na řádek je délka uloženého obsahu navýšena o přibližně 17%.

5.6.1 Normalizace textu

Jako součást indexace je před dalším předzpracováním provedena normalizace textů do znakové sady UTF-8 a široké délky znaku 32 bitů (využívá *wchar_t*, *std::wstring*). Normalizace zaručí, že v následujících algoritmech každému jednomu písmenu náleží právě jeden prvek znakového řetězce. Důvodem je zabránění běžně se vyskytujícím problémům u znaků s diakritikou.

Příklad reprezentace textu s diakritikou (*řeka*) během indexace:

Výchozí UTF-8 pro 8 bitů (*char*, vyžadováno 5 prvků řetězce)
- ř(0xc5, 0x99) e(0x65) k(0x6b) a(0x61);

Normováno pro 32 bitů (*wchar_t*, vyžadovány 4 prvky řetězce)
- ř(0x0000c599) e(0x00000065) k(0x0000006b) a(0x00000061);

Na indexaci českého i slovenského jazyka by plně postačila délka znaku 16 bitů (*char16_t*, *u16string*), jež však není kompatibilní s ostatními funkcemi ve standardní knihovně *std*, a sice například s regulárními výrazy.

Užití 32bitových znaků s sebou nese i nevýhody, zejména ve větším množství spotřebované paměti a nekompatibilitě s rozhraními standardního výstupu či MongoDB. Před výpisem normovaného textu je nutné první provést konverzi, za kterou jsou zodpovědné funkce v modulu *Encoding*.

K účelům podobných konverzí bylo dříve možné použít standardní knihovny *codecvt*, která je nyní označena jako zastaralá bez uvedené náhrady[13]. Alternativně by bylo možné text rovnou převést do UTF-32, které by sice umožnilo přímý výpis do standardního výstupu, ale stále by neumožňovalo kompatibilitu s dalšími rozhraními.

5.6.2 Předzpracování textu

Předzpracování (*preprocessing*) i normalizace textu probíhají prostřednictvím typu *PreprocessorPipeline* (pipeline[14] předzpracování). Jedná se o flexibilní třídu, do které je možné zaregistrovat úkony předzpracování (potomky abstraktní třídy *PreprocessorStage*). Po spuštění pipeline budou zaregistrované úkony sekvenčně vykonány.

Hlavní funkcí pipeline je správa mezivýsledků. Běžící úkony si mohou kdykoliv z pipeline vyzvednout některý ze dříve vyhodnocených mezivýsledků a uložit do ní

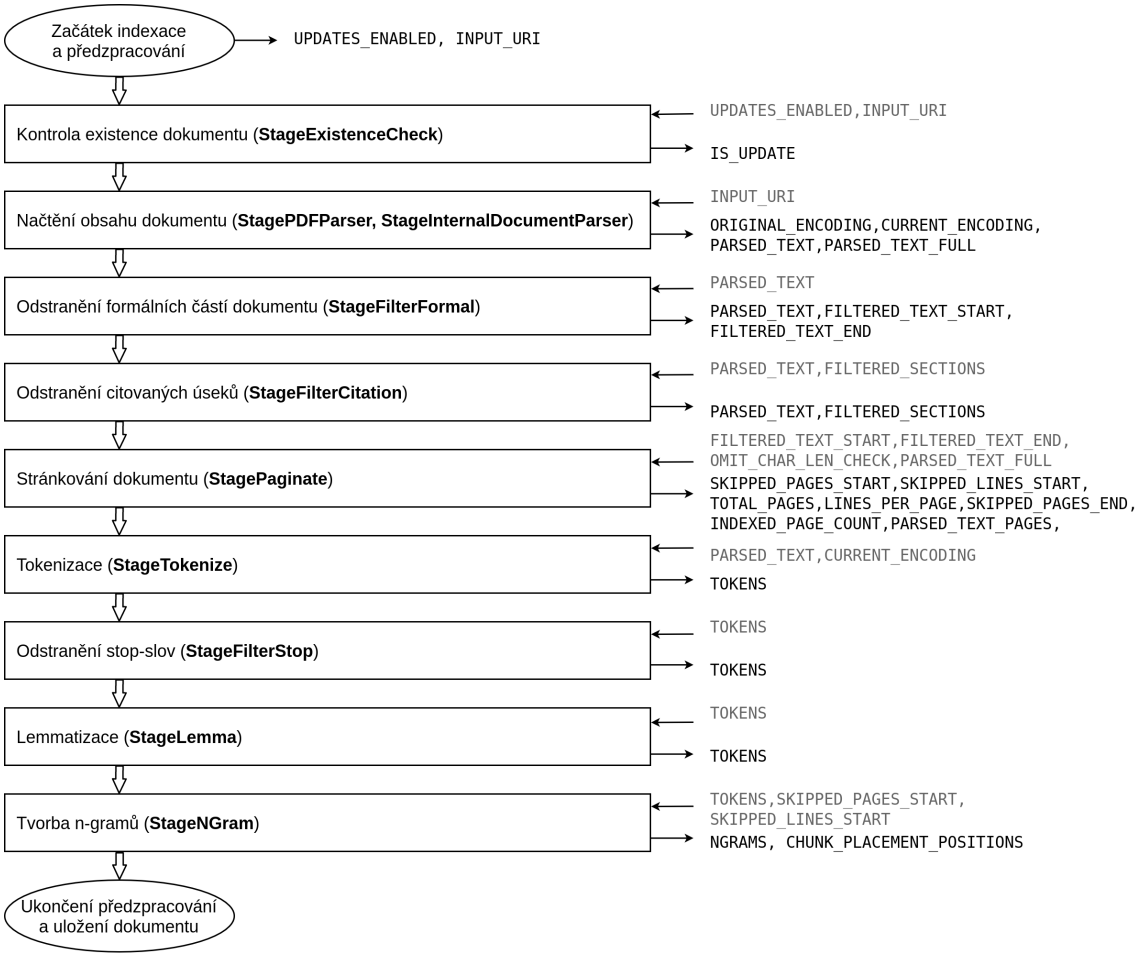
[13]Údaj o označení potřebných funkcí za zastaralé např. zde `https://en.cppreference.com/w/cpp/locale/codecvt`.

[14]Česky „potrubí“, je myšleno jako výraz pro sérii vykonávaných úloh, které si navzájem předávají data.

své vlastní. Práce s mezivýsledky probíhá na bázi řetězcových indexů do kolekce *std::any*, které jsou deklarovány jako konstanty v hlavičkovém souboru *indices.h*. Mezivýsledky tak mohou uchovat libovolný datový typ.

Výhodou implementovaného uspořádání je, že pipeline umožňuje dynamickou registraci různých úkonů předzpracování v závislosti na indexovaném dokumentu bez nutnosti vytvářet explicitní rozhraní pro předávání mezivýsledků. Stejně tak je možné volit pořadí, ve kterém budou úkony provedeny.

Následující subkapitoly popisují činnost úkonů použitých k indexaci dokumentů (také na obrázku).



Obr. 5.3: Úkony předzpracování při indexaci dokumentu a související mezivýsledky.

Úkon StageExistenceCheck

Zkontroluje, zda daný dokument existuje v korpusu či nikoliv. Vytvoří zachytitelnou výjimku v případě, kdy je dokument již indexován a aktualizace není umožněna.

Pro kontrolu se využije vyhledání MongoDB dokumentu v korpusu dle požadované cesty.

Úkon StagePDFParser

Úkon na základě předané lokální cesty k PDF souboru získá jeho obsah v podobě normalizovaného textového řetězce při zachování rozložení textu na stránce. Součástí řetězce jsou také znaky konců stran „\f“.

Během vývoje se nepodařilo nalézt vhodnou bezplatnou knihovnu pro analýzu PDF, kterou by bylo možné přeložit společně s aplikační službou. Proto bylo využito sady programů *XpdfReader*[15], jejíž součást *pdftotext* je tímto úkonem spouštěna jako subproces. Program je k dispozici pod GPLv3 licencí[16]. Práci s procesem zprostředkovává implementovaná třída *Process*, která tvoří nádstavbu nad knihovnou *boost::process*.

Úkon StageInternalDocumentParser

Obdobně jako v případě PDF parseru, parser interních dokumentů má za úkol na základě identifikace (id a rok odevzdání) interní práce VUT získat její text ve formě normalizovaného textového řetězce.

Stahování interních dokumentů zprostředkovává služba *InternalDatabaseAPI*, jež komunikuje se vzdáleným serverem pomocí REST API. Ke tvorbě HTTP požadavků se používá knihovna *curlpp* a implementovaný adaptér v podobě pomocné třídy *Curl*.

Výstupem API jsou informace o dokumentu a textový obsah stránek. Navrácené stránky jsou znakem „\f“ spojeny do jednoho řetězce, který je poté normalizovanán a uložen jako mezivýsledek předzpracování.

Úkon StageFilterFormal

Z textu se pokouší odstranit formální úseky, které by s velkou pravděpodobností byly označeny za plagiát. Cílem je ponechat u závěrečných prací pouze čistopis od úvodu po závěr. Úkon pracuje na základě regulárních výrazů (*std::wregex*). Výrazy jsou kompilovány pro syntax *EcmaScript* a optimalizovány pro rychlost.

V prvním kroku je testována sada výrazů hledajících počátek čistopisu (obsah, úvod, seznam obrázků, anotaci, ...) v prvních 15% dokumentu. Výrazů je více, aby byla umožněna prioritizace (nalezení úvodní kapitoly je větší úspěch, nežli nalezení pouze abstraktu práce).

[15] `https://www.xpdfreader.com/`

[16] GPLv3 je copyleft licence, která zaručuje právo na (i komerční) použití a modifikaci dotčeného SW pod podmínkou, že bude i nadále distribuován jako GPL.

Regulární výrazy pro vyhledání začátku:

- `(^|\\n)\\f[ \\t1.]*((.vod)|(Introduction)|(Intro))[ \\t]*(\\n|$)`
- `(^|\\n)\\f[ \\t]*((Abstrakt)|(Abstract))[ \\t]*(\\n|$)`
- `(^|\\n)\\f[ \\t1.]*((Obsah)|(Contents)|(Outline)|`
  `(Seznam (obr.zk.|tabulek|v.pis.)))[ \\t]*(\\n|$)*$`

Obdobným způsobem je poté aplikován jeden regulární výraz na vyhledání konce obsahu dokumentu (literatura, přílohy). Vyhledávání konce probíhá v posledních 40% dokumentu (přílohy mohou být rozsáhlé).

Regulární výraz pro vyhledání konce:

- `(^|\\n)\\f[ \\t]*((Literatura)|(Pou.it. literatura)|(Liter.rn. zdroje)|`
  `(Zdroje)|(Bibliography)|(References)|(Seznam p..loh))[ \\t]*(\\n|$)`

Úkon StageFilterCitation

Úkon odstraní z textu citované úseky dvou typů:

- odstavce s citací až po místo výskytu citace;
- seznamy i odstavce za uvozovací větou končící dvojtečkou a opatřenou citací.

Funkce je opět založena na principu regulárního výrazu, kdy jsou nalezené úseky z původního textu odstraněny. Odstraněné konce stran a řádků jsou zpětně na daná místa doplněny, aby nedošlo k narušení stránkování dokumentu.

Při návrhu regulárního výrazu byl volen kompromis mezi jeho rychlostí a rozumným množstvím bezpečně odstranitelného textu. Regulární výraz vždy vyhledává v části textu o maximálně 100 000 znacích[17] (vzdálenost mezi sousedícími citovanými úseky nesmí překročit tuto hodnotu, aby nedošlo k ukončení úkonu).

Výraz pro vyhledání obou typů citovaných úseků:

```
((\\n|^)[^\\n]+){1,30}(\\[[0-9]{1,2}]|\\([ ]?[^0-9\n]{1,25},[ ]{0,20}[0-9]{4}
(,[ ]{0,20}[pstr0-9., ]{1,20})?\\))([ ]{0,20}:\\n(\\n[^\\n]+){1,30})?
```

Předpokládá se odkazování pomocí číselných odkazů (např. „od autora [1]“) nebo užití harvardského systému („od autora (Novák, 2018, str. 10)“).

Úkon StageTokenize

Projde text ve formě normalizovaného řetězce a vytvoří z něj vektor tokenů, tedy oddělených slov zbavených všech znaků kromě písmen převedených na malá.

[17] Navzdory snaze o optimalizaci regulárního výrazu docházelo před uvedením tohoto limitu k chybným přístupům do paměti na úrovni standardní knihovny (u dlouhých dokumentů).

Za písmena jsou pokládány UTF-8 znaky:

- malé anglické abecedy 0x00000060 - 0x0000007A;
- velké anglické abecedy 0x00000041 - 0x0000005A;
- Latin 1 supplement 0x0000C380 - 0x0000C5BF;
- Latin extended A 0x0000C480 - 0x0000C7BF;
- Latin extended B 0x0000C680 - 0x0000C9BF.

Převod na malá písmena podporuje pouze písmena anglické abecedy a české znaky s diakritikou. Pro aplikaci na celé spektrum výše uvedených znaků by bylo nutné užít externí knihovnu nebo vytvořit obsáhlejší převodní tabulku.

Tokenizace je implementována jako paralelní úloha pro Nvidia CUDA. Počet vláken je odvozen od délky dokumentu a limitu 500 tokenů/vlákno. Vlákna si mezi sebou lineárně rozdělí text na kratší úseky.

Během tokenizace je dbáno na propagaci konců stránek a řádků (tvoří samostatné tokeny „\n“ a „\f“), kontrolu hranic slov při přesahu vyčleněné paměťové oblasti vlákna a spojení výrazů oddělených pomlčkou do jednoho tokenu (v akademických pracích je časté rozdělení slova na konci řádku pomlčkou vlivem zarovnání textu do bloku).

Úkon StageFilterStop

Provádí odstranění stop-slov a všech jednopísmenných tokenů (mohou vznikat při procesu tokenizace v rovnicích aj.). Nalezené tokeny jsou tímto filtrem z vektoru odstraněny. Seznam stop-slov k odstranění je udržován v konfiguračním souboru aplikační služby.

Úkon StageLemma

Zlemmatizuje tokeny v textu. Ty, ke kterým není známé lemma, jsou ponechány v originálním tvaru.

Lemmatizace využívá stejně jako *StagePDFParser* tvorby subprocesu pro volání externího programu určeného k lemmatizaci českého jazyka. Na základě praktického odzkoušení dostupných lemmatizátorů pro češtinu[18] byl vybrán software *Majka*[19] ([17]) vyvinutý na Masarykově univerzitě. Tento software je distribuován pod GPLv2 licencí a společně s ním je dodáván i lemmatizační slovník[20]. Cesta k programu je volitelná v parametrech aplikace.

[18]Seznam k dispozici na `https://cs.wikipedia.org/wiki/Lemmatiz%C3%A1tor`.

[19]`https://nlp.fi.muni.cz/czech-morphology-analyser/`

[20]Slovník dokáže lemmatizovat slova běžné mluvy, u odborných výrazů však selhává (dle zběžného odzkoušení během implementace).

Tokeny, ke kterým bylo již lemma určeno, jsou vedeny v perzistentní lemmatizační cache. Práce úkonu probíhá průchodem vektoru tokenů, přičemž je každý token vyhledán v cache. Nalezené lemma jej poté ve vektoru nahradí. Neznámé tokeny jsou předány službě *Lemmatizer*, která je zodpovědná za spuštění lemmatizačního subprocesu a zpracování získaného výstupu. Pro celý úkon je použita pouze jedna instance subprocesu. Potřebné tokeny jsou průběžně zapisovány na jeho standardní vstup a proces na ně průběžně odpovídá.

Podobně jako v případě n-gramů (v kap. 5.5.1), bylo nezbytné lemmatizační cache rozdělit před uložením do vícera MongoDB dokumentů. Cache je proto vedena formou datového kontejneru *OrganizedMap*, který funguje shodně s již popsanou *BidirectionalOrganizedMap*, ale neumožňuje inverzní hledání klíče podle hodnoty. Lemmata jsou tak uložena v 512 dokumentech, do kterých jsou tříděna na základě následujícího umělého klíčování:

$$\mathrm{d}(s) = (|s| \cdot s_0) \bmod 512 \tag{5.2}$$

kde:

- s *je výchozí token před lemmatizací (řetězec o délce* $|s|$*)*
- s_0 *je číselná UTF-8 hodnota prvního znaku tokenu*

Předpokládá se, že cache nebude nutné nikdy zneplatnit (jednou nalezené lemma bude platná po celou dobu životnosti aplikace).

Náhled na vybraný databázový dokument s lemmatizační cache je k dispozici v příloze C.3.

Úkon StageNGram

Vytvoří z n-tic tokenů n-gramy a vloží vodicí značky. Princip tvorby n-gramů je již vysvětlen v kapitole 5.5.1.

Úkon prochází vektor tokenů, vyhledává n-gramová slova v převodních tabulkách a přidává do tabulek nové záznamy. Vodicí značky jsou vkládány vždy za n-gram, ve kterém by se jim odpovídající token nacházel (tyto tokeny „\f“ a „\n“ tak nejsou součástí tvořených n-gramových slov). Výstupem úkonu je vektor indexů n-gramů tvořících obsah předzpracovaného dokumentu.

5.7 Funkce vynechání ignorovaných vět

Součástí existujícího řešení pro detekci plagiátů na VUT je funkce umožňující větu označenou jako shodu v rámci uživatelského rozhraní ignorovat. Ignorované věty se ukládají do databáze informačního systému a při porovnání dokumentů nejsou brány v potaz. Využití je zejména pro vyjmutí často opakujících se frází z nalezených shod. Po dohodě byla funkce přidána i do tohoto nového řešení.

Aplikace si v databázi udržuje vlastní repliku ignorovaných vět předzpracovaných do potřebné podoby. Replika je tvořena synchronizačním procesem dostupným pod úlohou aplikační služby *TaskIgnoredSync*.

Věty jsou k dispozici na interním API včetně databázových ID (celočíselné klíče). Synchronizace nahlédne do kolekce *ignored* v MongoDB (pokud existuje) a získá maximální zpracované ID ignorované věty. Následně jsou přes *InternalDatabaseAPI* získány věty s ID větším, než je toto maximum. Je tak prováděna přírůstková synchronizace umožňující pravidelnou aktualizaci udržované repliky. Nalezené věty jsou dále předzpracovány, normovány, uloženy a nové maximální ID je zapsáno do databáze.

Získané věty jsou rozděleny do dávek o počtu daném konfiguračním parametrem služby *ignored.batch_size* [21]. Dávka je před dalším zpracováním spojena do jednoho řetězce, ve kterém jsou věty odděleny znakem konce řádku „\n“.

Ignorace vět v dokumentech neprobíhá při předzpracování, ale až v rámci hledání shod. Lze tak zpětně aplikovat nově ignorované věty na již indexované dokumenty. Vzhledem k tomu je nezbytné každý řetězec dávky předzpracovat podobně jako dokumenty ukládané do korpusu.

V rámci předzpracování ignorovaných vět probíhají úkony:

- stránkování pro kontrolu limitů na počet znaků (*StagePaginate*);
- tokenizace (*StageTokenize*);
- odstranění stop-slov (*StageFilterStop*);
- lemmatizace (*StageLemma*);
- tvorba n-gramů (*StageNGram*).

Vzhledem k odstranění stop-slov bylo nutné zavést limit pro minimální délku věty, neboť příliš malé množství n-gramů nenese dostatečnou informaci pro prohlášení úseku ignorovaným. Limit je konfigurovatelný parametrem *ignored.min_len*, kde výchozí hodnota činí minimálně tři n-gramy.

Pro usnadnění aplikace vět na načtený obsah dokumentu (hledání a nahrazení odpovídajících úseků ignorovanými značkami `NGRAM_MARKER_IGNORED`) jsou všechny

[21] Výchozí hodnotou je 512 vět na dávku. V době psaní práce bylo po synchronizaci uloženo cca 138 000 zpracovaných vět o celkové normované délce 3 321 120 n-gramů.

ignorované věty normovány na shodnou délku (v n-gramech) danou konfiguračním parametrem *ignored.norm_len*. Vektor vzniklý předzpracováním je znova procházen a věty kratší než stanovená délka jsou doplněny ignorovanými značkami, věty delší naopak zkráceny. Jako poslední platný n-gram ve větě je vždy nastaven vodicí znak konce řádku.

Ilustrační ukázka normování ignorované věty (norma 8 n-gramů):

Věta:
Sítě se učí algoritmem backpropagation.

Normováno:
(síť, učit) (učit, algoritmus) (algoritmus, backpropagation) (\n) (-) (-) (-) (-)

Načtené ignorované věty jsou v paměti spravovány strukturou *IgnoredSentences*.

5.8 Algoritmus hledání shod

Úloha hledání shod *TaskDetect* načte parametrem zvolený předmětný dokument (*subject*) z korpusu a porovná jej oproti všem ostatním dokumentům v korpusu (*compared*, testované dokumenty). Cílem úlohy je mezi dokumenty nalézt vzájemné shody, které mohou být potenciálními plagiáty.

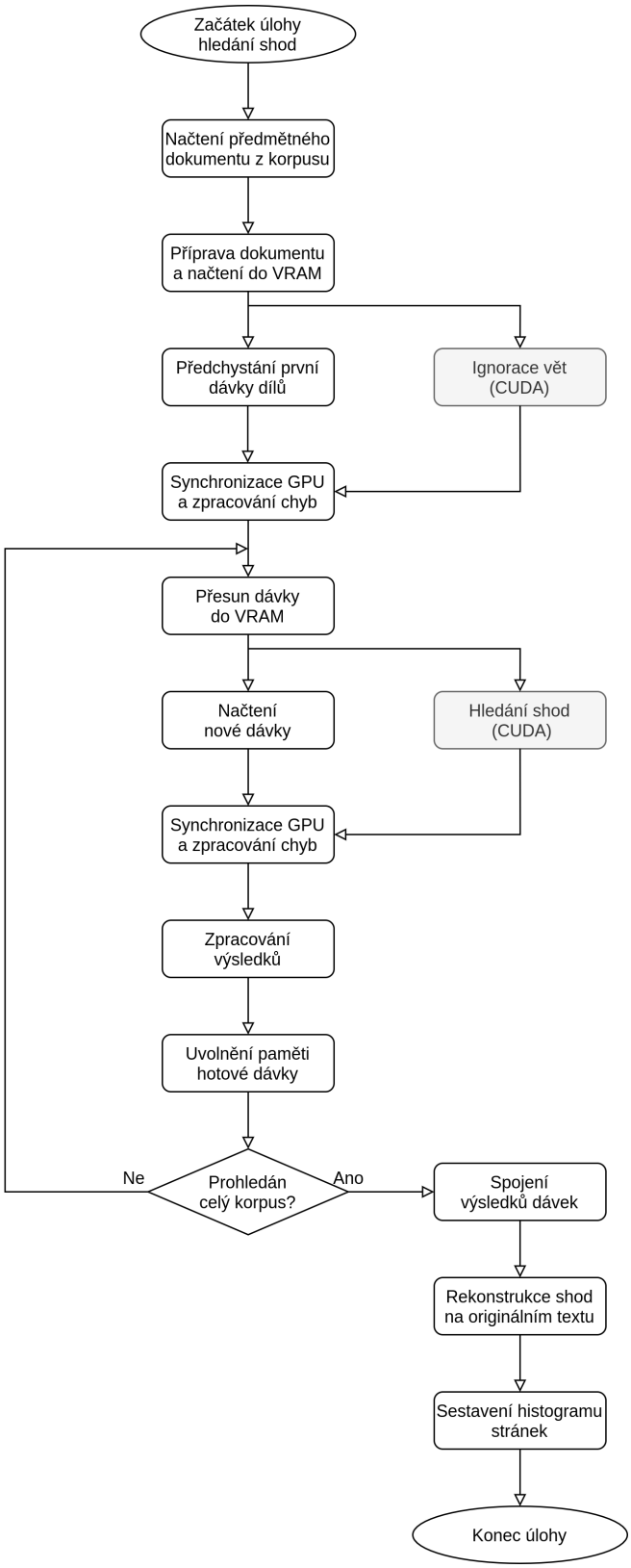
Na základě doporučení (kap. 3.5.1) nepředchází procesu hledání shod vyhledání podobných dokumentů, jelikož aplikace pracuje s n-gramy a navíc pouze v rámci vlastního korpusu (intrakorpální přístup).

Časově úsporného hledání shod napříč celým korpusem je docíleno využitím paralelizace na grafické kartě. Obsah testovaných dokumentů se rozdělí na menší celky o velikosti dané konfiguračním parametrem služby *compare.batch_chunking_limit*, dále zvané jako tzv. dokumentové díly (*chunks*).

Ke každému dílu náleží vlastní grafické vlákno, které s ním porovná předmětný dokument. Rozdělení na díly navyšuje efektivitu výpočtů rovnoměrnějším zatížením grafických vláken[22]. Velikost dílů však není vždy zcela shodná, neboť poslední díl každého dokumentu nemusí dosahovat předepsané délky.

Vzhledem k velkému množství dokumentů v korpusu je každý proces hledání shod rozdělen na sérii dávek (*DocumentMemoryBatch*), za jejichž načítání je zodpovědná pomocná služba *DocumentMemoryManager*.

[22]Před zavedením rozdělení na díly pracoval algoritmus s celým dokumentem pro každé grafické vlákno. Takové uspořádání však vedlo k neuspokojivé výkonnosti. Zpracování dávky na testovacích datech trvalo přibližně 2 minuty, přičemž se dá očekávat, že na reálných datech by mohlo být až několikanásobně delší.



Obr. 5.4: Vývojový diagram úlohy hledání shod.

Dávka je tvořena obsahem dokumentových dílů (v číselných n-gramech) a souvisejícími metadaty. Počet dílů v každé dávce je volitelný v konfiguraci aplikace, přičemž tato hodnota úzce souvisí s počtem grafických vláken a bloků pro proces hledání shod vyčleněných. Dávka sestává vždy pouze z kompletních dokumentů. Tedy v případě, kdy poslední dokument načítaný do dávky překračuje stanovenou velikost dávky, dojde k překročení této velikosti. Parametrická velikost dávky musí proto být vždy menší, nežli je celkový počet grafických vláken.

Na začátku úlohy je předmětný dokument k porovnání načten z korpusu jako modelová reprezentace třídy *Document*. Obsah dokumentu je připraven do grafické paměti VRAM a dochází na něm k aplikaci synchronizovaných ignorovaných vět.

Každá dávka načte z korpusu do operační paměti obsah tolika dokumentů, aby došlo k jejímu naplnění. Při vkládání dokumentu do dávky je provedeno rozdělení jeho obsahu na díly. Před užitím dávky pro srovnání na grafické kartě je obsah dávky přesunut z operační do grafické paměti VRAM.

Přesunutím dat do VRAM dojde k uvolnění další operační paměti, do které může být předchystána nová dávka. Předchystání nové dávky započne paralelně se zahájením hledání shod na dávce v grafické paměti. K synchronizaci GPU vláken dochází až po dokončení přípravy nové dávky. Po synchronizaci vláken jsou uloženy výsledky, VRAM paměť staré dávky je uvolněna, paměť nové dávky se přesune do VRAM a celý proces je opakován až do vyhledání shod napříč celým korpusem.

Po vyhodnocení všech dávek jsou nalezené shody spojeny do jednoho vektoru. Průchod shodami načte originální text dokumentů (předmětného a testovaných s pozitivními nálezy) v podobě stránek a umístí nalezené shody do načtených stránek na základě informací získaných z vodicích značek. Je sestaven histogram z nalezených shod (více v 5.8.4). Získaná data jsou prezentována uživateli.

5.8.1 Ignorace vět v rámci přípravy dokumentu

Ignorované věty popsané v kapitole 5.7 jsou aplikovány ve fázi přípravy dokumentu před započetím hledání shod na dávkách. Aplikace probíhá pouze na předmětný dokument, což je dostačující k vyjmutí dotčených úseků z nalezených shod.

Pro urychlení tohoto kroku je využito paralelizace na grafických vláknech. Vlákna si mezi sebou dokument rovnoměrně rozdělí, přičemž počet vláken odpovídá délce dokumentu v n-gramech dělené konfigurační hodnotou *ignored.thread_ngrams*[23].

Algoritmus pracuje ve dvou vnořených úrovních. Vnější úroveň srovnává n-gramy vyčleněné části předmětného dokumentu s počátečními n-gramy ignorovaných vět.

[23]Výchozí je 512 n-gramů. Algoritmus není přizpůsoben pro fungování na hranicích paměťových oblastí vláken, a tudíž je vhodné zmíněnou hodnotu volit spíše větší než menší.

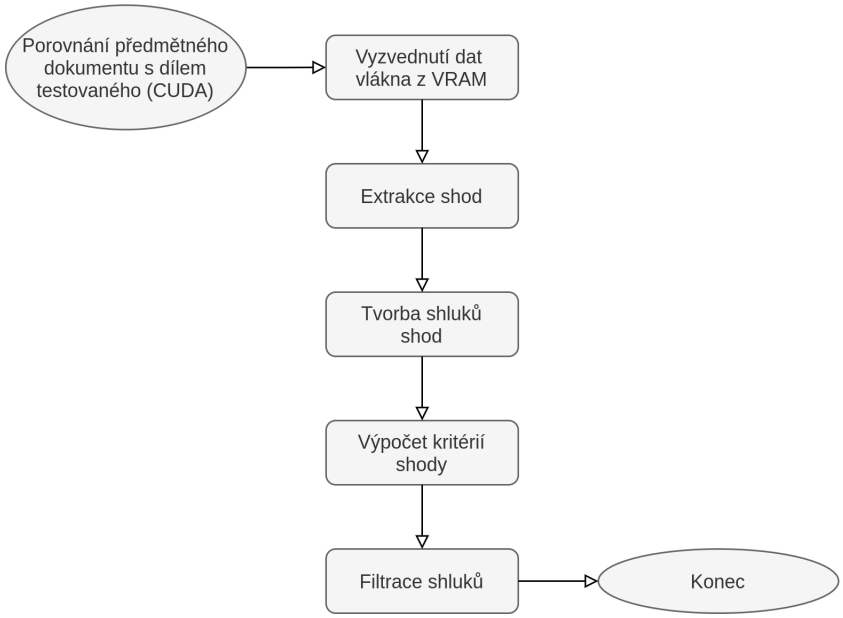
Díky předchozí normalizaci vět je možné projít jejich n-gramy s krokem odpovídajícím normě.

V okamžiku, kdy si n-gramy odpovídají, postupuje algoritmus do vnitřní úrovně. Ta projde celou ignorovanou větu a kontroluje soulad n-gramů na odpovídajících si pozicích s předmětným dokumentem. Pokud souhlasí všechny porovnané n-gramy, je celý úsek ignorován a nahrazen ignoračními značkami (`NGRAM_MARKER_IGNORED`).

5.8.2 Porovnání na úrovni grafických vláken

Po zahájení hledání shod na připravené dávce porovnává každé vlákno předmětný dokument oproti jednomu z dílů testovacích dokumentů umístěných v dávce.

Konkrétní grafické vlákno si po spuštění vyzvedne z paměti ukazatele na data příslušející číselnému indexu vlákna[24]. Společně s dokumentovými díly jsou předány i informace o poloze těchto dílů v rámci svého dokumentu (číslo n-gramu, řádku a stránky). Poté postupně probíhá extrakce shod na úrovni n-gramů, tvorba shluků z těchto shod vzdálenostní metrikou a výpočet kritérií shody na nalezených shlucích. Nakonec dojde k filtraci získaných shluků na základě konfigurovatelných prahů.



Obr. 5.5: Porovnání dvou dokumentů v rámci vlákna úlohy hledání shod (CUDA).

[24]Každé vlákno v CUDA má k dispozici údaj o pořadovém čísle svého bloku, pořadovém čísle sebe sama uvnitř bloku a velikosti bloku. Na základě těchto údajů je stanoven číselný index vlákna.

Extrakce shod

Extrakce shod probíhá dvojitým cyklem procházejícím předmětný dokument a testovaný díl „každý s každým“. Míra složitosti je tak $\mathcal{O}(m \cdot n)$.

Shoda n-gramů v testované iteraci vyústí ve vytvoření nové instance struktury *DocumentComparisonMatch*. Do té se uloží pozice shody na bázi indexu n-gramu, čísla stránky i čísla řádku, a to zvlášť pro předmětný dokument i testovaný díl (obě strany shody). Při průchodu algoritmu jsou čítány nalezené vodicí značky pro určení stránkové a řádkové pozice shody. Ignorované n-gramy jsou zcela vynechány.

Tvorba shluků shod

Struktura *DocumentComparisonMatch* pro ukládání shod má všechny pozice vedeny dvakrát, jednou pro začátek a druhou pro konec shody. Toho se využívá pro tvorbu shluků, kdy jsou sousedící shody seskupeny tak, aby tvořily co nejmenší počet co největších shod a umožnily tak identifikovat celé plagiované odstavce či stránky. Shlukování zároveň umožňuje efektivní filtraci podle významnosti shod a ulehčuje výpočet kritérií shody.

Zda jsou dvě nalezené shody sousedící, se rozhoduje práhováním autorem navržené vzdálenostní metriky na principu min-max:

$$\begin{aligned} d_x &= \min(|a_{substart} - b_{subend}|, |a_{subend} - b_{substart}|) \\ d_y &= \min(|a_{compstart} - b_{compend}|, |a_{compend} - b_{compstart}|) \\ \mathrm{dist}(d_x, d_y) &= \max(d_x, d_y) \end{aligned} \tag{5.3}$$

kde:

$dist(d_x, d_y)$.	*metrika pro rozdíly vzdáleností d_x a d_y*
d_x	*vzdálenost shod v předmětném dokumentu*
d_y	*vzdálenost shod v testovaném dokumentu*
a	*známá shoda, kterou rozšiřujeme*
b	*testovaná shoda*
$\{a, b\}_{substart}$.	*začátek pozice n-gramu shody v předmětném dokumentu*
$\{a, b\}_{compstart}$	*začátek pozice n-gramu shody v testovaném dílu*
$\{a, b\}_{subend}$ . .	*konec pozice n-gramu shody v předmětném dokumentu*
$\{a, b\}_{compend}$	*konec pozice n-gramu shody v testovaném dílu*

Extrahované shody mají vždy délku jedna (počáteční a koncové indexy i stránky jsou totožné). Tvorba shluků začíná seřazením shod podle indexu nálezu v předmětném dokumentu. Následuje postupné procházení shluků, výpočet vzdálenostní metriky

a podmíněně rozšiřování shluku na známé pozici se zneplatněním použitého jiného shluku (dochází ke spojení těchto shluků do jednoho).

Rozšiřování probíhá aplikací minima na počáteční indexy obou spojovaných shluků a aplikací maxima na koncové indexy shluků. Stejný proces platí i pro pozice stránek. U pozic řádků taktéž, avšak se zohledněním stránkových pozic spojovaných shluků. Nakonec jsou odstraněny z paměti zneplatněné shluky a ponechány pouze rozšířené shluky.

Ilustrační text před shlukováním (shody/shluky zvýrazněny šedou barvou):

(jež konzultace) *(mít uživatele)* *(uživatele význam)* *(význam zvláštní)* *(zvláštní lze)* *(lze přidat)* *(přidat příznak)* *(příznak symbol)* *(symbol vlajka)* *(vlajka zobrazený)* *(zobrazený daný)* *(daný najetí)* *(najetí řádek)* *(řádek konzultace)* *(konzultace konzultace)* *(konzultace příznak)* *(příznak mít)* *(mít poté)* *(poté symbol)* *(symbol vlajka)* *(vlajka vždy)*

Ilustrační text po shlukování:

(jež konzultace) *(mít uživatele)* *(uživatele význam)* *(význam zvláštní)* *(zvláštní lze)* *(lze přidat)* *(přidat příznak)* *(příznak symbol)* *(symbol vlajka)* *(vlajka zobrazený)* *(zobrazený daný)* *(daný najetí)* *(najetí řádek)* *(řádek konzultace)* *(konzultace konzultace)* *(konzultace příznak)* *(příznak mít)* *(mít poté)* *(poté symbol)* *(symbol vlajka)* *(vlajka vždy)*

Výpočet kritéria shody

Pro získané shluky jsou vypočtena kritéria shody aproximovanou symetrizovanou asymetrickou metrikou (vztah v kapitole 3.4.3) adaptovanou pro užití s n-gramy namísto slov. Metrika nevrací přesný výsledek v souladu s uvedeným vztahem (proto „aproximovanou“), jelikož počítání průniku n-gramů na obou stranách shody bylo zjednodušeno za účelem snížení výpočetní náročnosti.

Princip počítání počtu společných n-gramů (průniku) je založen na krocích:

1. celočíselné pole o 256 prvcích je inicializováno na nulové hodnoty;
2. pro každý n-gram předmětné strany shody je vypočten zbytek po celočíselném dělení hodnotou 256, výsledek určí prvek v poli, který bude inkrementován o hodnotu 1;
3. pro každý n-gram testované strany shody je vypočten zbytek po celočíselném dělení hodnotou 256, výsledek určí prvek v poli, který bude dekrementován o hodnotu 1;
4. rozdíl stran je stanoven sečtením absolutních hodnot prvků v poli;
5. průnik je dán odečtením poloviny rozdílu stran od celkového počtu n-gramů na příslušné straně shody.

Filtrace

Filtrace odstraní ty shluky, které nesplňují minimální hodnotu vypočteného kritéria shody, minimální délku v n-gramech na předmětné straně shody anebo minimální délku na testované straně shody. Minima jsou vedena jako konfigurovatelné parametry služby *compare.filter_min_ngrams* a *compare.filter_equality_threshold*.

5.8.3 Zpracování výsledků a rekonstrukce textů

Výsledné shluky jsou seskupeny do nových struktur *DocumentComparisonResult*. Každá uchovává všechny shluky, které patří pod stejný testovaný dokument. Součástí struktury je i obsah dokumentu použitý k nalezení shod.

Před prezentací výsledků uživateli je nezbytné shody převést do požadovaného typu výstupu (zrekonstruovat text z jejich n-gramů). Typ výstupu je parametrem úlohy hledání shod. Rekonstrukci textu provádí služba *DocumentReconstructor*.

Volitelné typy výstupu:

`none` Textový obsah nalezených shod nebude vypsán.

`ngram_indices` Vypíše n-gramy vybrané z dat uložených ve struktuře jako posloupnost hexadecimálních čísel.

Příklad: *. . . ,00000113,00000114,00000001,. . .*

`ngram_strings` Zpětným vyhledáním v převodní tabulce převede n-gramy na slova.

Příklad: *. . . (dopis který)(formálně který)(formálně zahajovat). . .*

`text` Načte z korpusu stránky původního textu a vybere z nich podřetězce náležející nalezené shodě.

Příklad: *. . . dopis, kterým formálně zahajuje. . .*

`page` Vypíše celé stránky původního textu s vyznačenou shodou.[25]

Příklad: *. . . na* `\n«««««\n` *dopis, kterým formálně zahajuje* `\n»»»»»\n` *vůbec. . .*

Výstupem rekonstruktoru je řetězec prezentovaný uživateli společně s dalšími údaji o nalezených shodách. Ke každé instanci datové struktury se vypíše samostatná zpráva.

[25] Syntax vyznačené shody odpovídá značení konfliktů ve verzovacím programu git.

Výpis 5.4: Formát zprávy výstupu z úlohy hledání shod.

```
{
  "subject_uri": "{identifikace předmětného dokumentu}",
  "compared_uri": "{identifikace testovaného dokumentu}",
  "matches": [
    {
      "approx_eql": "kritérium shody",
      "subject_pages": "rozsah stran na předmětné straně",
      "compared_pages": "rozsah stran na testované straně",
      "subject_lines": "rozsah řádků na předmětné straně",
      "compared_lines": "rozsah řádků na testované straně",
      "subject_content": "výstup rekonstrukce předmětné strany",
      "compared_content": "výstup rekonstrukce testované strany"
    },
    ...
   ]
},
...
```

Konkrétní příklady výstupu úlohy hledání shod jsou uvedeny pod přílohou E.

5.8.4 Histogram nalezených shod

Součástí výstupu z úlohy hledání shod jsou data ke zhotovení histogramu zobrazujícího počet řádků s nalezenou shodou pro každou stranu předmětného dokumentu (napříč všemi nalezenými shodami). Histogram má využití v uživatelském rozhraní, jelikož podává rychlý přehled, ve kterých částech dokumentu existuje podezření na plagiát[26].

Pro každou stranu dokumentu jsou sečteny řádky, ve kterých byla nalezena shoda. Součty jsou vypsány zprávou na výstup společně s celkovými počty řádků na daných stranách (umožňuje zobrazení absolutních i relativních hodnot).

Výpis 5.5: Formát zprávy histogramu získaného z úlohy hledání shod.

```
[
  {
    "matched":"řádky s nalezenou shodou",
    "total":"celkový počet řádků"
  },
  ...
]
```

[26]Například shody umístěné v přílohách nebo bibliografii pravděpodobně nebudou plagiátem.

5.9 Konfigurace aplikace

Aplikační služba pro své fungování vyžaduje řadu parametrů konfigurovatelných při instalaci v závislosti na prostředí, ve kterém má aplikace běžet. Práci s konfigurací zprostředkovává statická třída `Configuration`, která načítá strukturu aplikačních parametrů z konfiguračního JSON souboru (`config.json`) umístěného v kořenové složce repozitáře řešení. Součástí zdrojových kódů aplikace je i šablona pro jeho vytvoření `daemon/config.json.dist` (příloha C.1).

Načtené parametry jsou rozděleny do sekcí a zpřístupněny objektům aplikace přes metody `Configuration::Get` a `Configuration::GetVector`. Hodnoty jsou k dispozici po předání řetězcové cesty, která užívá znaku tečky k oddělení cílové sekce.

Soupis parametrů vyžadovaných službou:

`comm.host` - IP adresa pro socket TCP serveru;
`comm.port` - port pro socket TCP serveru;
`comm.connection_limit` - limit počtu obsluhovaných klientských spojení;

`pdf.parser` - cesta k aplikaci pro zpracování PDF souborů;
`pdf.config` - cesta ke konfiguraci pro zpracování PDF souborů;
`pdf.max_pages` - limit počtu stran indexovaného dokumentu;
`pdf.max_characters` - limit počtu znaků indexovaného dokumentu;

`stopwords.list` - seznam stop-slov;

`lemma.bin` - cesta k lemmatizátoru;
`lemma.dict` - cesta k lemmatizačnímu slovníku;

`ngrams.size` - velikost n-gramů;
`ngrams.chunking_unit` - krok vkládání značek zahajujících dokumentové díly;

`mongo.host` - adresa MongoDB serveru;
`mongo.user` - uživatel k MongoDB;
`mongo.password` - heslo k MongoDB;
`mongo.dbname` - jméno databáze;

`api.base_url` - kořenová URL k API interních dokumentů;
`api.auth_basic` - přístupové údaje k API interních dokumentů;

`compare.cuda_blocks` - počet CUDA bloků použitých při hledání shod;
`compare.cuda_threads` - počet CUDA vláken v každém bloku při hledání shod;
`compare.batch_size` - velikost dávky v dokumentových dílech;
`compare.batch_chunking_limit` - limit velikosti dokumentových dílů v dávce;
`compare.clustering_threshold` - práh shlukovací metriky;

`compare.filter_min_ngrams` - minimální délka shody v n-gramech;
`compare.filter_equality_threshold` - minimální kritérium shody;
`compare.comparison_concurrency` - maximální počet běžících detekčních úloh;

`ignored.norm_len` - normovaná délka ignorované věty;
`ignored.min_len` - minimální délka ignorované věty;
`ignored.batch_size` - velikost dávky při synchronizaci ignorovaných vět.
`ignored.thread_ngrams` - počet n-gramů na vlákno při aplikaci ignorovaných vět.

5.10 Implementované typy úloh

Následuje přehled úloh podporovaných aplikační službou. V závorce je vždy uveden alternativní název pro spuštění aplikačním klientem.

TaskCheckState (`task_check_state`) Slouží pro kontrolu stavu aplikační služby ze strany klienta. Vypíše informace o délce provozu aplikace, frontě úloh, využitých prostředcích, velikosti korpusu a dostupné grafické kartě.

TaskCorpusGet (`task_corpus_get`) Vypíše z korpusu metadata o dokumentu podle předané URI identifikace. Volitelně umí zobrazit i text stránek dokumentu. V případě neexistence dokumentu vyhodí zachytitelnou výjimku.

TaskCorpusList (`task_corpus_list`) Vypíše seznam dokumentů indexovaných v korpusu na základě předaných limitů (stránkování).

TaskDetect (`task_detect`) Spustí proces hledání shod pro předmětný dokument dle předané URI a s požadovaným typem výstupu.

TaskIgnoredSync (`task_ignored_sync`) Synchronizuje uložené ignorované věty se vzdálenou databází.

TaskIndexExternal (`task_index_external`) Zaindexuje do korpusu dokument umístěný na lokálním disku pod předanou cestou. Volitelně zaktualizuje existující dokument.

TaskIndexExternalDirectory (`task_index_external_directory`) Provede indexaci všech kompatibilních (PDF) souborů z adresáře na předaném umístění. Volitelně zaktualizuje existující dokumenty.

TaskIndexInternal (`task_index_internal`) Zaindexuje do korpusu soubor z interní API VUT identifikovaný svým ID a rokem nahrání. Volitelně zaktualizuje existující dokument.

TaskIndexInternalAll (`task_index_internal_all`) Spustí indexaci dostupných interních dokumentů pro ročník předaný parametrem. Volitelně zaktualizuje dokumenty.

6 Testy detektoru

V kapitole jsou uvedeny výsledky provedených testů na řešení detektoru plagiátů. Testy probíhaly na malém testovacím korpusu připravených anotovaných vzorků a na rozšířeném korpusu obsahujícím reálná data.

6.1 Testovací hardware a konfigurace

Hardwarové a softwarové vybavení použité pro běh testů je následující:

- CPU Intel Core i5-7400 (4 vlákna po 3GHz);
- RAM Crucial 16GB DDR4;
- SATA SSD Kingston, 480GB;
- GPU Nvidia GTX 1660S, arch. Turing, 6GB GDDR6, 22SM[1] x 1024 vláken;
- Host OS Fedora 33 Workstation Edition, 64-bit;
- CUDA version 11.2, driver 460.32.03;
- služba sestavena pro Release (produkční) konfiguraci;
- dále software dle souboru Dockerfile tvořící obraz kontejneru aplikační služby.

Není-li u daného testu uvedeno jinak, pak je pro testování použita aplikační konfigurace odpovídající nalezenému optimu. Konkrétně:

- délka n-gramu 2 slova;
- 22 CUDA bloků pro detekční úlohu;
- 512 CUDA vláken pro každý blok detekční úlohy;
- velikost dávky minimálně 11 000 dokumentových dílů;
- limit pro délku dokumentového dílu 512 n-gramů;
- práh pro metriku shlukování 100;
- minimální délka shody 16 n-gramů;
- minimální aproximovaná shoda 0.35.

6.2 Testování na anotovaném korpusu vzorků

Za účelem ladění a kvalitativního testování detekčního algoritmu byla sestavena databáze testovacích vzorů (*sample*) a z nich odvozených dokumentů (*specimen*) ve složce projektu */daemon/samples*. Odvozené dokumenty jsou k dispozici ve formátu PDF pro použití s detekčním algoritmem a také ve formátu pro editor typu Word, ve kterém byly vytvořeny.

[1]Streamovacích multiprocesorů. Fyzické umístění, procesor, na kterém běží logické bloky spuštěných CUDA kernelů (funkcí).

Word dokumenty zároveň obsahují i anotace formou komentářů, ve kterých jsou popsány úseky významné pro detekci (vytvořené přímým plagiátorstvím nebo parafrází testovacích vzorů).

Anotovaný korpus má dokumenty obsahující:

sample_1 - první testový vzor, bakalářskou práci [2] ;
sample_2 - druhý testový vzor, bakalářskou práci [3];
sample_3 - třetí testový vzor, úseky tvořené ignorovanými větami a úseky převzaté ze zpravodajských portálů;
specimen_1 - dva odstavce vzniklé přímým plagiátorstvím ze vzoru sample_1;
specimen_2 - text shodný se specimen_1, odstavce opatřeny citací;
specimen_3 - odstavce z předchozích *specimen* parafrázované a bez citací;
specimen_4 - přímou kopii sample_1 editovanou v prostředí Word s odlišným formátováním a stránkováním;
specimen_5 - dva odstavce ze sample_2 vzniklé přímým plagiátorstvím, jeden odstavec parafrázovaný;
specimen_6 - tři navzájem prokládané úseky, z nichž první je opatřen citací, každý úsek je tvořen dvěma odstavci ze sample_1 a dvěma ze sample_2;
specimen_7 - dvakrát jeden a tentýž úsek ze sample_2, jednou jako přímou kopii a jednou parafrázovaný;
specimen_8 - doslovnou kopii většiny textu ze vzoru sample_3;

Korpus je navržen tak, aby jen vzory (*sample*) sloužily jako předmětné dokumenty pro hledání shod. Odvozené dokumenty mají mezi sebou navzájem shodné části, jejichž detekce však není předmětem testu[4].

Po indexaci uvedených dokumentů[5] je do paměti uloženo 220 820 unikátních dvoj-slovných n-gramů a 50 779 lemmat. V příloze D.1 je zobrazen stav aplikace po nahrání anotovaného korpusu.

Příprava dokumentu provádějící odstranění ignorovaných vět trvá přibližně 4s a vzhledem k užité paralelizaci roste s velikostí dokumentu neznatelně. Časy uvedené u výsledků testů v sobě čas potřebný na přípravu dokumentu zahrnují.

[2] *KOŘÍNEK, Lukáš. Uživatelské rozhraní pro expertní systém.* Brno, 2019, 82 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce: doc. Ing. Václav Jirsík, CSc.

[3] (Se svolením autora) *HLAVINKA, Radek. Robotický stolní fotbal.* Brno, 2020, 50 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce: prof. Ing. Petr Pivoňka, CSc.

[4] Byť byla také testována a shodné části v ostatních odvozených dokumentech byly skutečně označeny jako plagiáty.

[5] A po předchozí synchronizaci 138 380 ignorovaných vět.

Část výstupu testů na anotovaném korpusu je uvedena v tabulce níže. Při užitých konfiguračních parametrech se podařilo nalézt veškeré přímé plagiáty a polovinu parafrází.

Dokument	Očekáváno	Nalezeno	Komentář
specimen_1	2	2	Oba odstavce přímého plagiátorství se vyskytly v nalezených shodách.
specimen_2	0	0	Odstavce opatřené citací byly při předzpracování úspěšně odstraněny a nebyly tudíž označeny jako plagiované.
specimen_3	2	1	Byl nalezen jeden parafrázovaný úsek ze dvou.
specimen_4	1	77	Vzhledem ke vzniklým rozdílům ve formátování a rozdělení dokumentu na díly došlo k vytvoření řady nezávislých shluků shod řádově až o stovkách n-gramů napříč okopírovaným dokumentem.
specimen_5	0	0	-
specimen_6	2	2	Správně rozpoznány přímé plagiáty po odstranění citovaných úseků.
specimen_7	0	0	-
specimen_8	0	0	-
Čas detekce: 8s			

Tab. 6.1: Výsledky testů na vzoru *sample_1* (anotovaný korpus).

Výsledky testů na zbývajících vzorech *sample_2* a *sample_3* jsou přiloženy pod E.1.

Úpravou konfiguračních parametrů na větší citlivost dojde k nalezení zbývajících parafrází. Bylo odzkoušeno na minimální délce shluku 10 n-gramů a minimální aproximované shodě 0,1.

6.3 Testování na rozšířeném korpusu

Za účelem testování zatížitelnosti aplikace při velkých datových tocích a odladění úkonů předzpracování na reálných datech byl sestaven rozšířený korpus obsahující:

- obsah anotovaného korpusu;
- závěrečné práce VUT (2020);
- závěrečné práce VUT (2018);
- prvních 100 dokumentů závěrečných prací VUT (2016).

Indexovaná data rozšířeného korpusu tvoří celkem 4 597 dokumentů o 9 106 625 unikátních n-gramech a 718 733 unikátních lemmatech. Na tabulce následující níže je zobrazena doba trvání detekčního procesu v závislosti na počtu indexovaných n-gramů, množství nalezených shod a komentář k nim. Příloha D.2 zobrazuje stav aplikace po nahrání rozšířeného korpusu.

Dokument	N-gramů	Čas [s]	Nálezů	Komentář
sample_3	395	5	3	Nalezeny stejné shody jako u anotovaného korpusu.
sample_2	5 412	16	180	Vyjma shod z anotovaného korpusu nalezeny shody se semestrální prací patřící k této vzorové závěrečné práci (charakter parafrází) a shody v obsahu dokumentu.
sample_1	10 756	28	275	Nalezeny shody se semestrální prací a dále přímo s interní reprezentací práce (obě odevzdány v akademickém roce 2018).
2018:113078	17 206	41	946	Shody v seznamu literatury, kterou se nepodařilo odstranit.

Tab. 6.2: Výsledky testů na rozšířeném korpusu.

Z výsledků lze konstatovat, že doba trvání detekčního procesu roste s narůstající velikostí předmětného dokumentu přibližně lineárně.

Řada shod vzniká na úrovni indexovaných semestrálních prací nebo neúspěšně odstraněné literatury. Taktéž neúspěšně odstraněné přílohy mají tendenci v pracích z určité oblasti (např. oceňování nemovitostí) vykazovat shodné formální prvky.

6.4 Vybrané časové testy

Na tabulkách níže jsou uvedeny výstupy testů sloužících ke stanovení optimální konfigurace detektoru (z pohledu rychlosti zpracování detekční úlohy).

Bloků	Vláken	Min. dílů v dávce	Dávek	Čas celkem [s]
4	256	800	98	65
4	512	1 800	44	46
4	1 024	3 800	21	37
8	256	1 800	44	37
8	512	3 800	21	26
8	1 024	7 900	10	23
16	256	3 800	21	22
16	512	7 900	10	18
16	1 024	16 000	5	58
22	256	5 400	15	18
22	**512**	**11 000**	**8**	**17**
22	1 024	22 300	4	78
32	256	7 900	10	18
32	512	16 000	5	32
32	1 024	32 500	3	65

Tab. 6.3: Čas detekce na předmětném dokumentu sample_2 v závislosti na počtu CUDA bloků a vláken (rozšířený korpus).

Limit velikosti dílu [n-gramů]	Dávek	Čas celkem [s]
256	14	17
512	**7**	**16**
1 024	4	17
2 048	**2**	**16**
4 096	2	22
8 192	1	21

Tab. 6.4: Čas detekce na předmětném dokumentu sample_2 v závislosti na limitu velikosti dokumentového dílu v dávce (rozšířený korpus).

Dle výsledků testů je nejlepšího výkonu dosaženo použitím 22 CUDA bloků po 512 vláknech při limitu velikosti dokumentového dílu 512 nebo 2 048 n-gramů.

7 Požadavky na sestavení a běh aplikace

Kapitola uvádí výčet požadavků na běh aplikace a seznam kroků pro spuštění řešení na cílovém hardware.

7.1 Systémové požadavky

Na základě testů provedených na rozšířeném korpusu (kap. 6), údajích z databáze a výstupů UNIXových nástrojů (*top*, *nvidia-smi*) byly stanoveny požadavky pro nasazení detektoru plagiátů. Požadavky se vztahují na aplikační službu a MongoDB korpus. Nároky aplikačního klienta jsou zanedbatelné. Aplikace pro běh vyžaduje:

- vícejádrový procesor (minimálně čtyři jádra[1]), větší množství jader umožní spuštění více úloh najednou;
- 16GB operační paměti, při testování si aplikace vyžádala 2,1GB fyzické paměti a celkem až 9,9GB paměti virtuální[2];
- disk s dostatečným prostorem pro uložení korpusu (cca 25GB)[3]), nejlépe SSD pro kratší odezvy a rychlejší přenos dat;
- grafická karta podporující Nvidia CUDA [4] s minimálně 3GB VRAM paměti;
- operační systém s jádrem Linux;
- ovladače grafické karty podporující CUDA;
- kontejnerizační služba *podman* nebo MongoDB server a software instalovaný podle instrukcí v *Dockerfile*.

Počet souběžně běžících detekčních úloh je limitován hodnotou v konfiguračním souboru aplikace *compare.comparison_concurrency*. Běh více než jedné detekční úlohy naráz může vyžadovat další operační paměť a paměť grafické karty nad rámec uvedených hodnot. Vždy je nezbytné přizpůsobit velikost zpracovávaných dávek dostupnému hardware (z ohledu dostupné paměti i počtu vláken paralelně spustitelných na grafické kartě).

Indexační operace mají ze své povahy menší nároky na paměť, stále však vyžadují výpočetní výkon a provádějí velké množství čtení a zápisů na disk.

[1] Jedno jádro řídí úlohy aplikační služby, detekční úloha vyžaduje jedno jádro a indexace až dvě. Další jádro je vhodné ponechat pro MongoDB, klienta a systém.

[2] Lze očekávat, že při zaindexování úplného korpusu (všechny práce z VUT) budou paměťové nároky vyšší. Uvedená hodnota proto zachovává určitou rezervu. Skutečné nároky je třeba znova ověřit až při nasazené aplikaci.

[3] Pro 4 597 indexovaných dokumentů činí průměrná velikost uložených dat 160kB na dokument. V korpusu se předpokládá uložení až 150 000 dokumentů.

[4] Seznam podporovaných karet je k dispozici na adrese `https://developer.Nvidia.com/cuda-gpus`.

7.2 Instalace aplikace v kontejnerech

Doporučeným postupem instalace řešení je využití služby *podman* pro tvorbu obrazů a následné spuštění čtyř kontejnerů. Aplikační služba tvoří obraz podle instrukcí v souborech *daemon/Dockerfile.debug* a *daemon/Dockerfile.release*, aplikační klient taktéž (*client/Dockerfile*). Databáze se spouští z existujícího obrazu hostovaného ve webovém úložišti.

Před započetím instalace řešení je nutné připravit konfigurační soubor *config.json* s parametry aplikace a umístit jej do kořenového adresáře projektu. K jeho vytvoření lze využít šablony *daemon/config.json.dist* a pouze překontrolovat v něm umístěné hodnoty (zejména údaje pro připojení ke korpusu a přihlašovací údaje k interní API).

V repozitáři aplikace je obsažen skript *unix_install.sh* automaticky konající všechny nezbytné úkony až po spuštění aplikace (musí být splněny hardwarové i softwarové požadavky ze subkapitoly 7.1). Skript také pomocí proměnné prostředí *TDPD_RELEASE* umožňuje volbu mezi sestavením produkční konfigurace (*Release*) a konfigurace určené pro ladění (*Debug*). Debug konfigurace vytvoří a spustí obraz aplikačního klienta, spustí MongoDB databázi a připraví běžící vývojový kontejner aplikační služby. Sestavení služby je dále nutné již provést v tomto kontejneru ručně, přičemž se předpokládá použití *gdb* debuggeru[5]. Release konfigurace vychází z debug, avšak navíc provádí sestavení aplikační služby s optimalizací -O2[6] a její spuštění.

Ukázka spuštění instalačního skriptu produkční konfigurace:

```
> TDPD_RELEASE=1 bash unix_install.sh
```

V rámci skriptu probíhá:

- sestavení obrazu aplikační služby;
- sestavení obrazu aplikačního klienta;
- zastavení již běžících kontejnerů;
- spuštění nových kontejnerů;
- přepis standardního výstupu kontejnerů do konzole (zobrazení chyb).

Po spuštění aplikace dojde k automatické indexaci obsahu složky *daemon/samples* (anotovaného korpusu) a zobrazení stavu aplikace pomocí TaskCheckState do standardního výstupu.

[5] Aplikace byla laděna v prostředí MS Visual Studio Code připojeného přímo do vývojového kontejneru. Součástí repozitáře jsou úlohy pro VS Code umožňující sestavení a ladění na jeden klik.

[6] Parametr překladače gcc.

7.3 Instalace aplikace přímo v hostujícím systému

Za účelem přímé instalace řešení bez využití kontejnerů je možné využít příkazů v souborech Dockerfile jako návodu. Příkazy lze využít beze změn nebo je upravit pro kompatibilitu s hostujícím systémem (obraz aplikační služby vychází z distribuce Ubuntu, klient z Alpine Linux).

Orientační kroky k sestavení a spuštění aplikační služby:

1. příprava konfiguračního souboru;
2. instalace CUDA toolkit s kompilátorem *nvcc*;
3. instalace systémových utilit *git*, *fontconfig*, *make* a *gcc*;
4. sestavení *CMake* verze 3.19 nebo novější;
5. instalace správce balíků *vcpkg* (více níže);
6. instalace C++ knihoven *boost*, *curlpp* a *mongocxx* pomocí *vcpkg*;
7. stažení PDF parseru *Xpdf*[7];
8. sestavení služby skriptem *daemon/unix_build_daemon.sh*;
9. spuštění nově vytvořeného binárního souboru *tdpddaemon*.

Pro správu potřebných C++ knihoven je využit nástroj *vcpkg*[8]. Jedná se o správce balíčků umožňujícího snadnou instalaci (přeložení) cílových knihoven pro zvolenou platformu. Software je distribuován pod MIT licencí a dovede pracovat s nástrojem *CMake*, jenž je použit k sestavení aplikace.

Postup při instalaci potřebných balíčků:

Výpis 7.1: Postup při instalaci knihoven nezbytných pro přeložení aplikace.

```
git clone https://github.com/microsoft/vcpkg
cd vcpkg
vcpkg install boost-process:x64-linux
vcpkg install boost-property-tree:x64-linux
vcpkg install mongo-cxx-driver:x64-linux
vcpkg install curlpp:x64-linux
vcpkg integrate install
```

Dále k sestavení klienta je třeba nainstalovat NodeJS (verze 14 a novější), přejít do adresáře klienta a spustit instalaci závislostí příkazem *npm install*. Nezávisle na klientovi a službě je také třeba provozovat MongoDB server pro korpus a pomocnou paměť aplikace.

[7]Aplikace vyžaduje také lemmatizátor majka, ten je však v rámci repozitáře přiložen a není třeba jej zvlášť instalovat.

[8]https://github.com/microsoft/vcpkg

8 Použití aplikačního klienta

Kapitola dává krátký návod k použití aplikačního klienta, jehož implementační část je popsána v kapitole 5.2.2. Předpokládá se, že koncový uživatel nebo nadřazený systém bude s detektorem plagiátů komunikovat výhradně skrze tohoto klienta.

8.1 API rozhraní

Klient disponuje rozhraním REST API, přes které je možné spouštět úlohy služby a číst jejich výstup pro další zpracování. Úlohy lze spouštět v synchronním nebo asynchronním režimu. Seznam dostupných úloh je uveden v kapitole 5.10.

Synchronní režim spustí úlohu na aplikační službě a poté okamžitě přeposílá její výstup do HTTP odpovědi jako proud (*stream*) dat. Předávání výstupu z úlohy probíhá po částech tak, jak jej služba vytváří (v reálném čase). Spojení je opatřeno časovým limitem 60 sekund, během kterého musí dojít k předání nového výstupu ze služby do klienta, aby nedošlo k jeho ukončení.

V případě použití asynchronního režimu je v HTTP odpovědi vrácen pouze unikátní číselný identifikátor úlohy, pomocí něhož je možné později výstup úlohy vyzvednout (celý výstup úlohy najednou, nikoliv po částech).

Spuštění úlohy v synchronním režimu

Ke spuštění synchronní úlohy je třeba zavolat níže uvedený cíl požadavku, předat mu název úlohy (alternativní název ze seznamu 5.10) a argumenty.

`POST /{úloha}`?format=`{formát}`&args[]=`{argument}`&args[]=`{argument}`&...

Argumenty úlohy se předávají přes parametry v rámci URL dotazu jako pole řetězců *args*. Zavolání úlohy s chybnými argumenty vyvolá nápovědu umožňující zjistit, jaké argumenty daná úloha vyžaduje.

Volitelně lze také uvést argument *format* pro volbu formátu výstupu aplikační služby. Dovolené hodnoty jsou „json“ (výchozí, řádkově oddělené JSON objekty) a „plain“ (prostý text).

Spuštění úlohy v asynchronním režimu

Pro zavolání asynchronní úlohy je nezbytné k uvedenému volání výše přidat argument *async* o hodnotě „1“.

`POST /{úloha}`?async=1&args[]=`{argument}`&args[]=`{argument}`&...

Ukázková odpověď pro formát JSON obsahující ID spuštěné úlohy:

```
{
   "id": "516178c2a26af1"
}
```

K získání výstupu asynchronně spuštěné úlohy slouží další cíl požadavku, který předpokládá uvedení ID v rámci cesty URL.

`GET` /results/`{ID}`

Zavoláním bude navrácen kompletní výstup z úlohy včetně případných chybových zpráv. Pokud úloha zatím nebyla dokončena, bude navrácen kód HTTP 202 - přijato (*accepted*).

Výpis 8.1: Ukázka synchronního volání úlohy TaskCorpusGet přes aplikačního klienta (utilitou *curl*).

```
curl -X POST "http://127.0.0.1:3000/task_corpus_get?
args[]=samples/sample_1.pdf&format=plain"
---
RESULT: Retrieved document information.

uri: samples/sample_1.pdf
indexed_at: 07.04.2021 18:41:48
updated_at: 07.04.2021 18:41:48
total_pages: 83
indexed_pages: 56
ngram_count: 10756
original_filetype: PDF
original_encoding: UTF-8
```

8.2 Demonstrační grafické rozhraní

Součástí klienta je i webová stránka poskytující grafické uživatelské rozhraní pro nahlížení na výstup z detekčního procesu. Stránka byla vyvinuta pouze za účelem ladění a demonstrace řešení, nikoliv k praktickému užití. Předpokládá se, že při nasazení detektoru plagiátů do nadřazeného systému bude implementováno nové rozhraní na míru anebo využito existující.

Aby došlo k vyvolání webového rozhraní, je nezbytné spustit asynchronní detekční úlohu ve formátu JSON a s argumentem „page“ (volí typ výstupu úlohy s výpisem stran).

Předpis pro spuštění úlohy:

`POST` /task_detect?args[]=`{URI dokumentu}`&args[]=page&async=1

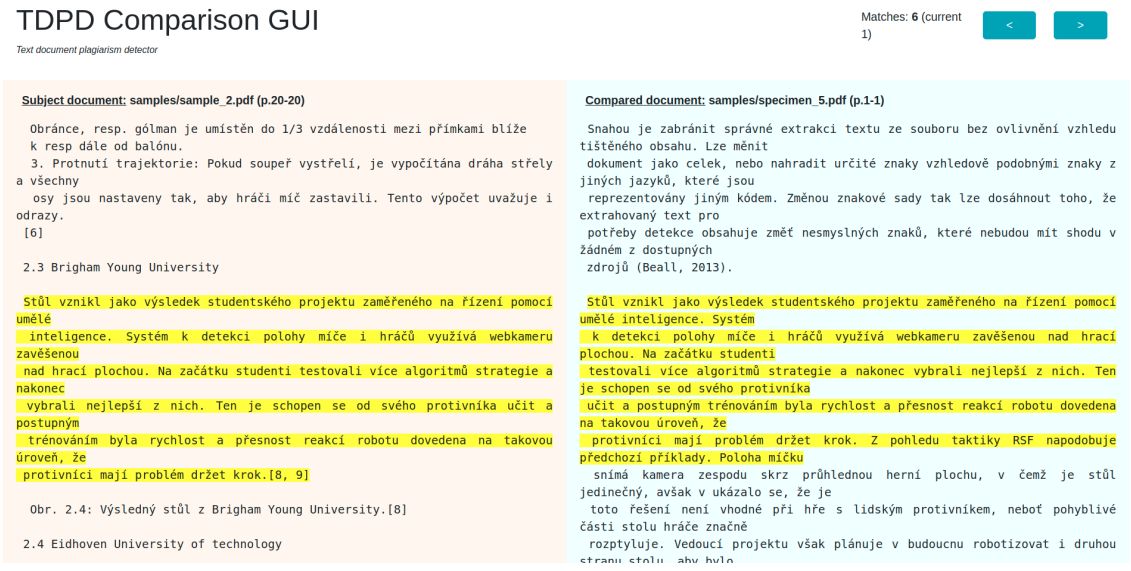
Získané ID se po dokončení úlohy vloží do webového prohlížeče:

`GET` /comparison/`{ID}`

Dojde k otevření webové stránky s přehledným zobrazením nalezených shod. Prvky umístěné na stránce shora jsou:

- hlavička zobrazující informace o nalezené shodě;
- oranžově podbarvený levý panel s předmětným dokumentem;
- modře podbarvený pravý panel s porovnaným dokumentem;
- histogram nalezených shod v předmětném dokumentu.

Nalezená shoda je zvýrazněna žlutou barvou ve vypsaném textu, který odpovídá stránkám daného dokumentu. Mezi jednotlivými shodami lze přecházet tlačítky umístěnými v hlavičce.



Obr. 8.1: Náhled na demonstrační grafické rozhraní klienta (zobrazení shod).

Čitelnější snímky vybraných shod jsou dále přiloženy pod přílohou E.4.

9 Připomínky zadavatele

Na konzultacích byla v průběhu vývoje získávána zpětná vazba od konzultanta práce (zástupce zadavatele) za účelem umožnit integraci aplikace do školní infrastruktury.

Přijaté připomínky byly zapracovány, aplikace byla zadavateli představena a poté jím schválena pro rozsah diplomové práce jako postačující (z aspektů funkčních i uživatelských). Jelikož řešení zatím nedospělo do fáze reálného nasazení, lze eventuálně očekávat další požadavky nad rámec práce.

Rozhraní aplikačního klienta Původním záměrem bylo vytvoření klienta jako programu dostupného z příkazové řádky, který by komunikoval se službou prostřednictvím pojmenovaných rour. Zadavatel však preferoval implementaci klienta v podobě HTTP serveru s REST API, který by mohl přímo přijímat požadavky ze sítě. Vzhledem k použití této technologie bylo zároveň vhodnější spojit klienta se službou prostřednictvím TCP socketů.

Lepší zvládání parafrází Prvotní verze aplikace využívala n-gramy o délce tří slov a vykazovala slabou úspěšnost detekce parafrází, a sice podle zadavatele i horší, než jakou disponuje dosavadní systém. Dalším vyladěním aplikace a snížením délky n-gramů na dvě slova došlo k navýšení schopnosti rozpoznávat parafráze.

Přesnější lokace shod Vodicí značky lokalizující pozici nálezu v dokumentu byly zpočátku implementovány pouze pro úroveň stránek. Dle zadavatele nebyla kombinace nalezené shody v podobě n-gramů a čísel odpovídajících stránek dostačující z pohledu uživatelské přívětivosti. Z toho důvodu byly přidány i vodicí značky na úrovni řádků umožňující přesnější lokaci nálezu.

Forma výstupu z aplikace Do korpusu se od počátku ukládala jen zpracovaná podoba dokumentů jako sled n-gramů. Ta umožnila rekonstrukci nalezených shod převodní tabulkou (varianta výstupu detekční úlohy *ngram_strings*), ne však plnohodnotné zobrazení nalezených shod na originálním textu. Zadavatel si přál výstup aplikace vylepšit, aby dosahoval alespoň úrovně dosavadního systému. Bylo tak přidáno ukládání nezpracovaných stránek do korpusu a jejich zpřístupnění jako forma výstupu se zvýrazněnými nálezy (varianty *text*, *page*).

Ignorované věty Dokumenty vkládané do korpusu od začátku používaly úkon předzpracování pro odstranění formálních částí na základě regulárních výrazů. Na CVIS je však vedena databáze ignorovaných vět, které se mohou vyskytnout na různých místech dokumentu. Zadavatel si přál tuto databázi použít. Algoritmus pro odstranění ignorovaných vět byl dodatečně implementován.

10 Možná rozšíření aplikace

Při návrhu bylo myšleno na možná rozšíření aplikace do budoucna. V následujících subkapitolách jsou shrnuty klíčové myšlenky, které by zdokonalily aplikaci v oblasti uživatelské použitelnosti i kvality a rychlosti detekčního procesu.

Sepsané podněty jsou seřazeny tak, že ty s větším potenciálním přínosem anebo snadnější implementací jsou uvedeny jako první, ty obtížnější nebo rozporuplnější jako poslední.

10.1 Vylepšení aplikačního klienta

Podpora nahrávání souborů skrze klienta

Aplikace umí indexovat nejen interní dokumenty z VUT, ale i lokální dokumenty načtené z disku. REST API aplikačního klienta však zatím neumožňuje libovolný soubor nahrát (umístit jej na disk serveru). Skrze API lze pouze zavolat funkci pro indexaci externích dokumentů, a to na soubor, který již na disku existuje.

Řešením je přidání nového cíle pro POST požadavek do API rozhraní klienta, který by přijal soubor v těle požadavku, detekoval formát souboru, uložil soubor na serverový disk do připravené složky a vrátil uživateli výslednou adresu souboru (URI identifikátor). V dalším požadavku by poté bylo možné zavolat externí indexaci na získaném URI identifikátoru.

10.2 Vylepšení aplikační služby

Indexace textů z jiných formátů

Aplikační služba umožňuje jen indexaci textů ve formátu PDF. Korpus by však bylo možné rozšířit i o soubory ve formátu prostého textu, dokumenty Microsoft Word a zpracování HTML dokumentů. Přidání těchto funkcí vyžaduje nalezení vhodných knihoven či externích programů a poté rozšíření úkonu předzpracování pro syntaktickou analýzu textu (*StagePDFParser*).

Indexace textů z webových zdrojů

Služba pro indexaci dokumentů (*DocumentIndexer*) je připravena na rozšíření, které bude umět indexovat i webové dokumenty. Nejvhodnější implementací je vytvoření nového úkonu pro pipeline předzpracování, *StageWebLoader*, který by stáhnul webovou stránku identifikovanou pomocí webové adresy (sloužící jako URI dokumentu)

do lokální kopie skrze existující třídu *Curl*. Zbytek úkonů v pipeline by bylo třeba vhodně přizpůsobit (není nutné odmazávat formální úseky závěrečných prací apod.).

Rozšíření pro více grafických karet

Při spouštění CUDA kernelů není zvoleno, která grafická karta se má využít. Vždy se tedy použije primární zařízení (řízeno grafickými ovladači nebo systémem).

Užití N nezávislých grafických karet by umožnilo zpracování většího množství dávek dokumentů (*DocumentMemoryBatch*) najednou, vlivem čehož by bylo možné dosáhnout až téměř N násobného zrychlení hledání shod. Z pohledu implementace se nabízí přidání funkce pro výběr aktuální karty do třídy *Cuda* podle jejího číselného indexu, nebo důmyslněji funkce pro uvolnění první volné grafické karty (rotování dostupných karet mezi běžícími úlohami, kdy např. jedna úloha používá maximálně 3 karty ze 6 dostupných).

Předpokladem je dostatečná datová (vstupně-výstupní) propustnost korpusu, aby bylo možné daný počet dávek načíst z disku dříve, než skončí hledání shod na předchozích dávkách. V případě nedostatečné propustnosti se nabízí navýšení limitu pro velikost dokumentových částí, což by prodloužilo dobu výpočtu na každé z karet a procesor by tak získal větší čas pro načtení nových dávek.

Srovnání dvou dokumentů vůči sobě navzájem

Úloha *TaskDetect* dovede pouze srovnat dokument vůči celému korpusu, nikoliv však srovnat jeden dokument s druhým, který by byl úloze také předán parametrem.

Vhodnou implementací by bylo vytvoření nového typu úlohy (*TaskDetectSingle*) a rozšíření služby *DocumentMemoryManager*, jež by pro detekční proces mezi dvěma dokumenty byla schopna vytvořit zvláštní typ dávky *DocumentMemoryBatch* obsahující pouze požadovaný testovaný dokument. Algoritmy srovnávacího procesu jsou nyní pevně svázány s úlohou *TaskDetect* a musely by být předem přemístěny do samostatného modulu.

Optimalizace přístupů do grafické paměti

Dle výsledků z kapitoly 6.4 dochází při použití velkého množství grafických vláken pro hledání shod ke znatelnému zpomalení. Možným důvodem pro toto zpomalení je neoptimální konkurenční přístup do grafické paměti. Algoritmus hledání shod nebyl během vývoje paměťově optimalizován.

Vývojářská příručka CUDA[1] uvádí soupis technik pro optimalizaci paralelních výpočtů, ze kterých by se dalo vycházet.

[1] K dispozici na `https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#optimizing-cuda-applications`.

Jedním z možných přístupů je také navázání velikosti dávky na počet vláken v grafickém výpočetním bloku. Dávky by byly menší a bylo by jich načítáno vícero najednou. Pro každý použitý výpočetní blok by byla vytvořena i samostatná kopie předmětného dokumentu. Nedocházelo by tak ke konkurenčnímu přístupu do VRAM mezi bloky. Toto uspořádání by navíc zjednodušilo přidání podpory vícera grafických karet.

Lepší nakládání s citovanými úseky

Citace jsou z textů odstraňovány již během indexace před vložením do korpusu. Tím, že dojde k odstranění citací ze všech dokumentů, není možné odhalit plagiáty v případě, kdy zdroj plagiovaného úseku je sám opatřen citací. Tato situace je znázorněna na příkladu níže.

Předmětný dokument s plagiátem:
CAN sběrnice tvoří pomyslnou páteř každého moderního vozu....
Testovaný dokument (zdroj pro plagiát, bude během indexace odstraněno):
Pomyslnou páteří každého moderního vozu je sběrnice CAN ... (Novák, 2018).

Řešením je zavedení nových vodicích značek pro začátek a konec citace, přičemž během indexace by došlo pouze ke vložení těchto značek do sledu n-gramů. Před začátkem hledání shod, tedy během přípravy dokumentu odstraňující ignorované věty, by se provedlo odstranění citovaných úseků pouze z předmětného dokumentu a nikoliv z dokumentů testovaných.

Uchování načtených dávek dokumentů v paměti

Načtené dávky s dokumentovými díly se mezi jednotlivými úlohami *TaskDetect* odlišují pouze tím, že se v nich nenachází předmětný dokument (nedochází tudíž ke srovnání předmětného dokumentu se sebou samým).

Za předpokladu dostatku dostupné operační paměti se nabízí možnost v ní ponechat načtené dávky z korpusu. Nemusely by tak být načítány z disku nové dávky pro každý detekční proces a neustále alokována a uvolňována paměť. Implementačně by bylo vhodné takovou úpravu provést na úrovni služby *DocumentMemoryManager*. Dále by bylo nutné provádět vynechání předmětného dokumentu na jiné úrovni (kupříkladu až při hledání shod) a zabezpečit přidání nově indexovaných dokumentů do dávek uchovaných v paměti (případně do nové dávky).

V případě, kdy by i paměť dostupných grafických karet disponovala dostatečnou kapacitou pro uchování dávek, bylo by možné tyto dávky mít připravené již na úrovni grafické paměti. Je však třeba počítat s dostatečnou rezervou, jelikož grafické karty mohou být využity i jinými úlohami (např. indexace).

Rozšíření o intrinsické přístupy

Řešení by bylo možné rozšířit i o intrinsickou detekci. Nabízí se implementace nového úkonu předzpracování (*StageIntrinsic*), který by na indexovaném dokumentu pro vhodná okna menší velikosti stanovil intrinsické příznaky. Z výstupu těchto oken by byly poté vhodnou metrikou vytvořeny větší shluky, obdobně jako se děje s nalezenými shodami během úlohy *TaskDetect*. Získané shluky by odpovídaly sekcím dokumentu s přibližně homogenními intrinsickými charakteristikami a s dalšími metadaty by byly uloženy do korpusu.

Před zahájením hledání shod by proběhlo nahlédnutí do intrinsických charakteristik dokumentu a v případě výskytu textových úseků (shluků, homogenních sekcí) s výrazně odlišnými charakteristikami by bylo uživateli na výstup předáno varování před možným plagiátorstvím z neznámého zdroje.

Vodicí značky kapitol

K vodicím značkám pro začátky stran a řádků by bylo možné přidat i vodicí značky začátků kapitol. Jejich další užití při hledání shod a zobrazení čísla a názvu kapitoly ve výstupu aplikace by mohlo dále vylepšit uživatelskou přívětivost a usnadnit kontrolu získaných nálezů.

Lokace začátků kapitol by bylo možné zjistit přímo ze software pro syntaktickou analýzu PDF, avšak pouze za předpokladu, že by software umožňoval čtení metadat. Alternativou je hledání nadpisů pomocí regulárních výrazů, což však nemusí být na reálných datech ve všech případech spolehlivé.

Implementace by si vyžádala změny v předzpracování a úloze *TaskDetect*.

Vylepšení vzdálenostní metriky

Vzdálenostní metrika sloužící jako kritérium pro tvorbu shluků z nalezených shod je charakteru min-max a za vývoje aplikace neproběhlo žádné měření, které by ji srovnalo s jinými metrikami. Z důvodu absence měření je možné, že jednodušší nebo naopak pokročilejší metrika by vykazovala lepší výsledky při tvorbě shluků.

Nabízí se užití standardizovaných metrik (například euklidovská nebo manhattanská vzdálenost). Vedoucí práce jako možný přístup navrhl i použití metriky s hysterezí po vzoru Cannyho detektoru[2] užívaného v oblasti zpracování obrazu. Byly by užity dvě odlišné hodnoty prahu, přičemž pro zahájení (vznik nového) shluku by musela vzdálenost sousedních shod splňovat přísnější práh, u již zahájeného shluku by mohl být práh nižší.

[2]Více např. zde `https://en.wikipedia.org/wiki/Canny_edge_detector`.

Synonymizace lemmatizovaného textu

Nalezení vhodného tezauru pro český jazyk by umožnilo implementaci nového úkonu předzpracování (*StageSynonymize*). Ten by byl aplikován na lemmatizovaný text a tokeny nahradil základními tvary synonym. Interní fungování úkonu by mohlo být shodné s lemmatizací.

Automatické vyhledávání webových zdrojů pro detekci

Je pravděpodobné, že podstatné množství plagiátů nebude pocházet ze závěrečných prací, ale ze zdrojů na internetu. Vzhledem k intrakorpální povaze řešení zůstanou takové plagiáty i nadále tímto nástrojem neodhaleny.

Jako nejschůdnější řešení pro eliminaci tohoto nedostatku se jeví automatické vyhledávání webových zdrojů na základě klíčových slov celého dokumentu nebo specifických sekcí. K extrakci klíčových slov je nachystán neimplementovaný úkon předzpracování *StageKeywords* a struktura *DocumentKeywords*. Je však třeba nejprve implementovat indexaci textů z webových zdrojů.

Vyhledávání by mohlo být spouštěno periodicky takovým způsobem, aby nebyla vyčerpána denní kvóta dotazů podle daného zprostředkovatele hledání (s cílem minimalizovat až zcela eliminovat spojené náklady dle smluvních podmínek).

Dokumenty pro vyhledání zdrojů se mohou vybírat náhodně nebo podle počtu hledání, která již byla s daným dokumentem provedena, a to s omezením pouze na interní dokumenty. Prvních N získaných nálezů z každého výsledku hledání by bylo zaindexováno do korpusu a detekční proces pro dokument, ze kterého byla klíčová slova získána, by byl opakován. Pozitivní nálezy by byly odeslány administrátorům k dalšímu prověření.

Výhodou nastíněného postupu je, že může být vykonáván postupně během času bez uživatelských zásahů a s přibývajícími dokumenty v korpusu z dotčených oblastí tak vylepšovat úspěšnost detektoru. Naopak hlavní nevýhodou je nárůst velikosti korpusu a nutnost zdokonalení implementovaných datových struktur i použitého hardware.

Závěr

V teoretické části práce je rešerše umožňující návrh detektoru plagiátů. Kapitola 1 uvádí základní pojmy z oblasti plagiátorství a citování. Následuje průzkum řešení používaných na VUT a ostatních vysokých školách, jehož zjištění jsou uvedena v kapitole 2. Následující kapitola 3 se již zaměřuje na algoritmy strojové detekce plagiátů a vysvětluje jednotlivé aspekty detekční úlohy od definice obecné úlohy a možných přístupů, přes způsoby předzpracování dokumentů, až po kritéria podobnosti dokumentů i shody jejich částí. Jsou uvedena i doporučení pro návrh detektorů plagiátů vycházející z předchozích prací.

Teorie pokračuje přehledem softwarových prostředků (kap. 4), na které dále navazuje praktická část práce. V rámci této praktické části byl navržen nový systém detekce plagiátů (kap. 5) vhodný pro využití (nejen) v rámci informačního systému VUT. Návrh je proveden v souladu s požadavky stanovenými s ohledem na budoucí využitelnost řešení, které jsou definovány v subkapitole 5.1. Primárním návrhovým kritériem je umožnění kontroly oproti co nejvyššímu počtu dokumentů v co nejkratším čase.

Navržený systém má intrakorpální charakter a indexuje vkládané dokumenty do vlastního korpusu (databáze). Indexace využívá vícekrokového předzpracování. Detekce plagiátů probíhá srovnáním obsahu předmětného dokumentu se zbytkem korpusu, přičemž výstupem je množina nalezených shod. Hledání shod probíhá na úrovni vektorů n sousedících slov (n-gramů) reprezentovaných celočíselnými indexy do převodní tabulky.

Mezi komponenty navrženého řešení patří aplikační služba spravující systémové prostředky a vykonávající operace indexace dokumentů i hledání shod. Služba je konfigurovatelná a využívá paralelizace na grafické kartě i na CPU (každá úloha má své samostatné vlákno). Další komponentou je tenký aplikační klient, webová aplikace zpřístupňující funkce aplikační služby přes rozhraní typu REST API. Klient komunikuje se službou prostřednictvím TCP socketů. Poslední komponentou řešení je MongoDB NoSQL databáze pro dokumentový korpus a pomocná data. Návrh počítá s překladem pro UNIXové systémy a s během jednotlivých komponent ve virtualizovaných kontejnerech usnadňujících budoucí integraci do webového systému.

Systém detekce plagiátů byl dle výše uvedeného návrhu implementován (také kap. 5), a to v jazyce C++ a s využitím platformy Nvidia CUDA pro akceleraci výpočtů na grafické kartě. Řešení běží v clusteru celkem čtyř kontejnerů. Hardwarové požadavky, potřebný software a instrukce k sestavení jsou uvedeny v kapitole 7. Popis REST API aplikačního klienta se nachází v kap. 8. Nad rámec budoucích potřeb (integrace do jiného systému) bylo do klienta přidáno webové uživatelské rozhraní umožňující nahlížení na výsledky detekčního procesu.

Za účelem ladění byla sestavena databáze dokumentů obsahující vzorové texty (bakalářské práce) a dále texty, ve kterých jsou části vzorových textů plagiované. Plagiáty jsou provedeny formou přímých kopií i parafrází. Konkrétní zdroj a umístění plagiátů v těchto textech jsou anotované komentáři zobrazitelnými v editoru typu Word.

Chování systému bylo během vývoje laděno na uvedené anotované databázi a také na rozšířené databázi reálných dat závěrečných prací z VUT (celkem 4 597 dokumentů). Bylo provedeno testování z hlediska úspěšnosti nalezení shod a časové testy měřící délku trvání hledání shod v závislosti na volbě parametrů služby.

Systém při testování odhalil všechny přímé plagiáty a část parafrází v anotované databázi. Na rozšířené databázi byly odzkoušeny i náhodně vybrané dokumenty, přičemž došlo k nálezům zejména v seznamu literatury, přílohách nebo obsahu. Také byly nalezeny shody pocházející ze semestrálních prací patřících ke vzorovým dokumentům.

Z časového hlediska se podařilo dosáhnout srovnání dokumentu o 83 stranách oproti 4 597 dokumentům v rozšířené databázi během 28 sekund (na testovacím hardware, kap.6.1). Trvání detekčního procesu je přibližně lineárně závislé na délce předmětného dokumentu. Z výsledků testů dále vyplývá, že současný algoritmus nedokáže efektivně využít výpočetních kapacit grafické karty, jelikož při navýšení počtu použitých grafických vláken nad nalezené optimum dochází k poklesu výkonnosti. Předpokládaným důvodem je chybějící optimalizace přístupů do grafické paměti. Výsledky testů jsou uvedeny v kapitole 6.

Indexace reálných dat zviditelnila nedostatky v úkonech předzpracování, jelikož každý dokument může mít mírně odlišné formátování. V některých případech tak selhává odstranění citací nebo formálních částí dokumentů (začátek po obsah, konec od literatury dále). Z pohledu kvality strojového zpracování budoucích prací lze doporučit cílení na algoritmickou rozpoznatelnost nadpisů kapitol, začátku práce až po obsah, konce práce od literatury dále, ohraničení výpisů a zavedení jasných pravidel pro kladení citačních odkazů do relevantních směrnic VUT.

Řešení detektoru bylo během vývoje několikrát představeno zadavateli práce, jehož připomínky byly zapracovány a jsou popsány v kapitole 9.

Rychlost a kvalita výstupu detekčního procesu jsou dle názoru zástupce zadavatele práce dostačující k tomu, aby mohla aplikace v rámci školní infrastruktury sloužit jako alternativní nástroj pro srovnávání závěrečných prací. Před reálným nasazením je třeba nalézt vhodné nastavení aplikační služby a ověřit funkčnost na úplném korpusu (přes 100 000 dokumentů). V kap. 10 jsou popsány náměty na další vylepšení. Optimalizace přístupů do grafické paměti a přidání podpory více grafických karet by urychlilo detekční proces tak, aby bylo možné zpřístupnit detekci plagiátů na počkání z rozhraní informačního systému (například i studentům).

Literatura

[1] *ANON. Zákon č. 121/2000 Sb.: Zákon o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon)*. In: 121/2000. 2000, 36/2000. Dostupné také z: https://www.zakonyprolidi.cz/cs/2000-121

[2] *KOBATH, M. Detekce plagiátů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2015. 59 s. Vedoucí bakalářské práce Ing. Petr Honzík, Ph.D.

[3] *BIERNÁTOVÁ, Olga a Jan SKŮPA. Bibliografické odkazy a citace dokumentů: dle ČSN ISO 690 (01 0197) platné od 1. dubna 2011* [online]. 2011, , 27 [cit. 2020-10-01]. Dostupné z: https://www.citace.com/CSN-ISO-690.pdf

[4] *POLÁKOVÁ, Barbora. Americké citační styly [American citation styles]*. Praha, 2007-07-11. 43 s. VIII s. příl. Bakalářská práce. Univerzita Karlova v Praze, Filozofická fakulta, Ústav informačních studií a knihovnictví. Vedoucí bakalářské práce PhDr. Eva Bratková. Dostupné z: https://is.cuni.cz/webapps/zzp/download/130052004

[5] *Citation Styles: APA, MLA, Chicago, Turabian, IEEE: IEEE Style. University of Pittsburgh - Library system* [online]. University of Pittsburgh, 29.9.2020 [cit. 2020-10-06]. Dostupné z: https://pitt.libguides.com/citationhelp/ieee

[6] *KRATOCHVÍL, Jan. JAK CITOVAT* [online]. 1. Masarykova Univerzita, 2014 [cit. 2021-04-03]. Dostupné z: https://kuk.muni.cz/animace/eiz/pdf.php?file=publikacni_etika/citace.pdf

[7] *O Theses. Theses.cz: Vysokoškolské kvalifikační práce* [online]. Masarykova univerzita [cit. 2021-04-03]. Dostupné z: https://theses.cz/about_theses.pl

[8] *Odevzdej.cz: Seminární a školní práce* [online]. Masarykova Univerzita, 2021 [cit. 2021-04-03]. Dostupné z: https://odevzdej.cz/

[9] *Turnitin. Knihovna mff uk* [online]. Univerzita Karlova, Matematicko-fyzikální fakulta, 2021 [cit. 2021-04-03]. Dostupné z: https://www.mff.cuni.cz/cs/knihovna/informace/navody-pro-studenty/turnitin

[10] *Turnitin Similarity: Robust Plagiarism Checker. Turnitin* [online]. California, 2021 [cit. 2021-04-03]. Dostupné z: https://www.turnitin.com/products/similarity

[11] *PŘIBIL, Jiří. Efektivní metody detekce plagiátů v rozsáhlých dokumentových skladech* [online]. Jindřichův Hradec, 2010 [cit. 2020-10-08]. Dizertační práce. Vysoká škola ekonomická v Praze - Fakulta managementu v Jindřichově Hradci. Vedoucí práce Radim Jiroušek. Dostupné z: http://www.vse.cz/vskp/show_file.php?soubor_id=1237269

[12] *POTTHAST, Martin. Overview of the 6th International Competition on Plagiarism Detection* [online]. 2014, 32 [cit. 2020-09-28]. Dostupné z: http://ceur-ws.org/Vol-1180/CLEF2014wn-Pan-PotthastEt2014.pdf

[13] *SUCHOMEL, Šimon a Michal BRANDEJS. Determining Window Size from Plagiarism Corpus for Stylometric Features.* In Mothe, Josiane and Savoy, Jacques and Kamps, Jaap and Pinel-Sauvagnat, Karen and Jones, GarethJ.F. and SanJuan, Eric and Cappellato, Linda and Ferro, Nicola. Experimental IR Meets Multilinguality, Multimodality, and Interaction. Toulouse, France: Springer International Publishing, 2015. s. 293-299, 7 s. ISBN 978-3-319-24026-8. doi:10.1007/978-3-319-24027-5_31. Dostupné také z: https://is.muni.cz/publication/1317554/cs

[14] *What is NoSQL? MongoDB* [online]. 2020 [cit. 2020-10-21]. Dostupné z: https://www.mongodb.com/nosql-explained

[15] *An Even Easier Introduction to CUDA. Nvidia Developer* [online]. 2020 [cit. 2020-10-21]. Dostupné z: https://developer.Nvidia.com/blog/even-easier-introduction-cuda/

[16] *CUDA FAQ. Nvidia Developer* [online]. 2020 [cit. 2020-12-26]. Dostupné z: https://developer.Nvidia.com/cuda-faq/

[17] *ŠMERK, Pavel. Fast Morphological Analysis of Czech.* In: SOJKA, Petr a Aleš HORÁK. Proceedings of Third Workshop on Recent Advances in Slavonic Natural Language Processing: RASLAN 2009. Brno: Masaryk University, 2007, s. 13-16. ISBN 978-80-210-5048-8.

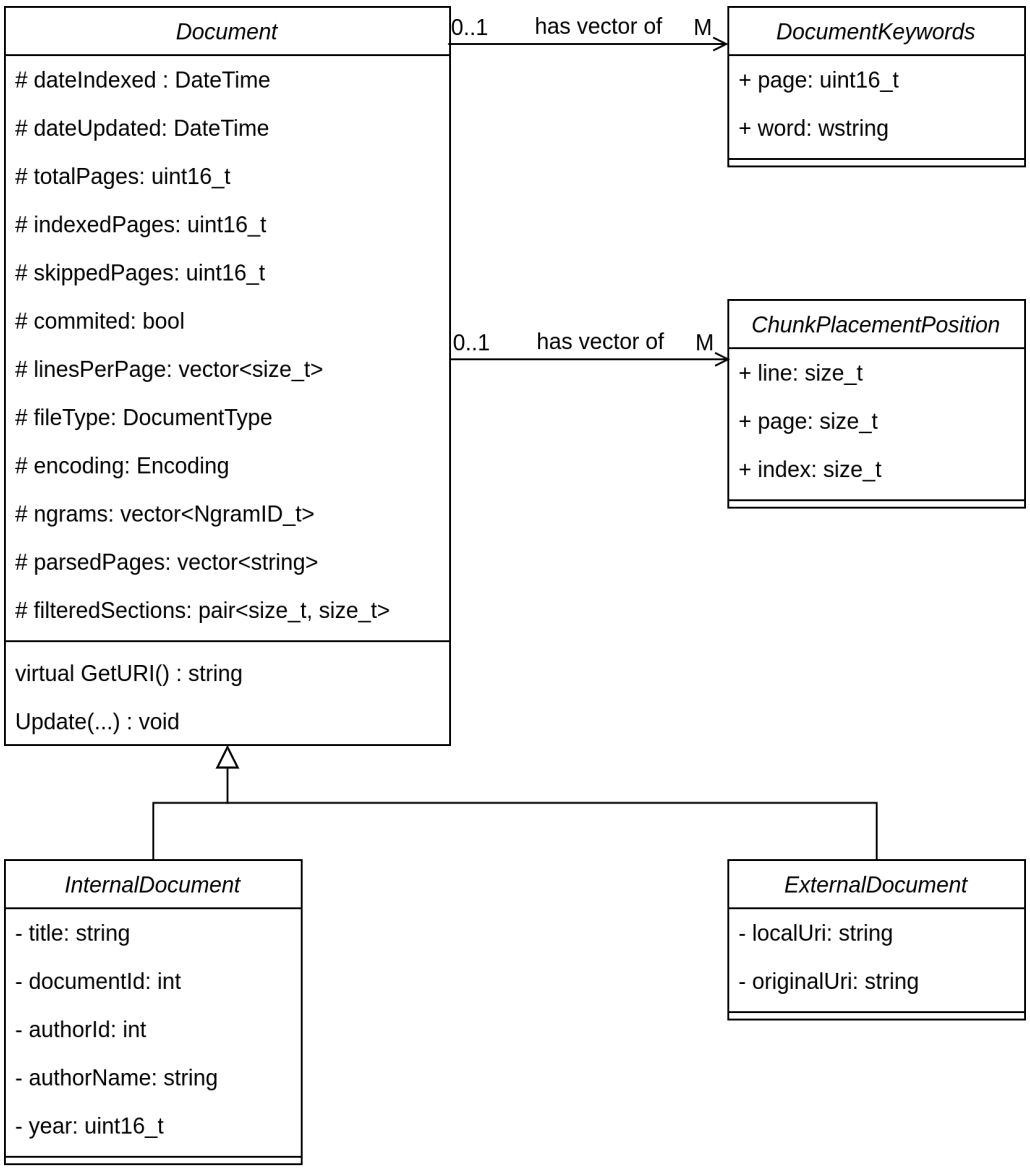
Seznam příloh

A Obsah přiloženého CD

- `/` ..... kořenový adresář přiloženého CD
  - `docs` ..... adresář se zdrojem textu práce (LaTeX)
  - `solution` ..... adresář repozitáře zdrojového kódu řešení
    - `client` ..... soubory aplikačního klienta
    - `daemon` ..... soubory aplikační služby
      - `example_build` ..... sestavená aplikační služba
      - `samples` ..... adresář s anotovaným korpusem
      - `vendor` ..... adresář s externím SW pro indexační úlohy
      - `config.json.dist` ..... šablona konfiguračního souboru
      - `unix_build_daemon.sh` ..... skript pro sestavení aplikační služby
    - `unix_install.sh` ..... instalační skript kontejnerizované verze
    - `unix_compose.yml` ..... definice kontejnerů
  - `xkorin12.pdf` ..... text práce v PDF

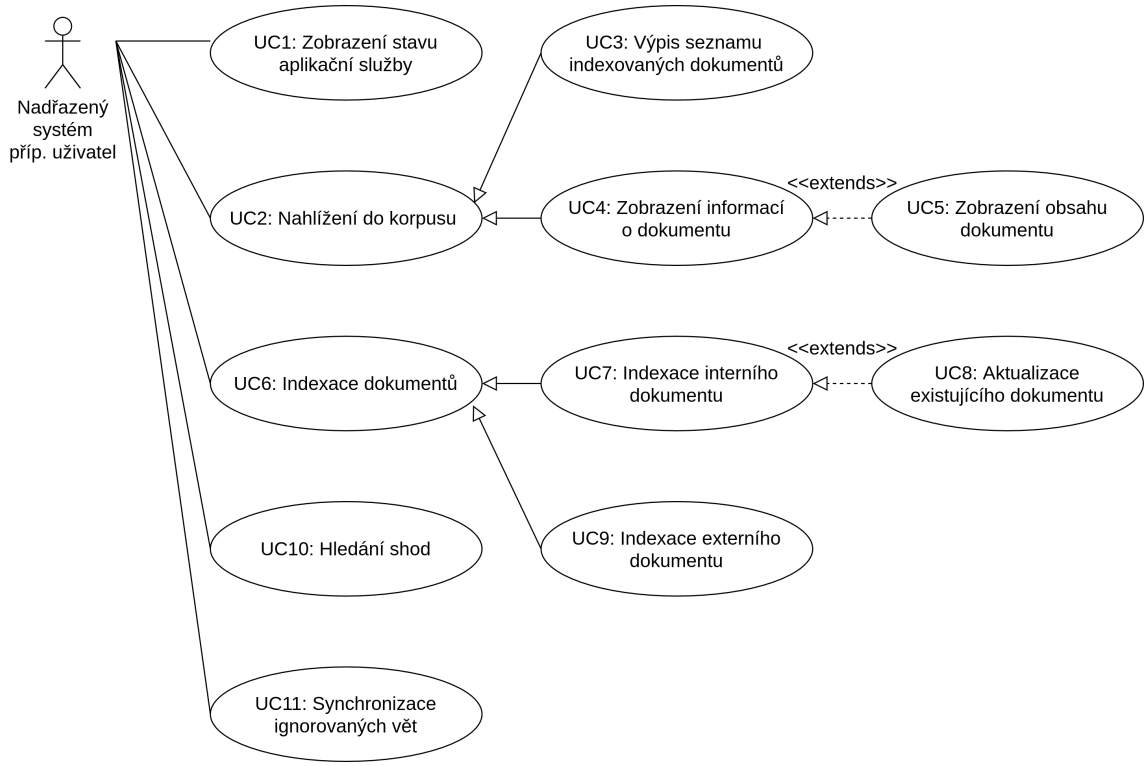
Sestavení v adresáři */daemon/example_build* vyžaduje podporovaný hardware, instalaci potřebného software, databázový server a správně nastavený konfigurační soubor. Více v kapitole 7. Doporučeno je použití kontejnerizované verze.

B Diagramy k návrhu aplikační služby



Obr. B.1: Diagram modelových tříd dokumentu.

V diagramu modelových tříd nejsou uvedeny jmenné prostory datových typů, konstruktory, destruktory a funkce pro získání i nastavení hodnot chráněných atributů.



Obr. B.2: Diagram využití služeb detektoru plagiátů.

C Implementace aplikační služby

C.1 Výchozí konfigurace aplikační služby

Výpis C.1: Šablona pro konfiguraci aplikace.

```
{
  "comm": {
    "host": "0.0.0.0",
    "port": 4096,
    "connection_limit": 50
  },
  "pdf": {
    "parser": "./vendor/xpdf/pdftotext.exe",
    "config": "./vendor/xpdf/config.xpdfrc",
    "max_pages": 300,
    "max_characters": 500000
  },
  "stopwords": {
    "list": [
      "a", "aby", "ale", "ani", "ano", "asi", "avšak", "jež", "
          alespoň", "jíž", "ať", "až", "ba", "bez", "bude", "budem",
           "budeš", "by", "bychom", "byl", "byla", "byli", "bylo", "
          byste", "být", "během", "chtít", "co", "což", "další", "do
          ", "doba", "dokonce", "druhý", "dva", "díky", "hned", "ho"
          , "holt", "i", "jak", "jako", "jaký", "je", "jeden", "
          jednak", "jeho", "jej", "jejich", "její", "jen", "jenom",
          "jenž", "jenže", "ještě", "ji", "jiné", "jist", "již", "jn
          á", "jsem", "jseš", "jsi", "jsme", "jsou", "já", "jít", "k
          ", "kam", "kde", "kdo", "když", "ke", "kolem", "kromě", "
          kterou", "která", "které", "který", "kteří", "kvůli", "kéž
          ", "let", "leč", "mezi", "mi", "mimo", "moci", "muset", "
          my", "má", "máte", "místo", "mít", "můj", "může", "na", "
          nad", "naopak", "naši", "ne", "nebo", "neboť", "nebýt", "
          nechť", "nejen", "nejsou", "není", "než", "nic", "nové", "
          nový", "nám", "nás", "náš", "nýbrž", "o", "od", "ode", "
          okolo", "on", "ona", "onen", "oni", "ono", "ony", "oproti"
          , "ovšem", "pak", "po", "pod", "podle", "podél", "pokud",
          "pomocí", "pouze", "pro", "prostě", "prostřednictvím", "
          proti", "proto", "protože", "proč", "první", "právě", "př
          ed", "přes", "při", "roce", "rok", "roku", "s", "se", "
          sebe", "si", "sice", "skrz", "své", "svých", "svým", "svý
          mi", "svůj", "sám", "ta", "tak", "takový", "také", "taktéž
          ", "takže", "tato", "tedy", "ten", "tenhle", "tento", "
          tentýž", "ti", "to", "tohle", "toho", "tohoto", "tom", "
          tomto", "tomu", "tomuto", "totiž", "tu", "tudíž", "tuto",
```

```
            "tvůj", "ty", "tyto", "této", "tím", "tímto", "týž", "u",
            "už", "v", "vaše", "ve", "vedle", "velice", "veliký", "vy"
            , "vám", "vás", "váš", "více", "však", "všechen", "vše", "
            všichni", "všechny", "vždyť", "z", "za", "zajisté", "zda",
             "zde", "ze", "zpět", "zčásti", "či", "čí", "řekl", "že"
        ]
    },
    "lemma": {
        "bin": "./vendor/majka/majka.exe",
        "dict": "./vendor/majka/majka.w-lt"
    },
    "ngrams": {
        "size": 2,
        "chunking_unit": 256
    },
    "mongo": {
        "host": "$MONGO_HOST:27017",
        "user": "user",
        "password": "password",
        "dbname": "tdpd"
    },
    "api": {
        "base_url": "$API_URL",
        "auth_basic": "$API_AUTH_BASIC"
    },
    "compare": {
        "cuda_blocks": 22,
        "cuda_threads": 512,
        "batch_size": 11000,
        "batch_chunking_limit": 512,
        "clustering_threshold": 100,
        "filter_min_ngrams": 16,
        "filter_equality_threshold": 0.35,
        "comparison_concurrency" : 1
    },
    "ignored": {
        "norm_len": 24,
        "min_len": 5,
        "batch_size": 512,
        "thread_ngrams": 512,
    }
}
```

C.2 Přehled jmenných prostorů v aplikační službě

- `TDPD` ..... kořenový jmenný prostor
  - `TDPD::Util` ..... pomocné třídy (Curl, Encoding, Process, ...)
  - `TDPD::IO` ..... vstupně-výstupní rozhraní (Corpus, InternalDocumentAPI ...)
  - `TDPD::Model` ..... modelové třídy (Document, ...)
  - `TDPD::Formatter` ..... třídy formátování výstupu (JsonFormatter, ...)
  - `TDPD::Service` ..... třídy služeb (DocumentIndexer, ...)
  - `TDPD::Task` ..... úlohy v aplikaci (TaskIndexExternal, TaskDetect, ...)
  - `TDPD::Exception` ..... vlastní výjimky (TaskException, ...)
  - `TDPD::Preprocessor` ..... třídy pro předzpracování (PreprocessorPipeline, ...)

C.3 Ukázkový výňatek z tabulek lemmat

Výpis C.2: Ukázkový výňatek z tabulek lemmat (umělý klíč 74, anotovaný korpus).

```
{
  ...,
  [ 'dosahu', 'dosah' ],
  [ 'dostat', 'dostat' ],
  [ 'dostál', 'dostál' ],
  [ 'dostát', 'dostát' ],
  [ 'dosvit', 'dosvit' ],
  [ 'dosáhl', 'dosáhnout' ],
  [ 'dotace', 'dotace' ],
  [ 'dotaci', 'dotace' ],
  [ 'dotací', 'dotace' ],
  [ 'dotazu', 'dotaz' ],
  [ 'dotazy', 'dotaz' ],
  [ 'dotazů', 'dotaz' ],
  [ 'doteku', 'dotek' ],
  [ 'dotený', 'dotený' ],
  [ 'dotkne', 'dotknout' ],
  [ 'dotlak', 'dotlak' ],
  [ 'dotyku', 'dotyk' ],
  [ 'dotýká', 'dotýkat' ],
  [ 'dotčen', 'dotčen' ],
  [ 'double', 'double' ],
  [ 'dovozu', 'dovoz' ],
  [ 'dovány', 'dovány' ],
  [ 'dování', 'dování' ],
  [ 'dozadu', 'dozadu' ],
  [ 'dozoru', 'dozor' ],
  ...
}
```

C.4 Ukázkový výňatek z tabulek n-gramů

Výpis C.3: Ukázkový výňatek z tabulek n-gramů (umělý klíč 12, anotovaný korpus).

```
{
  ...
  [ 'gabrlíkovi|odborný', 84403 ],
  [ 'gallery|krystalizace', 215784 ],
  [ 'galvanický|článek', 111063 ],
  [ 'galvanizérství|iii', 175604 ],
  [ 'galvegórriz|studies', 79378 ],
  [ 'ganttova|vizualizace', 154413 ],
  [ 'garantovaný|nízký', 61818 ],
  [ 'garantovat|materiál', 52115 ],
  [ 'garáž|parkoviště', 189483 ],
  [ 'gaussově|založený', 147699 ],
  [ 'gazárkovi|účinný', 18779 ],
  [ 'gašparík|realizace', 105503 ],
  [ 'gašparík|stavební', 66851 ],
  [ 'gelový|intelligence', 45854 ],
  [ 'gelový|vyhodnocení', 49995 ],
  [ 'gelový|zadluženost', 61864 ],
  [ 'generace|předešlý', 204244 ],
  [ 'generace|přenášet', 204463 ],
  [ 'generace|zachovávat', 134285 ],
  [ 'generalizability|the', 145548 ],
  [ 'generalized|spectral', 148985 ],
  [ 'generalized|standard', 80345 ],
  [ 'generický|obfuskace', 94580 ],
  [ 'generický|strategie', 142208 ],
  [ 'generování|princip', 70489 ],
  [ 'generování|systém', 181571 ],
  [ 'generování|čerpat', 165668 ],
  [ 'generátor|měření', 17908 ],
  [ 'generátor|připojen', 211248 ],
  [ 'generátor|rozhraní', 155499 ],
  [ 'generátor|síťový', 163109 ],
  [ 'generátor|vyrábět', 211249 ],
  [ 'generátor|vědecký', 154562 ],
  [ 'genetický|klasický', 156191 ],
  [ 'genetický|strategie', 88136 ],
  [ 'genetic|organization', 109716 ],
  [ 'geneva|švajčiarsko', 191646 ],
  [ 'genomic|organization', 127802 ],
  [ 'geodetický|činnost', 159801 ],
  ...
}
```

C.5 Formát uložení dokumentů v korpusu

Výpis C.4: Formát uložení dokumentů v korpusu.

```
  {
    _id: ObjectId('606dfced2ea52804a0f76d06'),
    uri: 'samples/sample_1.pdf',
    type: 'external',
    date_indexed: 1617820908,
    date_updated: 1617820908,
    original_encoding: 0,
    original_type: 2,
    total_pages: 83,
    indexed_pages: 56,
    skipped_pages: 11,
    chunk_placement_positions: [
        [ 12, 3, 255 ],
        ...
    ],
    pages: [
        '', '', '', '', '', '', '', '', '', '', '',
        'Úvod\n\nExpertní systémy jsou počítačové programy...',
        ...
    ],
    lines_per_page: [
        0, 0, 0, 0, 0, 0, 70, 28, 31, 30, 36, 38,...
    ],
    keywords: [],
    ngram_count: 5411,
    ngrams: BinData(0, '
       AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGclAAABAAAAAQAAAGg...')
       ,
    filtered_sections: [
        [ 8181, 8245 ],
        ...
    ]
}
```

D Výstupy indexačního procesu

D.1 Stav aplikace pro anotovaný korpus

Výpis D.1: Stav aplikace pro anotovaný korpus (PlainFormatter).

```
curl -X POST "http://127.0.0.1:3000/task_check_state?format=plain"
---
PROGRESS: Checking daemon state
---
RESULT: Runtime

started_at: 07.04.2021 18:41:41
virtual_memory_usage: 5620M
tasks:
  available_concurrency: 3
  available_detection_concurrency: 1
  managed: 1
  processed: 18
  running: 0
  awaited: 0
---
RESULT: CUDA

device: GeForce GTX 1660 SUPER
available_memory: 6232M
used_memory: 606M
threads_per_block: 1024
---
RESULT: Corpus

disk_usage: 9M
indexed_documents: 12
known_lemmas: 50779
known_ngrams: 220820
ignored_sentences: 138380
---
SUCCESS: Daemon state OK
```

D.2 Stav aplikace pro rozšířený korpus

Výpis D.2: Stav aplikace pro rozšířený korpus (PlainFormatter).

```
curl -X POST "http://127.0.0.1:3000/task_check_state?format=plain"
---
PROGRESS: Checking daemon state
---
RESULT: Runtime

started_at: 19.04.2021 13:28:21
virtual_memory_usage: 7276M
tasks:
  available_concurrency: 3
  available_detection_concurrency: 2
  managed: 1
  processed: 16
  running: 0
  awaited: 0
---
RESULT: CUDA

device: GeForce GTX 1660 SUPER
available_memory: 6232M
used_memory: 657M
threads_per_block: 1024
---
RESULT: Corpus

disk_usage: 733M
indexed_documents: 4597
known_lemmas: 718733
known_ngrams: 9106625
ignored_sentences: 138380
---
SUCCESS: Daemon state OK
```

D.3 Výňatek z indexace reálných prací roku 2017

Výpis D.3: Výňatek z indexace reálných prací roku 2017 (PlainFormatter).

```
curl -X POST "http://127.0.0.1:3000/task_index_internal_all?args
   []=2017&format=json"
---
PROGRESS: Fetching records for year 2017. Updates are disabled.
---
PROGRESS: Starting indexation of 4415 records.
---
PROGRESS: Started indexing internal document with ID: 65624
---
PROGRESS: Started indexing internal document with ID: 71505
---
RESULT: Formed uri.

uri: internal://2017:65624
---
SUCCESS: Finished indexing internal id '65624'. Processed 44 out of
    50 pages.
---
PROGRESS: Started indexing internal document with ID: 71506
---
RESULT: Formed uri.

uri: internal://2017:71505
---
SUCCESS: Finished indexing internal id '71505'. Processed 195 out
   of 206 pages.
---
PROGRESS: Started indexing internal document with ID: 71633
---
RESULT: Formed uri.

uri: internal://2017:71506
---
SUCCESS: Finished indexing internal id '71506'. Processed 216 out
   of 227 pages.
---
PROGRESS: Started indexing internal document with ID: 77147
```

E Výstupy hledání shod

E.1 Další výsledky testů

Dokument	Očekáváno	Nalezeno	Komentář
specimen_1	0	0	-
specimen_2	0	0	-
specimen_3	0	0	-
specimen_4	0	0	-
specimen_5	3	2	Úseky vzniklé přímým plagiátorstvím odhaleny, parafráze nenalezena.
specimen_6	2	2	Správně rozpoznány přímé plagiáty po odstranění citovaných úseků.
specimen_7	2	2	Přímý plagiát i parafráze nalezeny.
specimen_8	0	0	-
Čas detekce: 6s			

Tab. E.1: Výsledky testů na vzoru *sample_2* (anotovaný korpus).

Dokument	Očekáváno	Nalezeno	Komentář
specimen_1	0	0	-
specimen_2	0	0	-
specimen_3	0	0	-
specimen_4	0	0	-
specimen_5	0	0	-
specimen_6	0	0	-
specimen_7	0	0	-
specimen_8	3	3	Ignorované věty úspěšně odstraněny, zbývající text označen jako shoda.
Čas detekce: 4s			

Tab. E.2: Výsledky testů na vzoru *sample_3* (anotovaný korpus).

E.2 Hledání shod na sample_3 (anotovaný korpus)

Výpis E.1: Hledání shod na sample_3 (anotovaný korpus, none, PlainFormatter).

```
PROGRESS: Comparing "samples/sample_3.pdf" against batches of
   around 11000 document chunks in corpus of 11 documents
---
PROGRESS: Preparing subject document.
---
PROGRESS: Processing batch 1 (100%).
---
PROGRESS: Preparing results.
---
RESULT: Matched.

subject_uri: samples/sample_3.pdf
compared_uri: samples/specimen_8.pdf
matches:

  approx_eql: 1.000
  subject_pages: 1-1
  compared_pages: 1-1
  subject_lines: 2-4
  compared_lines: 1-3

  approx_eql: 0.973
  subject_pages: 1-1
  compared_pages: 1-1
  subject_lines: 19-22
  compared_lines: 15-20

  approx_eql: 0.975
  subject_pages: 1-1
  compared_pages: 1-1
  subject_lines: 33-34
  compared_lines: 31-33
---
RESULT_DETAIL: Per page matched lines histogram.

matched: 9
total: 46
---
SUCCESS: Comparison matched 1 documents with 3 matches on "samples/
   sample_3.pdf", matching took 4 seconds.
```

Výpis E.2: Hledání shod na sample_3 (rozšířený korpus, none, JsonFormatter).

```
curl -X POST "http://127.0.0.1:3000/task_detect?args[]=samples/
    sample_3.pdf&args=none&async=0&format=json"

{"type":"progress","message":"Comparing \"samples\/sample_3.pdf\"
    against batches of around 11000 document chunks in corpus of
    4597 documents"}
{"type":"progress","message":"Preparing subject document."}
{"type":"progress","message":"Processing batch 1 (15%)."}
{"type":"progress","message":"Processing batch 2 (28%)."}
{"type":"progress","message":"Processing batch 3 (46%)."}
{"type":"progress","message":"Processing batch 4 (61%)."}
{"type":"progress","message":"Processing batch 5 (76%)."}
{"type":"progress","message":"Processing batch 6 (89%)."}
{"type":"progress","message":"Processing batch 7 (100%)."}
{"type":"progress","message":"Preparing results."}
{"type":"result","message":"Matched.","data":{"subject_uri":"
    samples\/sample_3.pdf","compared_uri":"samples\/specimen_8.pdf",
    "matches":[{"approx_eql":"1.000","subject_pages":"1-1","
    compared_pages":"1-1","subject_lines":"2-4","compared_lines":"
    1-3"},{"approx_eql":"0.973","subject_pages":"1-1","
    compared_pages":"1-1","subject_lines":"19-22","compared_lines":"
    15-20"},{"approx_eql":"0.975","subject_pages":"1-1","
    compared_pages":"1-1","subject_lines":"33-34","compared_lines":"
    31-33"}]}}
{"type":"result_detail","message":"Per page matched lines histogram
    .","data":[{"matched":"9","total":"46"}]}
{"type":"success","message":"Comparison matched 1 documents with 3
    matches on \"samples\/sample_3.pdf\", matching took 6 seconds."}
```

E.3 Část výstupu pro sample_2 (parafráze)

Výpis E.3: Část výstupu pro sample_2 s nalezenou parafrází zobrazenou originálním textem (anotovaný korpus, PlainFormatter). Řádkování upraveno.

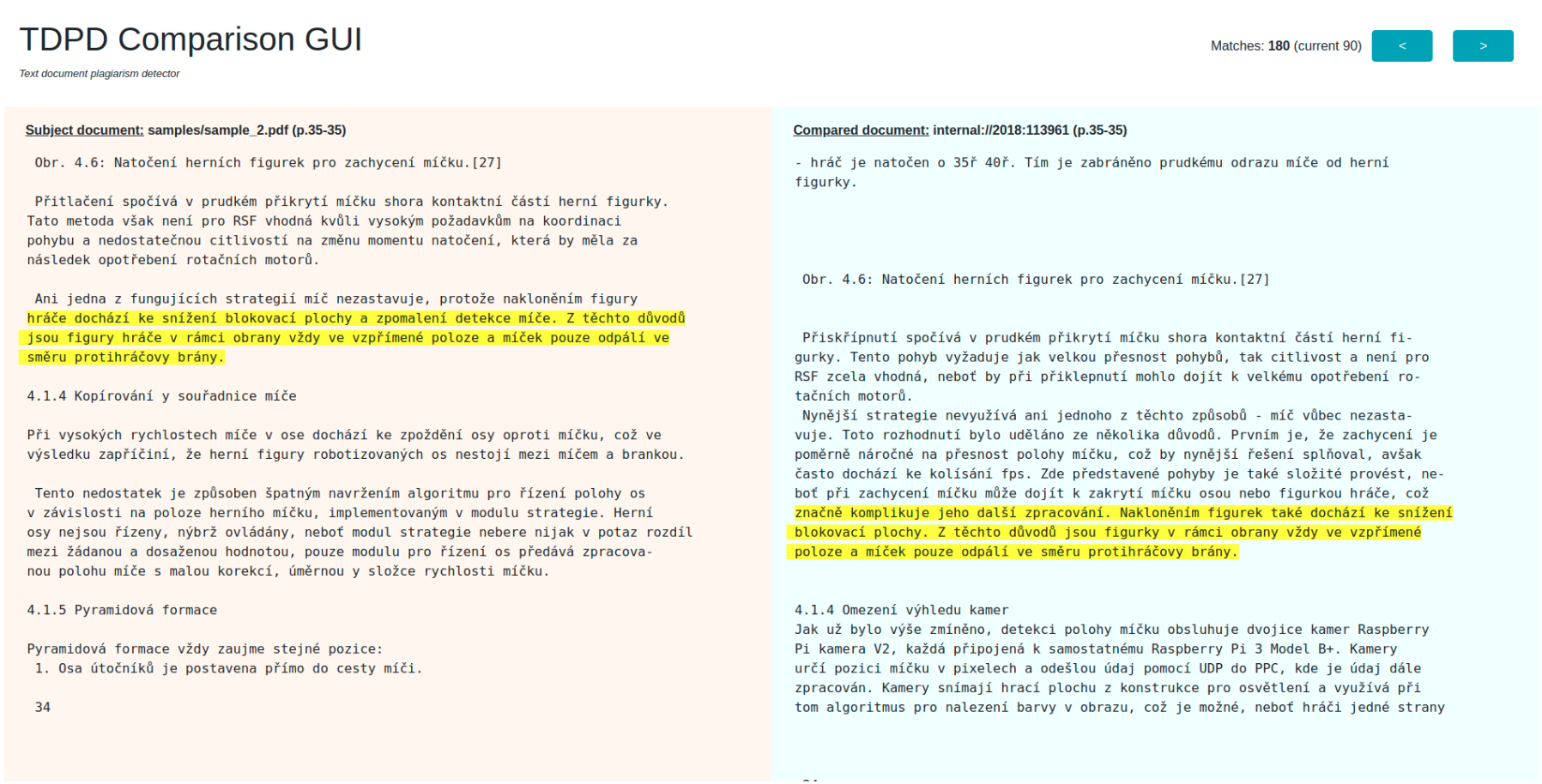
```
subject_uri: samples/sample_2.pdf
compared_uri: samples/specimen_7.pdf
...
approx_eql: 0.444
subject_pages: 39-39
compared_pages: 1-1
subject_lines: 29-37
compared_lines: 1-7
subject_content:     Výše uvedené úpravy byly odlazeny v simulaci a
    jsou využívány u střely s přihrávkou. V ostatních případech je
   stále využíván již originální algoritmus pro řízení, neboť přímé
    předávání hodnot vyžaduje úpravu parametrů plánovače trasy vš
   ech os, kterou nebylo možné provést bez přístupu k sestavenému
   modelu. Dalším důvodem pro užívání původního algoritmu je, že v
   některých případech byla při použití přímého předávání hodnot
   odchylka mezi žádanou a aktuální pozicí osy výrazně vyšší, než u
    původního algoritmu. Odchylka však může být způsobena i chybou
   v simulaci osy, neboť chyba se projevuje vždy ve stejné situaci
   a proto, že původní regulace s identickými vstupními daty
   funguje správně.

compared_content: Tyto úpravy byly odladěny simulací a jsou použity
    u střely s přihrávkou. Jinak se využívá originální řídicí
   algoritmus. Přímé předávání hodnotby vyžadovalo modifikaci
   parametrů plánovače trasy pro všechny osy, což nelze provést bez
    přístupu ke kompletnímu modelu. Původní algoritmus se používá
   také proto, že někdy byla při použití přímého předávání hodnot
   vysoká odchylka od žádané pozice osy. Důvodem pro vznik odchylky
    může být i chyba v simulaci osy, protože chyba nastává vždy ve
   stejné situaci proto, že původní regulace na těchto vstupních
   datech funguje bez chyby.
```

Výpis E.4: Část výstupu pro sample_2 s rekonstrukcí n-gramových slov nalezené parafráze (anotovaný korpus, PlainFormatter).

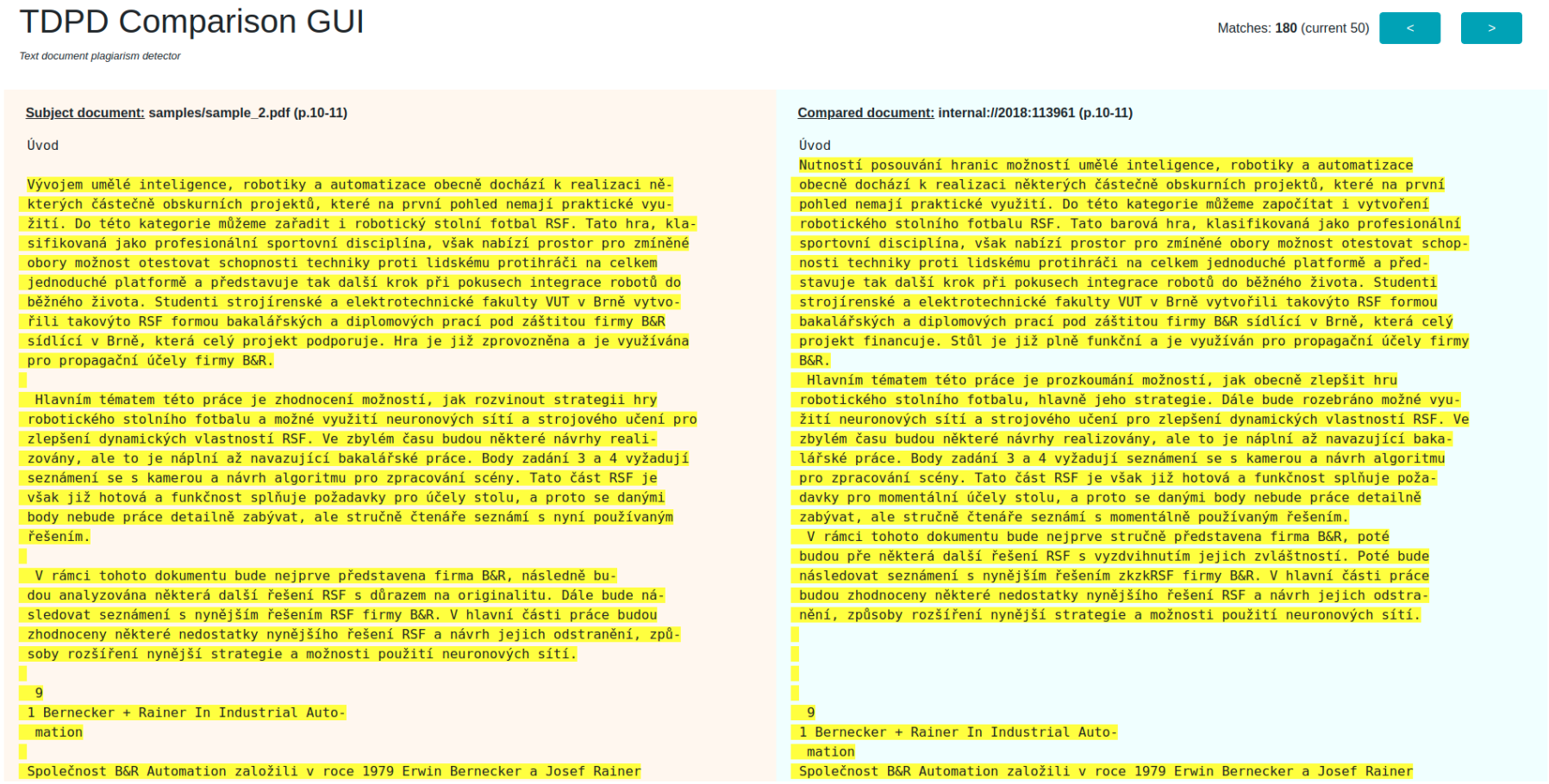
```
subject_uri: samples/sample_2.pdf
compared_uri: samples/specimen_7.pdf
...
approx_eql: 0.444
subject_pages: 39-39
compared_pages: 1-1
subject_lines: 29-37
compared_lines: 1-7
subject_content: (být úprava)(být odlazeny)(odlazeny simulace)(
   simulace využívat)(střel využívat)(přihrávka střel)(ostatní př
   ihrávka)|(ostatní případ)(případ stále)(stále využívat)(originá
   lní využívat)(algoritmus originální)(algoritmus řízení)(přímý ří
   zení)|(předávání přímý)(hodnota předávání)(hodnota vyžadovat)(vy
   žadovat úprava)(parametr úprava)(parametr plánovač)(plánovač
   tras)(tras všechen)(os všechen)(být os)|(být možný)(možný prové
   st)(provést přístup)(přístup sestavený)(model sestavený)(další
   model)(další důvod)(důvod užívání)|(původní užívání)(algoritmus
   původní)(algoritmus některý)(některý případ)(použití případ)(pou
   žití přímý)(předávání přímý)|(hodnota předávání)(hodnota
   odchylka)(odchylka žádaný)(aktuální žádaný)(aktuální pozice)(osa
    pozice)(osa výrazně)(vysoký výrazně)(původní vysoký)|(
   algoritmus původní)(algoritmus odchylka)(odchylka způsobit)(
   chyba způsobit)(chyba simulace)(osa simulace)|(chyba osa)(chyba
   projevovat)(projevovat vždy)(stejné vždy)(situace stejné)(původn
   í situace)(původní regulace)(identický regulace)|(identický
   vstupní)(datum vstupní)(datum fungovat)
compared_content: (být úprava)(být odladit)(odladit simulace)(použ
   it simulace)(použit střel)(přihrávka střel)(jinak přihrávka)(
   jinak využívat)(originální využívat)|(originální řídicí)(
   algoritmus řídicí)(algoritmus přímý)(předávání přímý)(hodnotby p
   ředávání)(hodnotby vyžadovat)(modifikace vyžadovat)(modifikace
   parametr)(parametr plánovač)|(plánovač tras)(osa tras)(lze osa)(
   lze provést)(provést přístup)(kompletní přístup)(kompletní model
   )(model původní)|(algoritmus původní)(algoritmus používat)(někdy
    používat)(někdy použití)(použití přímý)(předávání přímý)(
   hodnota předávání)(hodnota vysoký)|(odchylka vysoký)(odchylka žá
   dané)(pozice žádané)(osa pozice)(důvod osa)(důvod vznik)(
   odchylka vznik)(chyba odchylka)(chyba simulace)|(osa simulace)(
   chyba osa)(chyba nastávat)(nastávat vždy)(stejné vždy)(situace
   stejné)(původní situace)(původní regulace)(regulace tento)(tento
    vstupní)|(datum vstupní)(datum fungovat)
```

E.4 Vybrané nálezy (webové rozhraní klienta)

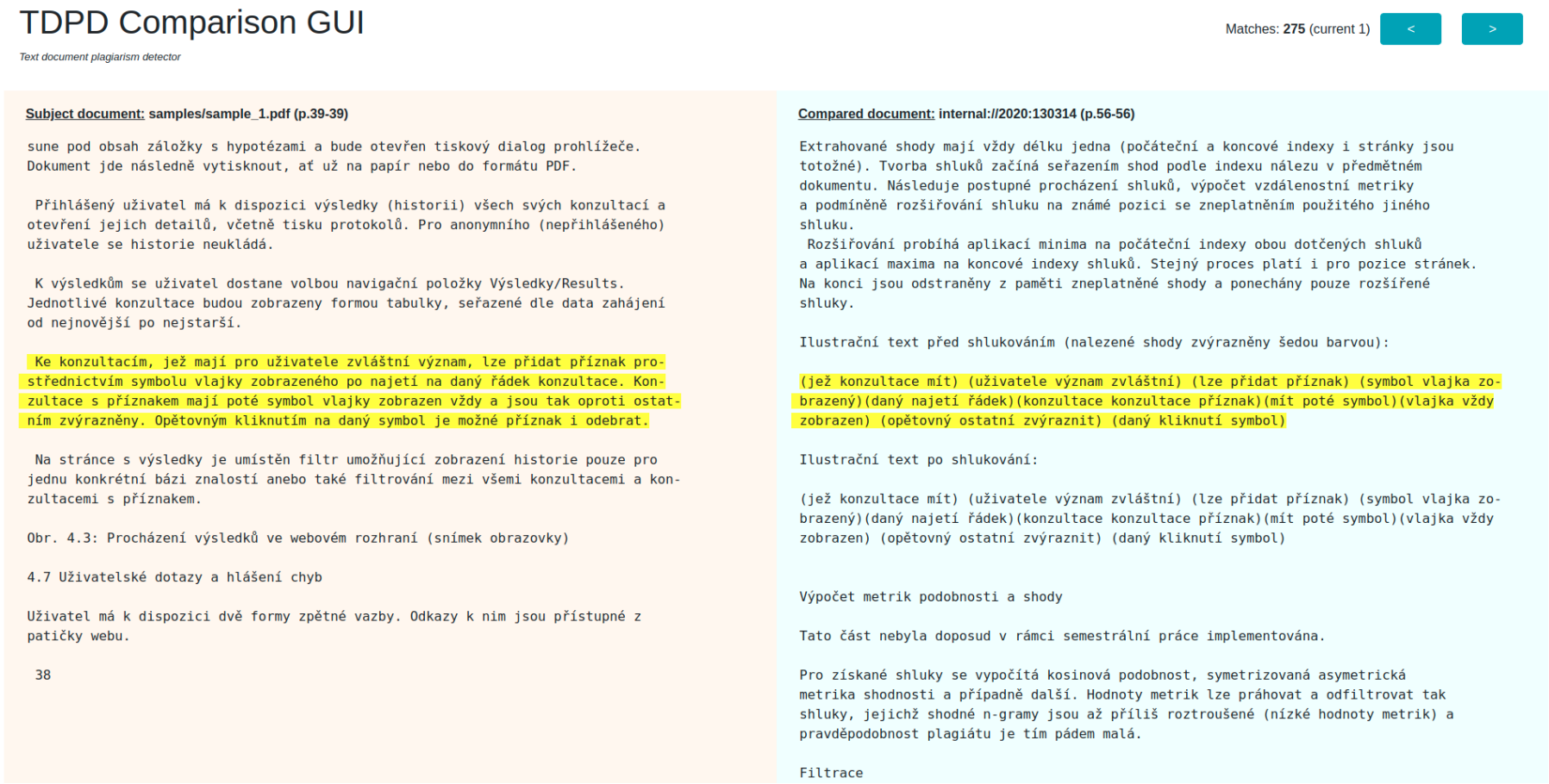
Na následujících stranách jsou umístěny snímky webového rozhraní aplikačního klienta pořízené při prohlížení výsledků hledání shod na rozšířeném korpusu.



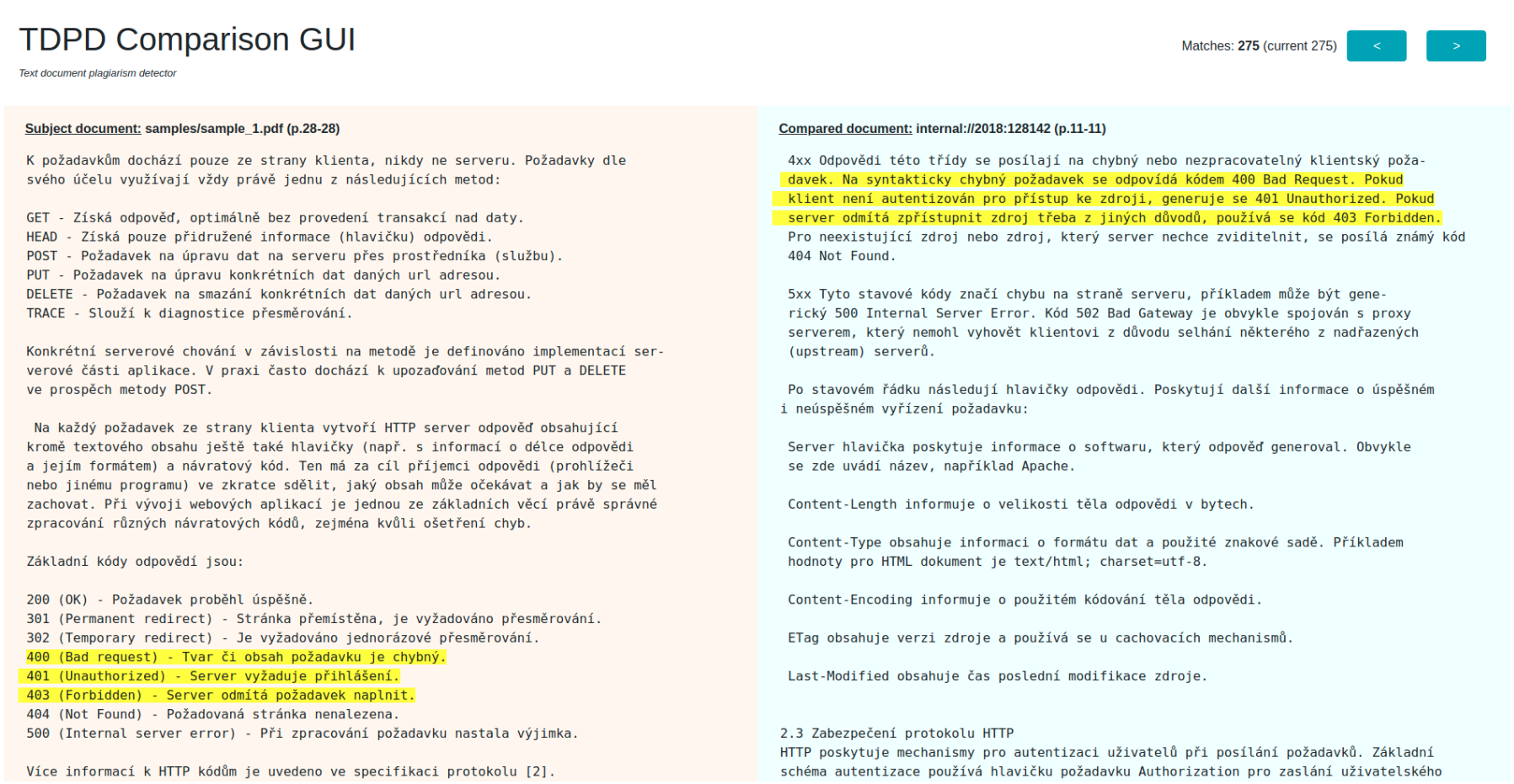
Obr. E.1: Vybraná shoda v rozhraní klienta (sample_2, krátký úsek).



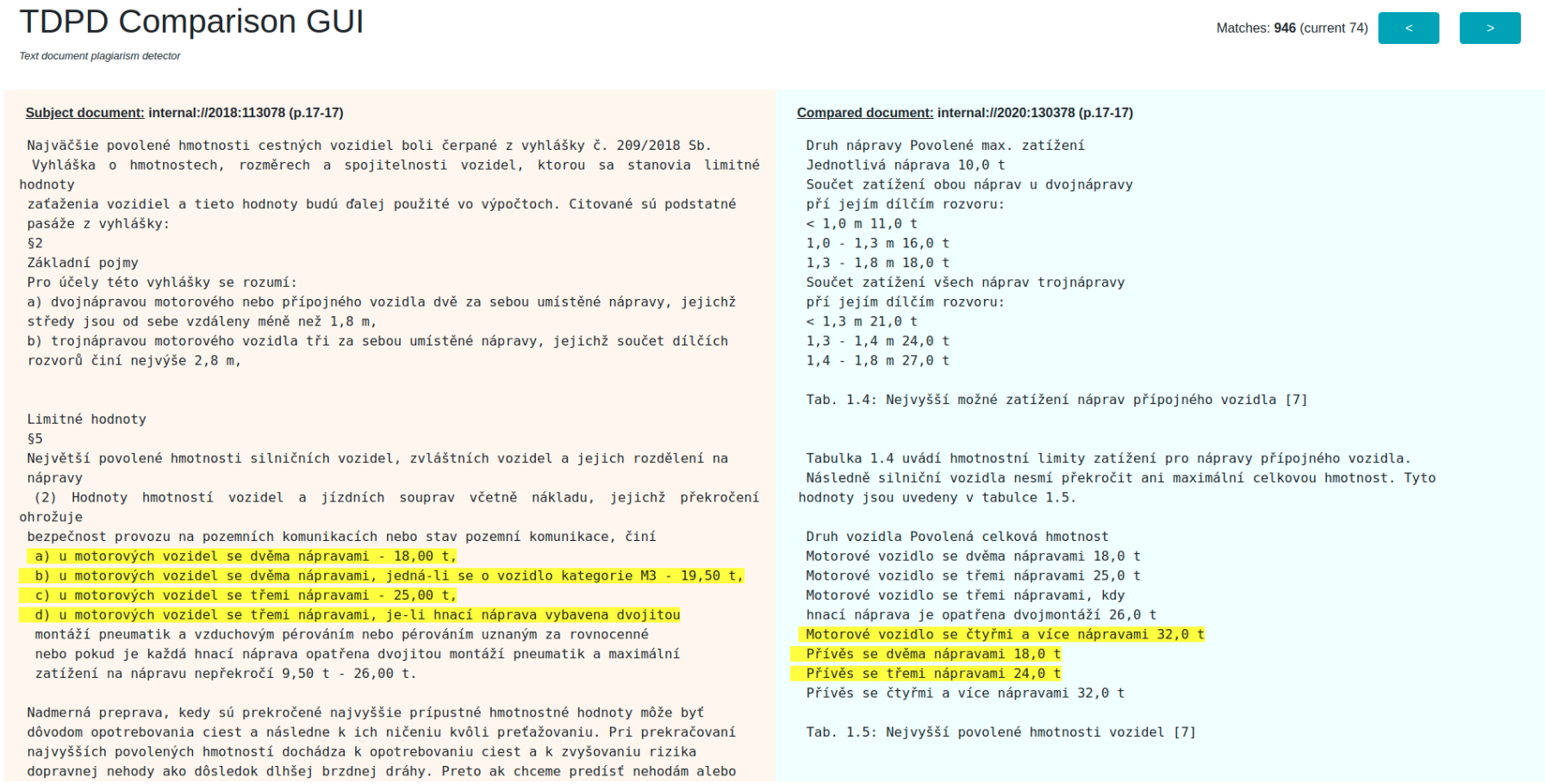
Obr. E.2: Vybraná shoda v rozhraní klienta (sample_2, parafrázovaný úsek).



Obr. E.3: Vybraná shoda v rozhraní klienta (viditelné předzpracování).



Obr. E.4: Vybraná shoda v rozhraní klienta (rozšířený korpus, 1).



Obr. E.5: Vybraná shoda v rozhraní klienta (rozšířený korpus, 2).