Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Bogáromi Zoltán, Juszt Ádám

PLÁGIUMKERESÉS A BME VIK DIPLOMATERV ÁLLOMÁNYÁBAN

KONZULENS

Dr. Kővári Bence

BUDAPEST, 2016

Tartalomjegyzék

Összefoglaló

2010 óta a Budapesti Műszaki és Gazdaságtudományi Egyetem (BME) Villamosmérnöki és Informatikai Karán (VIK) a beadott szakdolgozatok és diplomatervek elektronikus formátumban nyilvánosan hozzáférhetők. Ez a szabad információhozzáférés mellett megnyitotta a kaput a dolgozatok tartalmának gépi feldolgozása előtt is.

Kutatásunk során a BME VIK dolgozatállományát felhasználva kidolgoztunk egy megoldást, mely képes hatékonyan elemezni, hogy milyen hasonlóságok fedezhetők fel a különböző beadott anyagokban, illetve összevetni azokat ismert külső forrásokkal. Ez utóbbi mechanizmus működését a magyar nyelvű Wikipedia adatbázison demonstráljuk.

Dolgozatunkban bemutatjuk a kialakított keretrendszerünk és algoritmusunk működését, bemutatjuk azokat az algoritmikus trükköket, melyekkel a nagy méretű szövegállomány mellett is belátható időben elvégezhető a dolgozatok összevetése. Algoritmusunk segítségével már most is számos, megfelelő forráshivatkozás nélküli egyezést azonosítottunk a dolgozatok között. A bevont forrásmunkák körének a bővítésével egy általános, megbízható plágiumdetektor állítható elő az egyetemi szféra számára.

Abstract

The Faculty of Electrical Engineering and Informatics (VIK) at the Budapest University of technology and Economics (BME) made all thesis works public since 2010. Besides the free information access this also opened the gate for the automated processing of documents.

Throughout our research we have created a solution that is able to efficiently process the BME VIK dataset and analyze the similarities in the different thesis works. The solution is also able to compare these thesis works to different well known external sources, which we demonstrate through comparing our data to the Wikipedia dataset.

In our work we show the functional principles of our framework and algorithms. To allow the timely processing of large text databases we have used several special implementation techniques which we also cover in our paper. Using our algorithm, we have already identified several suspicious overlaps between existing thesis works. By extending the circle of reference documents, we will be able to get a reliable plagiarism detector for the purposes of higher education.

1 Bevezetés

Napjainkban az interneten megtalálható hatalmas adathalmaz számottevően megkönnyíti a kutatómunkákat. Számos tudományos cikk, könyv vagy folyóirat található meg elektronikus formában, amelyeket célzott kereséssel könnyen megtalál az ember. Ezeket a dokumentumokat felhasználhatjuk saját munkánkban, ha az átvett tartalmat megfelelően megjelöljük, és hivatkozzuk az eredeti munkát. Előfordul azonban, hogy a hivatkozás – véletlenül, akár szándékosan – hiányzik. Ekkor plágiumról beszélünk, a tartalom jogtalan átvételéről, azaz sajátként tüntetjük fel más munkáját, ami komoly jogi következményeket vonhat maga után.

Az egyetemi hallgatók szakdolgozat- és diplomatervkészítéshez is előszeretettel aknázzák ki az internet lehetőségeit, és használnak fel interneten megtalált forrásokat dolgozatuk készítése során. A felhasznált dokumentumokra azonban nem mindig hivatkoznak megfelelően.

A plágiumkeresés igen erőforrásigényes feladat. Csak a Budapesti Műszaki és Gazdaságtudományi Egyetem (BME) hallgatói évente többszáz szakdolgozatot vagy diplomatervet írnak meg. Ezek átvizsgálása jelentős időbefektetést igényel, hiszen még a dolgozatokban idézett források megfelelő hivatkozását is hosszadalmas munka ellenőrizni. Nem beszélve a nem hivatkozott, de jogtalanul átvett tartalmakról, amelyek esetén azok forrását is fel kell kutatni.

Ezt a feladatot hivatott könnyíteni egy automatizált megoldás, amely az elektronikus formában elérhető dokumentumokat képes másik dokumentumokkal összehasonlítani, és ezáltal a lehető legjobban leszűkíteni az ember által átnézett dokumentumok körét.

Kutatásunk célja olyan algoritmusok vizsgálata volt, melyek alkalmasak lehetnek a BME életében keletkező dokumentumok plágiumellenőrzésére. Algoritmusaink működésének demonstrálására megvalósítottunk egy rendszert, mely a BME Villamosmérnöki és Informatikai Kar (VIK) Diplomaterv Portál [1] adatainak a plágiumellenőrzését tudja elvégezni különböző forrásokkal való összehasonlítások által. A most bemutatott implementáció forrásként a teljes dolgozatállományt, illetve a magyar Wikipedia adatbázisát használja.

Tudomásunk szerint hasonló projekt az egyetemen eddig nem volt, ami külön kihívást jelent, hiszen elsőként számolhatunk be az egyetemi anyagok feldolgozásának eredményeiről.

Nem kell informatikusnak lenni ahhoz, hogy észre vegyük: ez az adathalmaz jelentős méretű, feldolgozása nem megy egyik percről a másikra. Emiatt elengedhetetlen különböző kreatív megoldásokat alkalmazni, hogy az algoritmus a vizsgálattal belátható időn belül végezzen. Az általunk felhasznált technikákat a futási idő csökkentésére a későbbi fejezetekben részletesen is ismertetjük.

2 Plágiumkereső technikák áttekintése

A plágiumkeresés célja egyértelmű, azonban a megvalósítással rengeteg különböző irányba el lehet indulni. Végig kell gondolni többek között, hogy miként karakterizáljuk a dokumentumokat, amiket össze kívánunk hasonlítani, valamint az is kiemelt fontosságú tervezői döntésnek számít, hogy globális vagy lokális hasonlóságkeresési algoritmusra alapozzuk az alkalmazást.

2.1 Dokumentumok karakterizálása

Jelen írás keretei közt kivétel nélkül természetes nyelveken íródott dokumentumokkal foglalkozunk, nem célunk tehát például programkódok közötti hasonlóságokat megkeresni, ezek alapjaiban különböző technikákat igényelnek, melyek nem részei a választott témánknak.

Ezt figyelembe véve kijelenthető, hogy a dokumentumokban érdemleges adatot a betűk, valamint az ezekből alkotott szavak alkotnak. Nem hordoznak plágiumkeresés szempontjából releváns információkat a számok, írásjelek, szóközök és sortörések. Ezeket minden karakterizálási módszer esetén szűrőkkel eltávolítjuk a dokumentumokból, hiszen többszöri (felesleges) feldolgozásuk indokolatlanul lassítaná az alkalmazás futását.

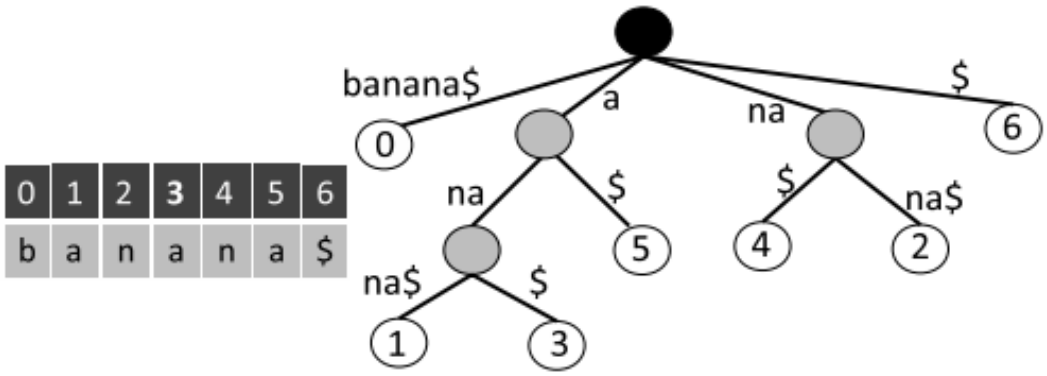
A legegyszerűbb, magától értetődő módszer a betű alapú karakterizáció. Ebben az esetben a dokumentum egymást követő betűk halmaza, így az összehasonlítás is karakterenként történik. [2]

Egy absztrakciós szinttel feljebb lépve vizsgálhatjuk úgy is a dokumentumot, mint szavak halmazát. Így kevesebb összehasonlításra lesz szükségünk, hiszen egy szövegben kevesebb szó található, mint betű.

Az általunk használt, a következő fejezetben részletesebben is kifejtett technika a kifejezéseket veszi a karakterizáció alapjául, ahol egy kifejezés legyen n darab egymást követő szó összessége. Az n változó különböző értékeinek előnyeiről és hátrányairól tudományos cikkek születtek, mi a plágiumkeresés területén legelterjedtebb értéket, a 3-at választottuk. Egy ilyen, három szóból álló kifejezés a 3-gram, vagy trigram.

Ennél nagyobb egységeket is választhatunk, például mondatokat, sorokat, vagy bekezdéseket. Minél nagyobb egységet veszünk, annál kisebb lesz az álpozitív[1] egyezések száma, viszont ily módon az algoritmus könnyedén elsiklik a kisebb egyezések felett. Egy speciális felhasználási esetben tekinthetjük az egész dokumentumot a feldolgozás legkisebb egységének, ebben az esetben hash[2] értéket képezhetünk belőle, és dokumentum duplikációt kereshetünk, mely a jóval rövidebb hash értékek között jóval kevésbé idő- és erőforrásigényes feladat.

Ezeken felül felmerülhet az ötlet, hogy az adatelemzési szektorban széles körben elterjedt Suffix Tree módszerrel reprezentáljuk az adatokat [3].



1. ábra - Az S = banana$ stringből képzett Suffix Tree [4]

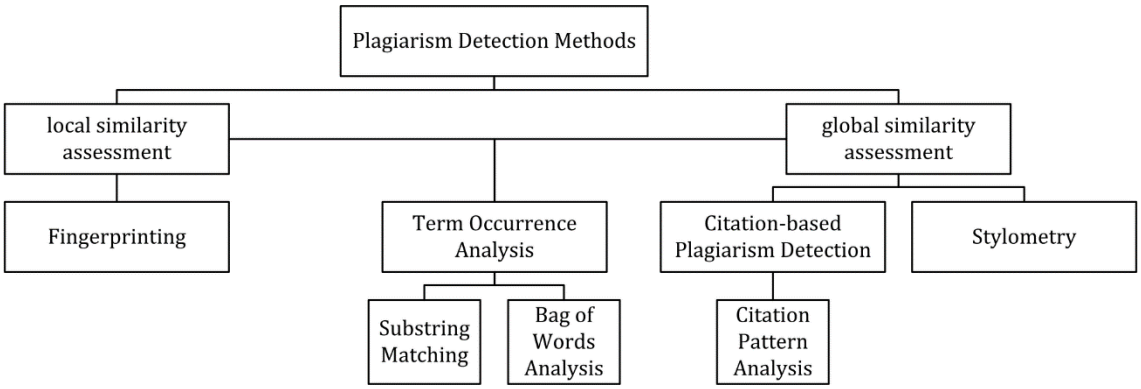
Az 1. ábra szemléletesen bemutatja a Suffix Tree struktúrájának jellemzőit. Egy ilyen fában megtalálható a tárolt szó összes szuffixuma, azaz az összes olyan részkaraktersora, melynek vége megegyezik a szó végével. A fenti példával élve tehát a nana$ karaktersorozat a banana$ szuffixuma, de például a „bana” karaktersorozat nem, hiszen a vége nem egyezik a banana$ string végével [5].

[1] Angol irodalomban false positive. Jelen szövegkörnyezetben akkor lesz két dokumentum között álpozitív egyezés, ha az algoritmusunk szerint plágiumgyanú merül föl köztük, valójában azonban nincs erről szó.

[2] Hash függvény segítségével bármilyen hosszúságú adatot adott hosszúságra képezhetünk le, az így kapot adat neve hash érték.

Egy ilyen adatszerkezet bevezetésével megoldható lenne a szótövesítés problémája, valamint a szóegyezések keresése is felgyorsulna a fastruktúrának köszönhetően. Egyelőre azért nem vezettük be ezt a módszert, mivel a fák felépítése jelentősen megnövelné az algoritmusunk futási idejét

2.2 Plágiumkeresési technikák



2. ábra - Plágiumkeresési módszerek áttekintése [16]

A fenti ábrán is látszik, hogy a plágiumkeresésnek több, alapjaiban eltérő módszere is van. A következőkben ezen kategóriák közül ismertetünk egy lokális és egy globális hasonlóságra alapuló technikát, valamint ismertetjük a klaszterezés fogalmát, mely bármilyen típusú adatelemzés egyik alapvető technikája.

2.2.1 Fingerprint matching

A magyarul „ujjlenyomat módszer” néven emlegetett technika a nevéből kitalálható módon ujjlenyomatokra alapul. Ezen metódus keretei közt minden dokumentumhoz szeretnénk valamilyen egyedi azonosítót, kivonatot képezni, melyet a szöveg ujjlenyomatának nevezünk [6]. A 2.1-es fejezetben láthattuk, hogy a dokumentumok karakterizálásának számos módja ismert, ezek alapján az ujjlenyomat módszer is különböző kategóriákra oszlik. Ezen kategóriák alapja szintén az, hogy mit tekintünk a dokumentum legkisebb egységének. Mivel azonban az általunk használt, kifejezés alapú ujjlenyomat módszer a legelterjedtebb, ezért csak ezt a technikát fogjuk részletezni.

A kifejezés alapú fingerprint matching technika trigramokat használ a dokumentum reprezentálására [7]. Tehát például a „Peti szereti az almás rétest” mondat a következő trigramokká lesz átalakítva: {„Peti szereti az”, „szereti az almás”, „az almás

rétest”}. Miután minden dokumentumot átalakítottunk ilyen trigramok összességéve, következhet ezen halmazok párosával történő összehasonlítása.

Naiv implementációval (ha az összes dokumentum összes trigramját páronként összevetjük a többivel, bármilyen optimalizálást és párhuzamosítást nélkülözve) azonban a diplomatervek adathalmazán ez az összehasonlításssorozat beláthatatlanul hosszú ideig tartana, a különböző gyorsítási módokkal a 3.3-as fejezetben foglalkozunk részletesebben.

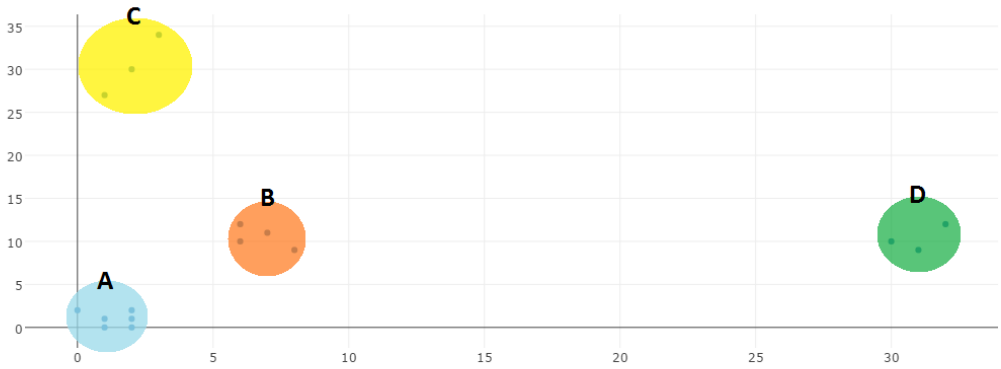
2.2.2 Stilometria

A stilometria célja egy adott dokumentumról – legyen az szakdolgozat, regény, szonett, stb. – eldönteni, hogy ki a szerzője [8]. Az egész technika abból a feltételezésből indul ki, hogy minden szerzőnek van egy jól behatárolható és azonosítható írási stílusa, amelynek jellegzetességeit ki lehet mutatni, amennyiben elegendő mennyiségű dokumentum áll rendelkezésre.

Plágiumkeresésre való használata során valójában a szerzői stílusok közti hasonlóságot keresnénk egy stilometriára alapuló alkalmazással. A valóságban ez a technika nem igazán terjedt el, mert az eredmények alapján úgy tűnik, hogy a szerzői stílus nem tekinthető állandónak, az idő előrehaladtával változik. Emiatt legfeljebb kiegészítő plágiumkereső technikaként alkalmazzák, azaz ha egy másik metódus gyanúsnak talált egy dokumentumpárt, akkor második körben alávetik egy stilometriai tesztnek is.

2.2.3 Klaszterezés

Klaszterezésnek nevezzük azt a technikát, amikor a hasonló objektumokat egy csoportba (klaszterbe) rendezzük. A klaszteren belül lévő objektumok hasonlítanak egymáshoz és nem hasonlítanak a klaszteren kívül lévő objektumokhoz. [9] [10]



3. ábra - Grafikus példa a klaszterekről

Ez az elv felhasználható a plágiumkeresés segítésére is. Megtehetjük ugyanis, hogy első lépésként lefuttatunk a teljes dokumentumhalmazra egy klaszterezési algoritmust, melynek eredményeképp megkapjuk az adathalmazra jellemző tematikus csoportokat. Ezt követően megtehetjük, hogy csak a klasztereken belül vizsgáljuk az egyezéseket, például az ujjlenyomat módszer segítségével. Ebben az esetben azonban felmerül az a probléma, hogy akár rengeteg lehetséges egyezést elhanyagolunk, hiszen plágiumgyanú nem feltétlenül csak tematikájukban hasonló dokumentumok között merülhet fel.

3 Plágiumkereső megoldás ismertetése

Ebben a fejezetben bemutatjuk az általunk vizsgált adathalmazt, amely BME VIK diplomaterv portálján elérhető szakdolgozataiból és diplomamunkáiból, valamint a magyar nyelvű Wikipedia adatbázisból áll. Emellett ismertetjük az adatokat feldolgozó algoritmusokat is.

3.1 Diplomaterv portál dolgozatai

A BME VIK Diplomaterv Portálon [1] több ezer szakdolgozat és diplomaterv található, időbeli és tematikus bontásban. A dolgozatok egyesével történő letöltése azonban rengeteg időbe és munkába kerülne, ezért ezt is automatizáltan oldottuk meg, melynek módszerét a következőkben ismertetjük.

A Diplomaterv Portál rendelkezik egy alkalmazások számára használható programozói felülettel. Ezzel lekérdezhetők a dolgozatokról elérhető bizonyos adatok. A hitelesítés nehézségei miatt azonban egy hibrid megoldást alkalmaztunk, részben ezt használva, részben pedig böngészőautomatizálás segítségével a felhasználói felületet használva. A portálról 5 840 darab dolgozatot sikerült letölteni.

Letöltés után a PDF formátumú dolgozatok szöveggé átalakítására volt szükség. A művelet nem bizonyult könnyűnek, még meglévő alkalmazás használata esetén sem. Számtalan PDF-et átalakító szoftver érhető el, azonban nagy problémájuk, hogy a magyar nyelvet nem támogatják. Magyar nyelven tudó szoftvert, ami több ezer dokumentumot is képes átalakítani viszonylag egyszerűen, nagyon nehezen találtunk. A végül alkalmazott megoldás [11] a dokumentumok számához képest elviselhető idő alatt képes volt a feladatot elvégezni. A feldolgozás során 56 dolgozat átalakítása (vélhetően a PDF formátum sajátosságai miatt) nem sikerült, így csupán 5 784 dokumentumot tudtunk megvizsgálni.

Az átalakított fájlok vizsgálata során kiderült, hogy számos dokumentum szöveg helyett különböző szimbólumokat tartalmazott. Ezeket a hibásan átalakított fájlokat a 3.3.6-os fejezetben leírt módon kiszűrtük, ezek konkrét számát az algoritmusunk eredményei között ismertetjük a 4.3-es fejezetben.

3.2 Wikipedia adatbázis

A Wikipedia bizonyos időközönként mentést készít a teljes cikkállományáról, amely publikusan elérhető letöltésre [12]. Mi a magyar nyelvű Wikipedia mentést töltöttük le (annak is pontosan a 2016. szeptember 1-i verzióját), amely azonban abban a formában nem volt alkalmas feldolgozásra. A teljes adathalmaz egyetlen fájlba volt tömörítve, és természetesen tartalmazta a Wikipedia oldalán szükséges jelölőnyelvi elemeket.

Első dolgunk volt, hogy ezeket a számunkra szükségtelen elemeket eltávolítsuk a cikkekből. Szerencsére a problémával nem mi találkoztunk először, ezért be tudtunk vetni egy szkriptet [13], amely a hatalmas mentésből több kisebb állományt készít, és eltávolítja a vizsgálatunk szempontjából szükségtelen jelölőnyelvi elemeket. Az általunk használt algoritmus, amelyet a 3.3-as fejezet mutat be, ezeket a fájlokat használja bemenetként.

A Wikipedia cikkei nagyban eltérnek a dolgozatoktól. A hosszú dolgozatokkal ellentétben jellemzően rövidek, és nagyságrendileg 500 000 darab van belőlük. A cikkek témája a BME VIK dolgozatait tekintve nagy többségében irreleváns, hiszen azok rendkívül sok területtel kapcsolatosak, szemben a dolgozatok specifikus szakterületével. Az irreleváns cikkek szűrésének megoldása meglátásunk szerint több időbe került volna, mint kivárni, hogy a teljes adatbázist feldolgozza az algoritmus; szűrés nélkül is elviselhető idő alatt sikerült végrehajtani a vizsgálatot az adathalmaz méretéhez képest.

3.3 Adathalmaz feldolgozása

Az adathalmaz feldolgozásakor a 2.2-es fejezetben ismertetett plágiumkeresési technikák közül a lokális mintákat vizsgáltuk, trigramok segítségével. Ennek erőforrásigénye elviselhető mértékű a globális vizsgálati megoldásokhoz képest.

A fejezet elején vázlatosan áttekintjük a feldolgozást. A 3.1-es és 3.2-es fejezetben ismertetett adathalmaz meglehetősen nagy méretű, feldolgozáshoz használt algoritmus optimalizálása ezért kritikus fontosságú feladat volt ahhoz, hogy a teljes adathalmazt meg tudjuk vizsgálni. Emiatt először nem az aktuális munkához használt algoritmust, hanem annak fejlődési lépéseit mutatjuk be, megindokolva az egyes módosítások szükségességét, és világossá téve az algoritmus végleges formájának okait. Megjegyzendő, hogy a fejlesztés első fázisában csupán a szakdolgozatok és

diplomatervek összehasonlítására koncentráltunk, a Wikipedia csupán később került bele a vizsgálatba.

Ezt követően pedig a végleges verzió kerül részletesebb ismertetésre, majd a kimenet további feldolgozásának miként je.

3.3.1 Vázlatos ismertetés

4. ábra – Algoritmus lépései

Az adathalmazt feldolgozó algoritmus alapvetően három fázisból épül fel. Az első fázis az adathalmazt betölti, és különböző szűrőfeltételek alapján szűri, átalakítja azt, előkészíti a vizsgált dokumentumokat az összehasonlításhoz.

A második fázis végzi az előkészített dokumentumok összepárosítását, valamint a párok összehasonlítását.

A harmadik fázisban pedig az összehasonlító program által előállított kimenet kerül rögzítésre, erre különböző lehetőségeket építettünk be.

3.3.2 Első verzió

A program első verziója egyfajta proof-of-concept funkciót töltött be. Képes volt összehasonlítani dokumentumokat a trigram technikával, de nagy adatmennyiség feldolgozására nem volt alkalmas, lassú teljesítménye miatt. Az algoritmus egyes verzióinak konkrét teljesítményét a 4.2-es fejezet ismerteti.

Az algoritmus a fájlok beolvasása után szűréseket végzett, eltávolította az összes nem alfanumerikus karaktert[3] (egyetlen kivétel a kötőjel: „-”), és az összes whitespace[4] karaktert, illetve azok láncait egyetlen szóközzel helyettesítette. Az előkészített szövegben szóról szóra haladt, és további szűréseket végzett: a gyakori, ám

[3] Alfanumerikus karakter az összes betű és szám karaktert jelenti. Nem tartoznak ide tehát szimbólumok, írásjelek.

[4] Újsor, tabulátor, szóköz karakterek

összességében semmitmondó névelőket (a, az, egy), angol megfelelőiket (a, an, the), és két kötőszót (és, is) nem vette be a vizsgálatba. Eltávolította ezek mellett a számokat (a számjegyeket tartalmazó szavakból a számjegyeket nem). Ezek az előfeldolgozási lépések a 3.3.6-os fejezetben részletezett végleges verzióban is szerepelnek.

Szóról szóra haladva a szűrés mellett a megmaradt szavakból 3 szavas listákat képzett, majd ezeket a szóhármas listákat mindkét listán iterálva összehasonlította, és mentette az egyező szóhármasokat.

3.3.3 Az első verzió problémái

Az első verzió számos problémát hordozott magában, amelynek eredménye lett a korlátozott használhatóság. Ezeket a következő pontok fejtik ki részletesen. A teljesítményre történő hatásukat pedig a 4.2-es fejezet részletezi.

3.3.3.1 Párhuzamosítás

Az első verzióban leírt algoritmus szekvenciális, nem jól kihasználva a manapság rendelkezésre álló többprocesszoros architektúrát. A feldolgozási folyamat egymás utáni lépései függenek az előzőtől, azonban a különböző adatok feldolgozása egymástól nem függ, így kihasználható az adatpárhuzamosság. Az algoritmust a megfelelő architektúrán különböző adatokra párhuzamosan futtatva rendkívül jelentős futási idő csökkenés érhető el.

3.3.3.2 Karakterláncok összehasonlítása

A program 3 szóból álló listákat hasonlít össze, amely karakterlánc alapú, ezért rendkívül erőforrásigényes. Ezen a listák helyett a szavakat egy karakterlánccá összefűző feldolgozás sem változtat. Számok összehasonlítása azonban a processzorok számára atomi műveletként elvégezhető. A szükséges közbenső lépés a karakterláncok számokká alakítása, erre tökéletesen alkalmas egy hash függvény, melyet a következőkben ismertetünk.

3.3.4 Hash függvény

Egy hash függvény egyirányú hozzárendelés valamilyen változó méretű adat és egy fix méretű adat között. Ebben az esetben ez rendre a szóhármasok egybefűzött karakterláncát, és egy számot jelent. Egy hash függvény determinisztikus, bármely bemenetre ad kimenetet. A különböző bemenetekre adott kimenetek a kimeneti

intervallumon a lehető legjobban elterítettek, azaz a különböző bemenetekre adott azonos kimenetek valószínűségét minimalizálja.

A hash függvényeket sokszor alkalmazzák kriptográfiai célokra. Ebben a visszafejthetőség megakadályozása fontos cél. Esetünkben erre nem volt szükség, ezért nem-kriptográfiai hash függvényt kerestünk, amelyek jobb teljesítményt nyújtanak.

Fontos kérdés még a hash függvény kimeneti intervalluma. A szóhármasok, mivel három szóból állnak, ezért viszonylag hosszúak. Ezekben a szóhármasokban alfanumerikus karakterek fordulhatnak elő, illetve szóköz és kötőjel („-”), a többi karaktert a 3.3.2-es fejezetben leírt módon kiszűri a program. A magyar ABC 44 betűből áll, ebből nyolc darab kettősbetű és egy darab hármasbetűt levonva 35 karakter marad. Ennek kis- és nagybetűs formáját is véve 70, emellé csatlakozik 10 számjegy, illetve a szóköz és a kötőjel, azaz összesen 82 karakterről beszélünk, és ezzel eltekintettünk a például külföldi nevekben előforduló további karakterektől. Egy 12 karakter hosszú szóhármasban (amely egyébként rövid, hiszen például két három, és egy négybetűs szóból állhat, illetve két szóközből) ez $82^{12} \approx 9{,}242 \cdot 10^{22}$ lehetőséget jelent. Ebből természetesen nagyon sok lehetőséget kizár, hogy a betűk csak bizonyos sorrendben következhetnek, hiszen csak úgy alkothatnak értelmes szavakat.

Azonban a fentiekből is látszik, hogy egy 32 bites szám ehhez nem elegendő, értékkészletük nagysága csupán $2^{32} \approx 4 \cdot 10^{9}$. 64 bites szám esetén az értékkészlet nagysága $2^{64} \approx 1{,}85 \cdot 10^{19}$. A fenti 12 karakteres példa is mutatja, hogy ez a tartomány sem fedi le az egyébként 12 karakternél jellemzően hosszabb szóhármasok esetén az összes lehetőséget. Architekturális megfontolásból mégis 64 bites számokkal dolgoztunk, ezeket ugyanis egyetlen gépi utasítással össze lehet hasonlítani a manapság elérhető processzorokkal. Ettől nagyobb értékek esetén több lépés szükséges, ami lassítaná a végrehajtást..

A program azonban nem dolgozza fel egyben az összes dolgozatot erőforráskorlátok miatt, ennek okait részletesen a 3.3.5-es fejezet ismerteti, egyelőre csak maga a tény lényeges. Egy ütemben 512 fájlt dolgoz fel, amelyek durva túlbecsléssel 50 000 szót, tehát nagyjából ennyi szóhármast is tartalmaznak. Ebben az esetben egyszerre 25 millió szóhármashoz tartozó hash értékről beszélünk. Ez az összes lehetséges hash értékhez képest nem olyan sok, akár 32 biten is kezelni lehet. Azonban, szintén architekturális adottságok miatt, egy 32 bites számon műveleteket végezni

ugyanannyiba kerül, mint egy 64 bites szám esetén. Memóriahasználat tekintetében pedig nem a hash értékek dominálnak, hanem a karakterláncok, e tekintetben sem jelentős a többlet.

A fentiek szerint egy 64 bites szám alkalmazása elég. A biztonság kedvéért a program vizsgálja az esetleges ütközéseket, és keres még nem használt értékeket a szóhármas tárolására, ezt a következő bekezdésben bemutatott hash táblák teszik lehetővé.

Hash alkalmazásával az összehasonlításban már csak számok vesznek részt. De a végén, ha nem csak az egyezés tényét szeretnénk megállapítani, hanem szeretnénk tudni, konkrétan melyik szavak egyeztek, akkor a hash értékeket vissza kell alakítani szöveggé. Erre hash táblákat alkalmaztunk, ezek segítségével lehetett egyező hash esetén visszakeresni, konkrétan mely szóhármas ütközött, emellett lehetővé vált az ütközéskeresés.

Az elérhető hash függvények közül végül a SpookyHash [14] nevű függvényt alkalmaztuk, amely nem kriptográfiai, kihasználja a modern architektúrát. 32, 64 és 128 bites változata létezik, mi a 64 bitest alkalmaztuk, úgy tervezve a megvalósítást, hogy szükség esetén viszonylag könnyen váltani lehessen például a 128 bites változatra.

3.3.5 Memóriakezelés

A dokumentumok feldolgozása jelentős mennyiségű memóriát igényel. Karakterlánc alapú feldolgozás esetén tulajdonképpen a teljes dokumentum végig a memóriában található. Hash használata esetén a 3.3.4-es fejezetben ismertetett hash táblákra van szükség, amelyek mérete rövid időn belül, körülbelül 512 dokumentum tárolása után elérte a fél GB nagyságrendet, hiszen ezek is tárolták a feldolgozott dokumentumok összes szóhármasát. A hash táblák memóriaigénye a dokumentumok számával lineárisan nő.

Emiatt szükség volt arra, hogy a teljes adathalmaz egyidejű feldolgozása helyett apróbb részeket dolgozzon fel a program, és lépésről lépésre haladjon végig az összes bemeneti dokumentumon. Minden apróbb rész végeztével a hash táblákat ki lehet üríteni a memóriából.

Természetesen ennek következtében egy dokumentumot többször betöltött és előkészített a program. Szükség volt valami megoldásra, amely a memóriában tart

bizonyos részeket, gyorsítva a feldolgozást, de mégsem foglal el rengeteg helyet. Elsőként egyetlen blokkot gyorsítótárazott az algoritmus. Ha üres volt a gyorsítótár, beletette a betöltött blokkot. Következő betöltéskor ellenőrizte, hogy a gyorsítótárazott blokkra vonatkozik-e a kérés, ha nem, akkor a gyorsítótár „kapott még egy esélyt”, azaz nem törlődött, és a kért blokk betöltésre került. Ha már „nem volt esélye” a gyorsítótárazott blokknak, akkor lecserélte az aktuálisan betöltött blokk. Ennek a megoldásnak az oka, hogy az algoritmus egy adott blokkot sorban összehasonlított a többivel, emiatt ezt a blokkot minden összehasonlításkor hivatkozta (célszerű volt tehát ezt gyorsítótárazni).

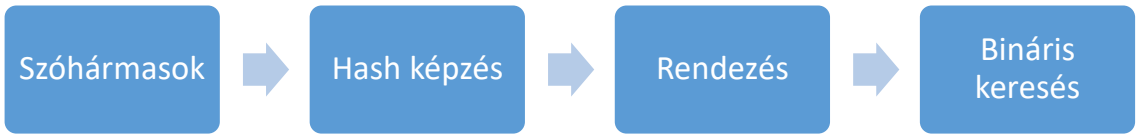
Azonban figyelni kellett arra, hogy a gyorsítótárazott blokkban található szóhármasokat ne ürítsük ki a hash táblából, azaz gyakorlatilag addig ne ürítsük ki a hash táblát, amíg a memóriában található az adott blokk. Ez a megoldás gyorsította a programot, de nem ez került a végleges verzióba.

Az előbb részletezett megoldás helyett bevezetésre került a kis és nagy blokk fogalma. A kis blokk a dokumentumoknak azt a halmazát jelenti, amelyeket egyszerre betölt, feldolgoz, összehasonlít a program. A nagy blokk pedig azt a halmazt, amelyet egyszerre a memóriában tart. A program tehát az első nagy blokk egymással történő összehasonlításával kezd, eközben betölti a benne található kis blokkokat a memóriába, és összehasonlítja azokat egymással. Ezután ezt a nagy blokkot összehasonlítja az összes másik nagy blokkal. Ezáltal mindig csak az összehasonlítás egyik felét kell betölteni és feldolgozni.

Fontos kérdés azonban a hash táblák kérdése. Az első ötlet szerint minden nagy blokk külön hash táblát kap, az egyik tehát folyamatosan a memóriában van, a másik pedig cserélődik, így biztosítható, hogy minden fájl szóhármasait tartalmazzák. De ez teljesen felesleges. Hiszen az egyezéseket keressük. Az egyezés mindkét dokumentumban megtalálható, azaz elérhető az egyik dokumentum felől és a másik dokumentum felől is, az egyik és a másik hash táblában is. Tehát az egyik hash tábla teljesen felesleges, mégpedig értelemszerűen az, amelyet mindig cserélnénk. Így a gyorsítótárazott nagy blokk betöltésekor felépíti a program az egyetlen hash táblát, amelynek a mérete a továbbiakban nem változik.

Ez a felismerés a hash tábla memóriaigényét kicsit több mint feleakkorára csökkentette az előző megoldásokhoz képest, illetve a felesleges lépések kihagyásával nagyságrendileg 10%-kal csökkentette a futási időt.

3.3.6 Végleges verzió



5. ábra – Az összehasonlítás lépései a végleges algoritmusban

Az algoritmusunk végleges verziója alkalmazza a korábbiakban leírtakat. Emellett további változtatások is szerepelnek, amelyeket ebben a fejezetben írunk le. Ebben a verzióban már természetesen a Wikipedia cikkei is bekerültek az összehasonlításba.

A fájlok beolvasása után (amely a dolgozatok és a Wikipedia esetén technikailag eltér) szűréseket végez, ezen lépések egyik részét 3.3.2-es fejezetben már leírtuk. Azokon a lépéseken túl belekerült a 3.1-es fejezetben említett átalakítási hibák által érintett fájlok kiszűrése. Azokat a fájlokat, amelyek valamilyen, az Unicode „Vegyes szimbólumok” blokkba tartozó karaktert tartalmaznak (valami oknál fogva az alkalmazott PDF átalakító program ilyen karaktereket helyezett a fájlokba), az algoritmus hibásnak jelöli. Ezek a fájlok az összehasonlításban üres fájlokként részt vesznek, a kimeneten pedig hibajelzést produkálnak, ez a 3.3.7-es fejezetben található meg bővebben. Továbbá készül egy feketelista a hibás fájlokról, ennek tartalmáról a 4.1-es fejezet tartalmaz információkat. Emellett további szűrőfeltételként az egy- és kétbetűs szavakat adtuk meg, ezek egyezése a kimenet számtalan helyen előforduló torzításért felelt, és egyáltalán nem jellemezte az adott két dokumentum hasonlóságát.

Ezután az algoritmus szóról szóra haladva a szűrés mellett a megmaradt szavakat szóközzel elválasztva összefűzte. Az így keletkezett karakterláncoknak képezte a hash értékét (részletesen 3.3.4-es fejezet), és eltárolta a hash értékhez tartozó szóhármast a 3.3.5-ös fejezetben ismertetett módon. Az összehasonlítás bemeneteként szolgáló listában a hash értékek egyediek voltak, a hash érték mellett azok előfordulásának számát tárolta a lista. Egy hash érték első előforduláskor tehát az algoritmus hozzáadja a listához a hash értéket 1 előfordulási darabszámmal, ezután pedig minden előforduláskor ezt a darabszámot növeli. Ez tette lehetővé a következő bekezdésben ismertetett bináris keresés alkalmazását, hiszen nem egyedi elemek esetén is csupán egyet talál meg.

Az összehasonlítás menete is különbözik az első verzióban látottaktól (3.3.2-es fejezet). A rövidebb listán iterálva bináris keresés segítségével keresi meg, hogy a másik

lista tartalmazza-e az iterációban aktuális hash értékeket. Ha igen, akkor hozzáadja a kimenethez az egyező szóhármast, amelyet a hash táblában a hash érték szerint keresve talál meg, illetve az egyezések számát, amely a két listában található előfordulási darabszám szorzata. Ez a szorzat azonban egyes esetekben hatalmas volt, ami torzította a végeredményt, különösen úgy, hogy többnyire semmitmondó szóhármasok egyezése mutatott magas darabszámot, ezért ezt az értéket 20-ban maximáltuk.

Az összehasonlítási folyamatban a másik fontos különbség a Wikipedia adathalmazának kezelése, hiszen a Wikipedia cikkeit egymással nem, csak a dolgozatokkal kell összehasonlítani.

Az összehasonlítás végeztével az összegyűjtött egyezéseket a program a 3.3.7-es fejezetben megtalálható módon lementi.

3.3.7 Kimenet

A program kimenete tartalmazza az összehasonlítás eredményét, azaz mely dokumentumpárban mely trigramok egyeztek meg, és hány darab egyező trigram található. Hibás bemeneti dokumentumok esetén az egyező trigramok száma -1. A dokumentumok azonosítók szerint szerepelnek, amelyet egy táblázat segítségével lehet visszakeresni.

Ez a kimenet önmagában viszonylag nyers, további feldolgozás szükséges a fogyasztható formába öntéshez, ezt a 3.4-es fejezet ismerteti. Azonban az egyezések konkrét szavak formájában itt találhatóak meg.

A program kimenete meglehetősen nagy helyet foglal, ezért bevezettünk egy olyan kimeneti opciót, amely csak a talált egyező trigramok számát menti el, a dokumentumok azonosítójával együtt. Ekkor a kimenet mérete a töredékére csökken, és amennyiben csak statisztikai adatokra van szükség, tökéletesen megfelel a célnak.

3.4 Kimenet feldolgozása

A 3.3.7-es fejezetben tárgyalt kimeneti fájlok tartalmazzák az adott dokumentumpárban megegyező trigramokat, valamint a szövegfájl utolsó sorába beírjuk, hogy hányas számú fájlokról volt szó, valamint hány trigram egyezett. Ezeket az értékeket

minden fájl utolsó sorából kiolvassuk, majd a kiválasztott feldolgozási technológiával eltároljuk (.csv[5] fájlba mentés, táblázat oszlopaiba beillesztés).

A feldolgozás egyik legfőbb célja egy alapvető automatizált szelekció végrehajtása, hogy minél kevesebb adatot kelljen manuálisan elemezni. Pont emiatt építettünk bele a programba egy szabályozható korlátot, amely a küszöbszám alatti egyezésekkel rendelkező dokumentumpárokat „eldobja", azaz a további elemzésben nem foglalkozik velük. Ezt mindenképpen szükséges megtenni, hiszen – a 4.3.1-es fejezetből is jól látható módon – rengeteg olyan pár van, melyek között minimális az egyezések száma, ezek feldolgozása jelentősen megnövelné az elemzési időt.

3.5 További vizsgálatok

A szóhármasok alapján történő összehasonlítás egy jó alapot adott a hasonló dokumentumok kiszűrésére, azonban a segítségével nem kaptunk meg minden lényeges adatot. A hiányzó részeket az itt ismertetett vizsgálatok adják meg.

3.5.1 Egyezések hossza

A 3.3-as fejezetben ismertetett algoritmus választ adott a hasonlóságok meglétére, de az egybefüggő egyező részek hosszát nem adta meg. Erre egy második körbeli vizsgálat ad választ, amelyet csupán a dokumentumok egy részére, a trigram algoritmus szerint nagyobb egyezésszámot mutató dokumentumpárokra futtattunk le, azok esetében fontos egy árnyaltabb kép.

Ez az algoritmus felépítését tekintve hasonlít a trigram algoritmushoz. Első lépésként a már ott megismert fájlbeolvasás és ugyanazok a szűrőfeltételek szerinti átalakítás megy végbe. Azonban szóhármasok helyett az egyes szavakból képez hash-t.

Összehasonlítás során a résztvevő két dokumentum közül az egyik hash listáján egy ciklusban iterál, majd egy belső ciklusban megkeresi a másik dokumentumban az első egyezést. Ha nem talál ilyet, folytatódik a külső ciklus. Ha talál, akkor elkezdi vizsgálni mindkét lista esetén a következő szavak egyezését. Ez a folyamat addig tart,

[5] Comma-Separated Values, azaz egy olyan fájl, ahol az egymást követő adatok vesszővel vannak elválasztva egymástól. Rendszerint olyankor alkalmazzák, ha automatizált módon szeretnék kiolvasni a fájl tartalmát, ami ilyen módon sokkal gyorsabban megoldható, mint például egy táblázatkezelő program által készített táblázatból.

amíg a szavak megegyeznek. Ha talált legalább négy egymás után következő szót, amelyek egyeznek, tárolja kimenetként az egyező listát. Ha a belső ciklus nem ért véget az egyező szóláncok vizsgálatakor, akkor további egyezéseket keres a külső ciklus által kijelölt szóra, ezután az ismertetett folyamat zajlik le újra.

Annak megakadályozására, hogy egy-egy hosszú egyezést annak minden egyes szavára mindig egyel rövidebb láncként megtalálja, a következő szavak vizsgálata előtt minden alkalommal ellenőrzi azt is, hogy az előző szavak egyeznek-e. Amennyiben igen, további vizsgálat nem szükséges, hiszen egy korábbi iteráció már megtalálta az aktuálistól korábban kezdődő láncot.

Kimenetként a dokumentumpárokban szereplő összes egyezést lementettük, a dokumentumok azonosítóival együtt. Statisztikát alapvetően a leghosszabb egyezések alapján készítettünk. Ezeket a fejezet ismerteti. Emellett lehetőség van az összes egyezés hosszának lementésére, amely olyan formátumban készül, hogy könnyen lehessen R nyelvű szkriptet készíteni belőle, ezáltal elemzéseket lehet lefolytatni.

3.5.2 Metaadatok

Az egymással hasonlóságokat mutató dokumentumok esetén nem csak a hasonlóságok, hanem a dokumentum adatai is beszédesek. Ki a dokumentum szerzője, kik a konzulensek, melyik évben keletkezett, illetve milyen típusú dokumentumról van szó.

A dokumentumok kötött felépítése lehetővé teszi, hogy különböző heurisztikákat alkalmazva ki lehessen nyerni belőlük a fent megnevezett adatokat, majd ezután össze lehessen őket hasonlítani. Emiatt, ha nem is azonnal, de a manuális áttekintésnél gyorsabban el lehetett érni elég nagy pontosságot az adatok megtalálása esetén.

A dolgozatok nagy többségében például megtalálható a hallgatói nyilatkozat, amelyben könnyen beazonosíthatóan szerepel a hallgató neve.

Másik esetben egyáltalán nem szerepelt hallgatói nyilatkozat, ekkor a „készítette” szót a dokumentum elején keresve szintén könnyen megtalálható a dolgozat szerzője. Más esetben a feladatkiírás szerepel a dokumentumban, amely szintén jól behatárolható helyen tartalmazza a keresett nevet. Végső esetben a „tanszék” vagy „kar” és a „konzulens” szavak közötti részt tekintette az algoritmus, e között a név mellett a diploma címe is megtalálható, de a kettő sorrendje változó, és nincsenek támaszpontok, ami alapján ettől

pontosabban be lehetne határolni a szerzőt. Legrosszabb esetben egyszerűen a dolgozat legelejétől számítva 180 karaktert tárolt le az algoritmus, általában a dolgozat legelején szerepel a szerző neve, ebben valószínűleg (de nem biztosan) megtalálható.

Azokban az esetekben, amikor a név mellett egyéb szavak is bekerültek a kiragadott részbe, ha okozhatnak is hibákat, összességében nem befolyásolják az összehasonlítást. Egyrészt ezekre a módszerekre más híján, viszonylag ritkán kerül sor. Másrészt ugyanaz a szerző két dokumentuma struktúrájában többnyire teljesen megegyezik. Ha az egyik esetén a hallgatói nyilatkozatban találta meg az algoritmus a nevet, valószínűleg a másikban is úgy fogja. Ha az egyikben a tanszék és a konzulens szó közötti részben, akkor a másikban is sok eséllyel ugyanígy történik.

A konzulensek megjelölése esetén kevésbé kötött a dolgozatok szerkezete. A „konzulens” szó nagyrészükben megtalálható, azonban nincs egy körülhatárolható rész, amely között keresni kell a nevet. Sokszor szerepel utána a „Budapest” szó utána, vagy a tartalomjegyzék, netán a hallgatói nyilatkozat. De előfordul, hogy egy ilyet sem talál a program, ekkor egy előre meghatározott hosszú karakterláncot ment le a „konzulens” szó végétől kezdve. Olykor a konzulensre témavezetőként hivatkoznak a munkák.

Az év megtalálása talán a legegyszerűbb feladat, hiszen a dokumentum négybetűs szavai között keresve a szavakat megpróbálja számmá alakítani a program, ha sikerül, megnézi, hogy 2000 és 2016 közé esik-e ez a szám. Az első megfelelő találatot menti.

Tanszéket a konzulenshez hasonlóan keres, itt az „egyetem” vagy „kar” szavak után nem sokkal következő „tanszék” szóval bezárólag szereplő részt menti ki, de legfeljebb a tanszék szó végétől számított 60 karaktert.

Egy további adatot keres a program, hogy szakdolgozatról, vagy diplomatervről van-e szó. Legelőször a hallgatói nyilatkozatban próbálja megtalálni a szavakat, ha nem található nyilatkozat, akkor a dokumentum egészében. Sokszor előfordul, hogy mindkettő szó szerepel (például a diplomaterv hivatkozik a szakdolgozatban szereplő munkára), ekkor nincs prioritás, a típus nem állapítható meg.

A fent leírt heurisztikák meglepően pontos eredményeket adnak, de az összehasonlítás ennek ellenére a nevekben nem teljes karakterhelyes egyezést vizsgál, hanem szavanként ellenőrzi, megtalálható-e az adott szó a másik dokumentum oda vonatkozó karakterláncában. Ha kettő, vagy attól több egyezés található, a program egyezést jelez. A pontosság érdekében az összehasonlításba feketelista is építhető.

Például kiszűri a „dr” vagy „tanszék” szavakat. Ekkor azonban úgy tűnt, hogy romlott a pontosság, ezért végül a végleges futtatáskor üres volt a feketelista, de benne hagytuk a logikát az algoritmusban.

A program összehasonlítás utáni kimenetét természetesen a fejlesztés során folyamatosan figyeltük, és annak megfelelően vizsgáltuk egyesével a problémásnak talált dokumentumok szerkezetét (amelyben nem ismerte fel a program a megfelelő részt, vagy irreleváns részeket talált meg), hogy további szavakat keressünk, vagy a lépések sorrendjét megváltoztassuk a jobb eredmény elérése érdekében.

A fentiek alkalmazásával elég nagy pontosságot elértünk, azonban egy további nehézség is fellépett. Ugyanis angol nyelvű dokumentumok is a feldolgozás részei voltak. Ezek azonban néha magyarul tartalmazták a hallgatói nyilatkozatot például. A program a fent ismertetett kulcsszavak angol verzióit is keresi, ha nem találja a magyar nyelvűeket. „Nyilatkozat” helyett „declaration”, „konzulens” helyett „consultant”, „advisor”, vagy „supervisor” szavakat. Angol nyelvű dokumentumok esetén a kulcsszavak megtalálása után (vagy előtt) fix hosszúságú karaktersorozatot ment le a program, az ilyen dolgozatok viszonylag kis aránya miatt a magyarhoz képest kevésbé „ügyes” a program (nem igazán lehet a magyarhoz hasonló statisztikailag jelentős mintákat találni), valamivel pontatlanabb.

A program kimenetét a 4.5-ös fejezet ismerteti, ha nem is tűpontosak, statisztikai szempontból irányadónak tekinthetők.

4 Eredmények

A plágiumkereső alkalmazásunk egyes lépéseinek lefuttatása után többféle formájú kimenetet kaptunk. A kimenetek lehetséges formáját már bemutattuk a 3.4-es fejezetben, most a kimenetek tartalmát tárgyaljuk.

A különböző kimenetek mindegyikénél külön szerepelnek a diplomatervek és szakdolgozatok egymással történő összehasonlításának eredményei, valamint a dolgozatok és a magyar nyelvű Wikipedia hasonlóságára vonatkozó eredmények. A könnyebb átláthatóság érdekében kék színű grafikonon a dolgozatok egymás közötti eredményei szerepelnek, míg narancssárga színű a Wikipedia cikkeivel történő összehasonlítás eredménye.

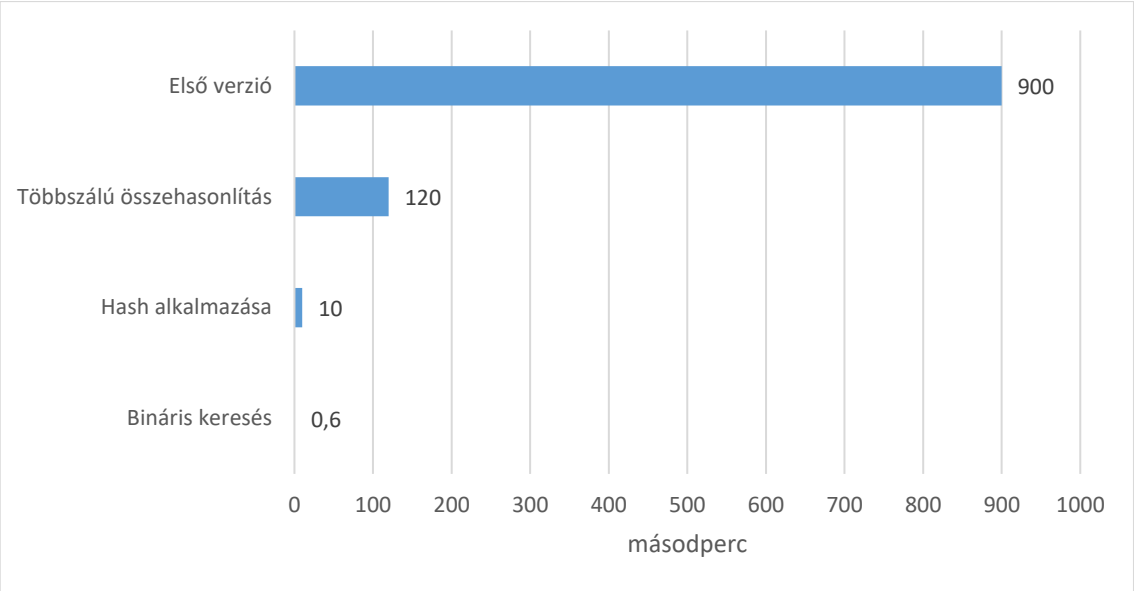
4.1 Hibás bemeneti fájlok

A 3.1-es fejezetben ismertetett PDF átalakítási hibákkal érintett fájlokat az algoritmus rögzítette egy feketelistába. 328 darab ilyen dolgozatot talált, ezek a 4.3-es fejezet eredményei között a 0 egyezés kategóriában szerepelnek.

4.2 Algoritmus teljesítménye

A 3.3-as fejezetben ismertetett algoritmus különböző verzióinak teljesítményét az itt látható diagram szemlélteti, 8 dolgozat összehasonlításával. A teszteléshez használt számítógép paraméterei a következőek:

- Intel 3610QM, 4 mag, Hyper Threading, 2,30GHz (3,30GHz turbó)
- 12 GB DDR3 RAM
- Samsung 840 SSD 512GB



6. ábra – Algoritmus teljesítménye 8 dolgozat összehasonlításakor

A diagramon nem szerepel a végleges verzióban szereplő gyorsítótárazási megoldás, ennek hatása ugyanis nagyobb mennyiségű dokumentum vizsgálata esetén érezhető. A fenti diagramon 1500-szoros gyorsulás látható, a diagram adatait a dolgozatok teljes halmazára (tehát a Wikipedia cikkei nélkül) vetítve a nagyságrendileg fél éves futási időt sikerült lecsökkenteni 6 óra hosszára.

A végleges verziót egy másik konfiguráción futtattuk, a Microsoft Azure felhőszolgáltatásában, egy F8-as típusú virtuális gép példánnyal, amely a következő paraméterekkel rendelkezik:

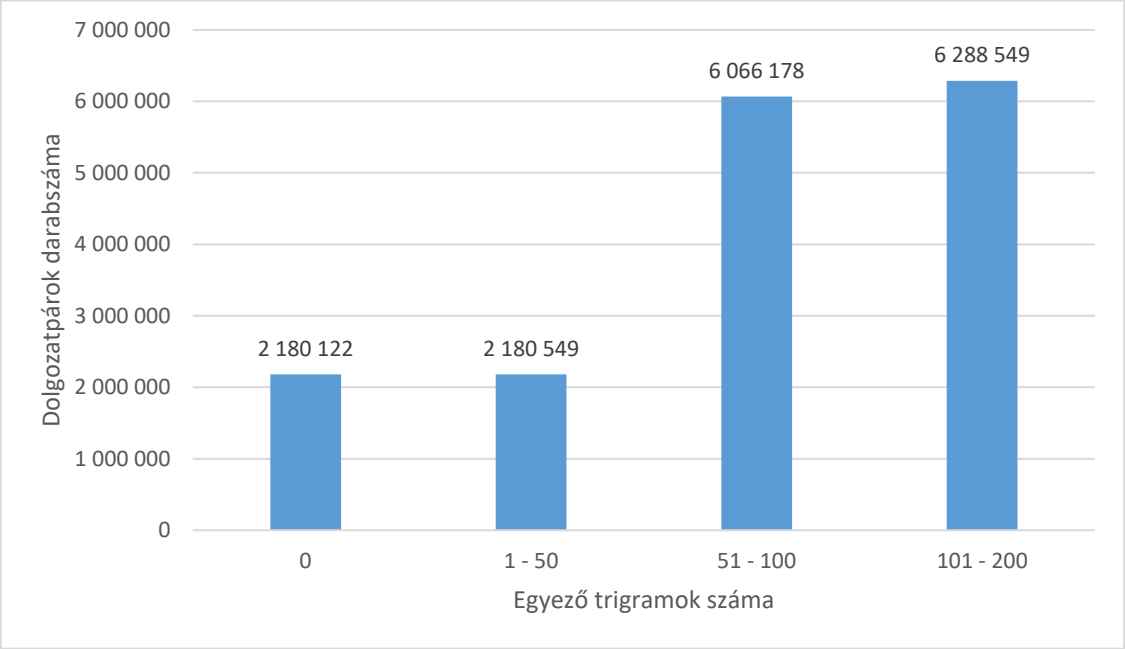
- Intel Xeon E5-2673 v3 2,4GHz, a virtuális gépben 8 mag volt elérhető
- 16 GB RAM
- 120GB virtuális HDD

Ez a virtuális gép gyorsabb, mint a másik konfiguráció. A végleges verzió futási ideje, amely jobb gyorsítótárazási megoldást alkalmaz, a dolgozatokra vonatkozóan nagyságrendileg 2 óra. A Wikipedia cikkeit is tartalmazó vizsgálatról csak a végleges verzió esetén van adatunk, nagyságrendileg 9 óra hosszan futott.

4.3 Egyező trigramok száma

Ebben a fejezetben a dokumentumokban megtalálható egyező trigramok számát ismertetjük hisztogramok formájában, amelyek gyors áttekintést nyújtanak az eredményekről.

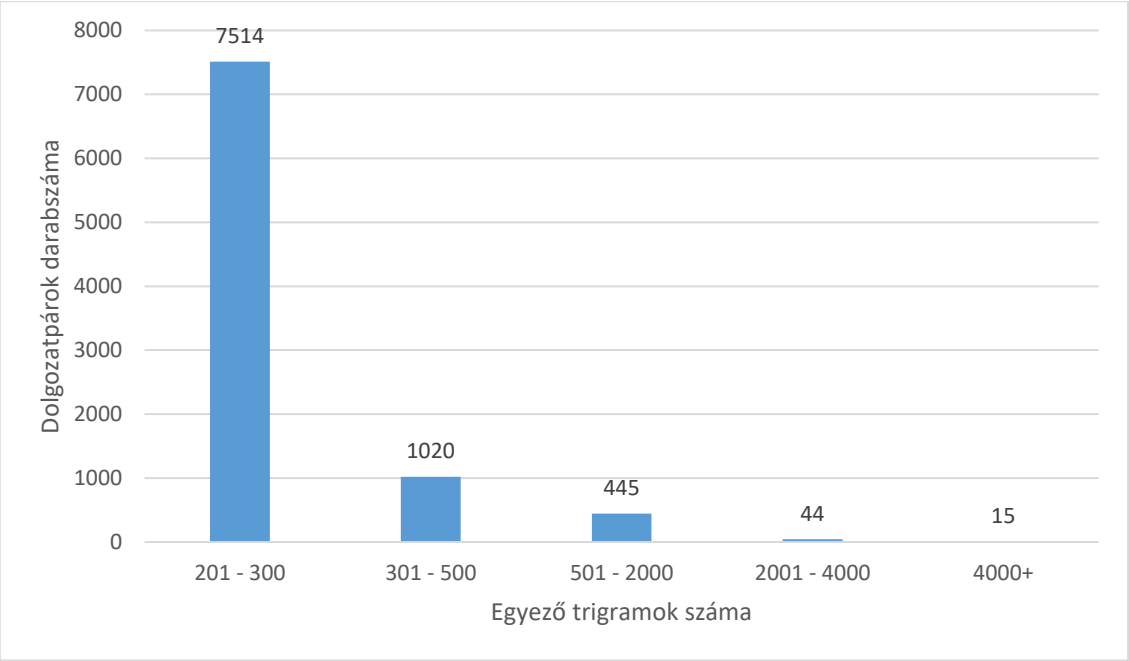
4.3.1 Dolgozatok összehasonlítása egymással



7. ábra – Kis átfedésű dolgozatok (legfeljebb 200 egyező trigram)

Az összehasonlításban szereplő 5 784 dolgozat összesen 16 724 436 összehasonlítást eredményezett. Ezek túlnyomó többsége előre várható módon kevés egyezést mutatott ki. Feltűnően sok ezek közül az, amelyek esetén az egyezések száma nulla. Ennek oka, hogy a dolgozatok egy részét nem sikerült megfelelően PDF formátumból szövegformátumba átalakítani. Ezeket a hibákat azonban a program felismeri, és kiszűri, viszont az összehasonlításban üres dokumentumokként részt vettek.

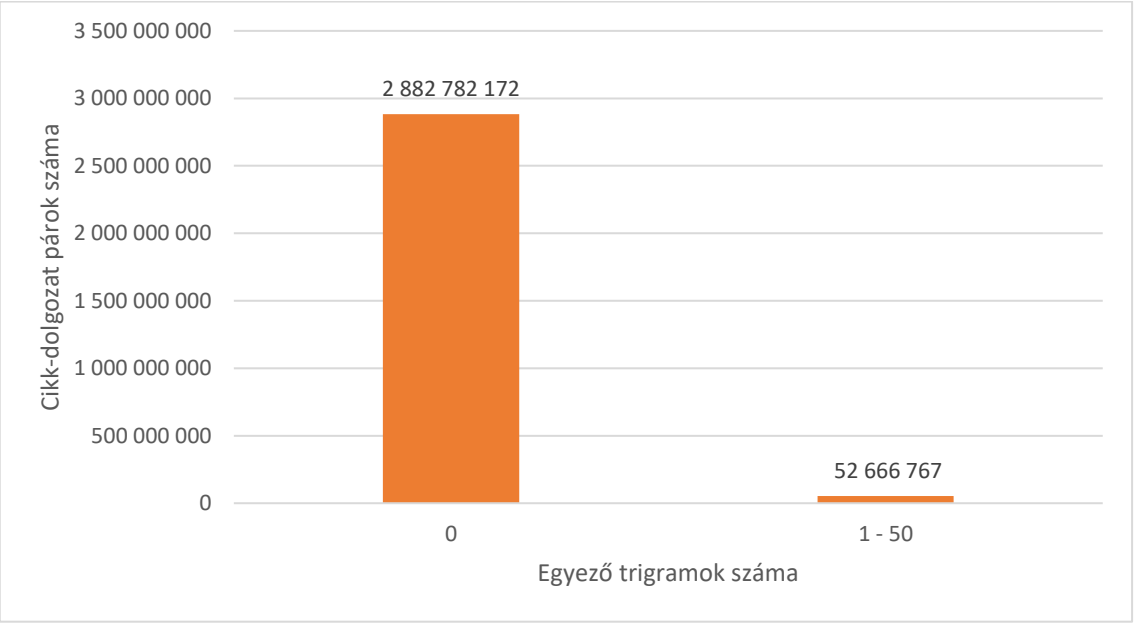
Kevés egyezés egyéb oka pedig a dolgozatokban megtalálható hétköznapi szókapcsolatok, amelyek egymástól teljesen eltérő témájú munkákban ugyanúgy megtalálhatóak.



8. ábra – Nagy átfedésű dolgozatok (legalább 201 egyező trigram)

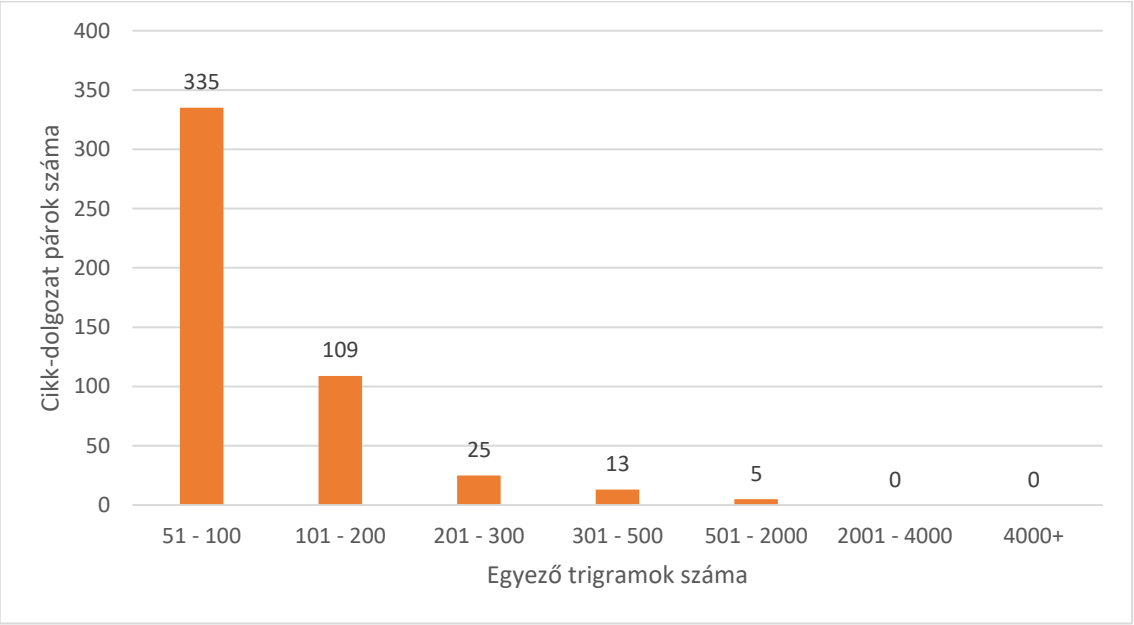
Magas trigram egyezésszámot jóval kevesebb dokumentumpár mutatott, a skála három nagyságrenddel kisebb. Itt már a hétköznapi szókapcsolatok mellett tényleges tartalmi hasonlóság is előfordul. A dolgozatok nagyságrendileg 7 000 – 14 000 szót tartalmaznak. Ez alapján 2 000 egyezés felett nagyobb tartalmi egyezés feltételezhető, de 500 egyezés felett is hasonlíthatnak egyes részek. A legtöbb trigram egyezést mutató dokumentumokat további elemzéseknek vetettük alá.

4.3.2 Dolgozatok összehasonlítása a magyar nyelvű Wikipediával



9. ábra – Kis átfedésű dolgozat – cikk párok (legfeljebb 50 egyezés)

A Wikipedia adatbázisával történő összehasonlítás körülményei különböztek a diplomamunkák egymással történő összehasonlításainak körülményeitől, ezt a 3.2-es fejezetben ismertettük. A cikkek nagy számának köszönhető a rengeteg összehasonlítás, és az irreleváns cikkeknek pedig a 0 egyezést mutató oszlop nagysága. Látható módon több nagyságrenddel kisebb azoknak a cikk–dolgozat pároknak a száma, amelyek egyáltalán tartalmaznak bármiféle hasonlóságot.



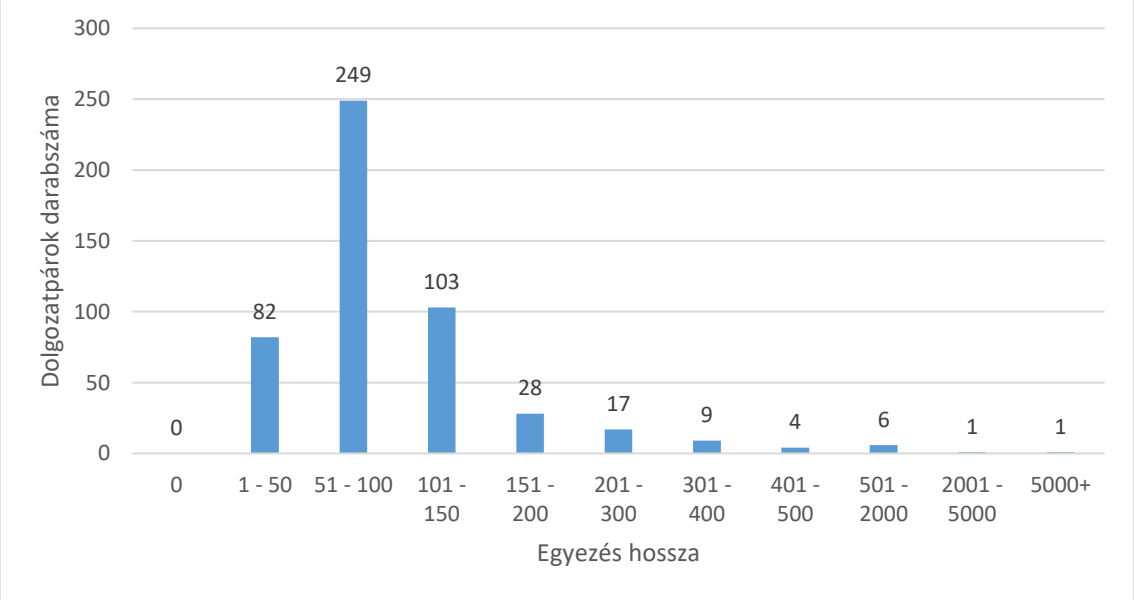
10. ábra – Nagy átfedésű dolgozat – cikk párok (legalább 51 egyezés)

Ismét nagyságrendeket ugrottunk a következő intervallumhoz tartozó párok esetén, itt nagyon kevés egyezést találunk. Köszönhetően egyrészt a cikkek rövidségének, és – a diagram szerint legalábbis – a közvetlenül átvett tartalom kis mennyiségének. 100 egyezés felett talán, 200 felett biztosan érdemes egy kicsit utánanézni a dokumentumoknak, esetleg a 4.4.2-es fejezet eredményeivel egyetemben.

4.4 Egyező szóláncok hossza

Ebben a fejezetben a 3.5.1-es fejezetben ismertetett algoritmus által adott eredmények találhatók, azaz a dolgozatok, illetve Wikipedia cikkek összehasonlításakor milyen hosszú szóláncok egyeztek. Elsőként a dolgozatok egymás közötti, másodjára a Wikipediával történő összehasonlítás eredményei következnek.

4.4.1 Leghosszabb egyezések a dolgozatok összehasonlításakor



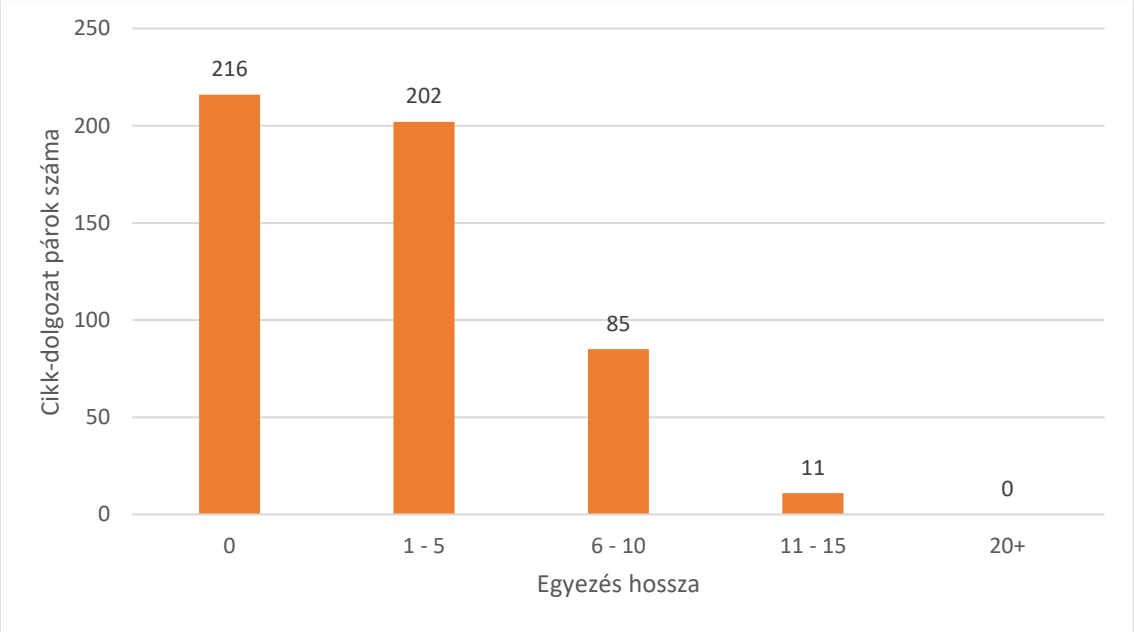
11. ábra – Adott hosszú egyező szóláncokat tartalmazó dolgozatpárok

A leghosszabb egyezések keresésekor csupán az 500 legtöbb egyezést mutató dolgozatot vizsgáltuk meg, a diagramon látható a dolgozatpárok száma a leghosszabb szólánc egyezés hosszának függvényében.

A fentiekkel ellentétben a felső intervallum felől közelítjük a bemutatást. Belekerült ugyanis az összehasonlítási folyamatban egy fájl másolata is. Önmagával természetesen teljes mértékben egyezik a fájl, ezért a hosszú egyezés szám. A második helyezett esetében azonban nincs ilyenről szó. A legtöbb dolgozatpárt 51 és 100 hosszú

leghosszabb egyezés kategóriában találjuk, ez önmagában körülbelül egy közepes hosszúságú bekezdés teljes egyezésének felel meg. E felett pedig a fél oldal nagyságrendű teljes egyezéseket találjuk.

4.4.2 Leghosszabb egyezések a magyar nyelvű Wikipediával

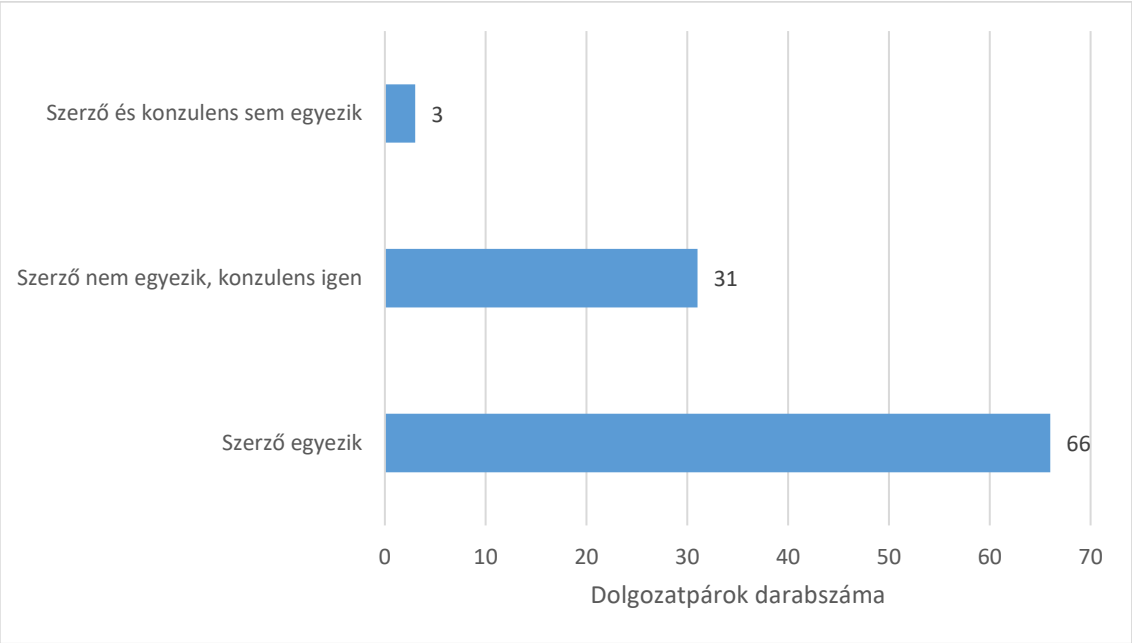


12. ábra – Adott hosszú egyező szóláncot tartalmazó dolgozat – cikk párok

Wikipedia esetén meglehetősen rövidek a dolgozatokban található leghosszabb egyezések, legalábbis azt az 500 dokumentumpárt tekintve, ahol a trigramok egyezésszáma a legnagyobb. A nulla oszlop meglehetősen érdekes. A 3.5.1-es fejezetben részletezett algoritmusban van ugyanis egy határérték, azaz a négy szónál rövidebb egyezéseket nem veszi figyelembe. A diagramon látható 216 dokumentumpár esetén tehát kizárólag három hosszú egyezések szerepelnek, amely arra utal, hogy valószínűleg nem is történt semmiféle másolás, csupán hétköznapi kifejezések miatt mutatott az algoritmus sok egyezést.

A többi egyezést tekintve is csupán egy-egy mondat az ami a dolgozatok és a Wikipedia cikkek között egyezik.

4.5 Dolgozatok adatainak összehasonlítása



13. ábra – A 100 legtöbb szóhármas egyezést mutató dokumentumpár adatai [15]

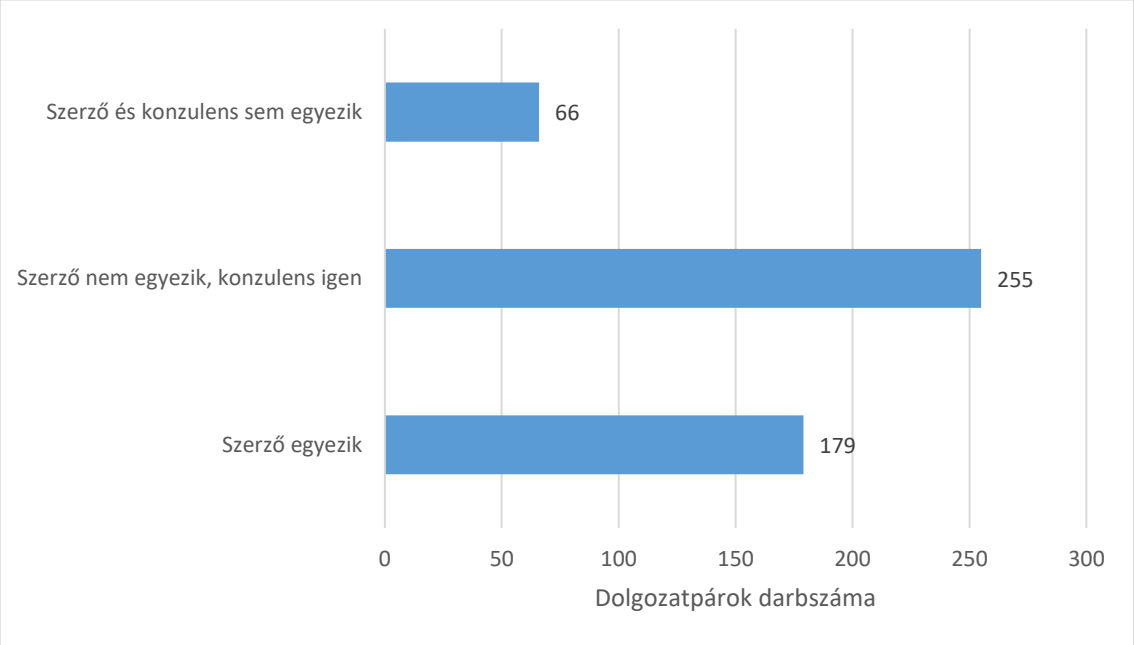
Talán a legérdekesebb elemzés annak a megállapítására vonatkozik, hogy a legtöbb egyezést mutató dolgozatpárok tagjai milyen viszonyban állnak egymáshoz. Azaz ki a szerzőjük, ki volt a szerzőt segítő konzulens. A fenti diagramon az látható, hogy a 100 legtöbb egyezést mutató dokumentumpár esetén milyen adatokat kaptunk.

Ahogyan várható volt a legtöbb dolgozat esetén azonos a szerző, tehát a szerző szakdolgozata és diplomamunkája szerepelt egy párban, amelyeket hasonlónak mutatott ki az algoritmus. Ez nem meglepő, hiszen sokan a szakdolgozat témájával foglalkoznak a diplomatervben is, és természetesen felhasználják saját korábbi eredményeiket a később írt diplomatervekben.

Talán az sem meglepő, hogy a dolgozatok másik része nem ugyanattól a szerzőtől származik, de ugyanaz volt a témavezető mindkét esetben. Előfordulhat, hogy egy közös projektmunka eredménye a két dolgozat, vagy csupán nagyon hasonló a téma, esetleg több közös forrást is megtalálunk.

Ha kis számban is, de találtunk azonban olyan dokumentumokat is, amelyek esetén sem a szerző, sem a konzulens nem egyezik. Közös források felhasználása, nagyon hasonló téma ugyanúgy oka lehet ennek az eredménynek, akár különböző tanszékeken párhuzamosan futó kutatások is okozhatják.

A következő diagramon összehasonlításképpen a legtöbb egyezést mutató lista első 500 helyezettje látható.



14. ábra – Az 500 legtöbb szóhármas egyezést mutató dokumentumpár adatai

Ha az első 500 dokumentumot vesszük figyelembe, akkor látható, hogy kisebbségbe kerültek az azonos szerzőtől származó dokumentumok. Ahogyan csökken az egyezésszám, egyre valószínűbb, hogy mástól származó dokumentumokban szerepelnek ugyanazok a részek.

A vizsgált lista bővítésének legnagyobb nyertese a „szerző nem egyezik, de a konzulens egyezik” kategória. Hasonló témák tehát rendszerint egy konzulenshez tartoznak.

Azon dolgozatok aránya is jelentősen megnőtt, ahol sem a konzulens, sem a szerző nem egyezik. A fenti diagramon látható elenyésző számokra lehetett talán magyarázatot találni, ilyen nagyságrend mellett azonban ettől tartózkodnánk.

5 Összegzés

A korábbiakban ismertetett kutatásunk után összegeznénk munkánkat, kezdve a plágiumkeresés tapasztalatairól, kitérünk a BME VIK dolgozataival kapcsolatos eredményekre röviden, valamint bemutatjuk a fejlesztési lehetőségeket.

5.1 Plágiumkeresés

Kutatásunk során sikerült egy megoldást kidolgozni, amely segítségével meg tudtuk vizsgálni a dolgozatokat plágiumkeresés céljából. Sikerült a vizsgálatot nagy mennyiségű dolgozaton végrehajtani, az eredmény belátható időn belül elkészült. Sikerült a vizsgálatba bevonni a magyar nyelvű Wikipedia cikkeit, így demonstrálni tudtuk a külső forrásokkal történő összehasonlítást is.

Ahogyan a Bevezetőben már említettük, tudomásunk szerint a BME keretein belül ilyen vizsgálatot elsőként végeztünk. Sikerült egy áttekintő képet adni a dolgozatok egyezéseinek mértékét tekintve, valamint dokumentumpáronként rendelkezésre állnak az eredmények további kutatáshoz.

5.2 BME VIK Diplomaterv állomány

A fenti vizsgálatok alátámasztják, hogy a BME VIK szakdolgozat és diplomaterv állománya esetén a nagy számú egyezések többségéért saját munkák hasonlósága felel, amelyek természetesen léteznek, hiszen gyakran ugyanarról a témáról készül egy hallgató szakdolgozata és diplomaterve is. Gyakori eset továbbá, hogy azonos a témavezető a két dolgozat készítése során, valószínűleg hasonló témában születtek a dolgozatok.

A Wikipedia esetén kiemelkedően alacsony az egyezésszám, szó szerinti átvétel tehát szinte alig történik a szakdolgozatokban, illetve diplomatervekben.

5.3 Fejlesztési lehetőségek

A jelen dokumentumban tárgyalt vizsgálatok jó alapot adnak, ám közel sem tökéletesek. A sokféle javítási lehetőség közül elsőként a 3.1 fejezetben ismertetett PDF átalakítási nehézségeket lenne érdemes kiküszöbölni, egy pontosabb bemeneti adathalmaz jelentősen javítana a vizsgálat pontosságán is. Ez utóbbin további plágiumkeresési technikák alkalmazása is segíthet.

A vizsgálat jelenlegi külső forrása, a magyar Wikipedia mellett fel lehetne használni az angol Wikipedia cikkeit is. Ez azonban nagyságrendekkel több adat, amely vizsgálatához további optimalizációk szükségesek az algoritmus tekintetében. Érdemes lenne megvizsgálni egy GPU alapú implementáció létjogosultságát is.

Kidolgozott módszereink hatékonyságának demonstrálására a BME VIK 6 évnyi dolgozatállományát használtuk, ugyanakkor módszereink könnyen általánosíthatók dokumentumok sokkal szélesebb körére is, mely jelenleg további kutatásaink egyik fontos fókuszpontja.

6 Irodalomjegyzék

[1] „BME VIK Diplomaterv portál,” [Online]. Available: http://diplomaterv.vik.bme.hu.

[2] A. Z. Abu Bakar, R. Ibrahim és N. Zakaria, „Databases and Text Processing Applications,” UNIVERSITI TEKNOLOGI MALAYSIA, ISBN 978-983-52-0631-3, Available: https://www.academia.edu/1458324/PLAGIARISM_DETECTION_TECHNIQUES.

[3] E. Chernyak, „Some Thoughts on Using Annotated Suffix Trees for Natural Language Processing,” National Research University – Higher School of Economics, Moscow, Russia, 2014.

[4] E. Mansour, A. Allam, S. Skiadopoulos és P. Kalnis, „ERA: efficient serial and parallel suffix tree construction for very long strings,” *Proceedings of the VLDB Endowment, Volume 5, Issue 1,* pp. 49-60, September 2011.

[5] D. Kelley, Automata and Formal Languages: An Introduction, ISBN 0-13-497777-7, London: Prentice-Hall International, 1995.

[6] K. Ito, H. Nakajima, K. Kobayashi, T. Aoki és T. Higuchi, „A Fingerprint Matching Algorithm Using Phase-Only Correlation,” IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences Vol.E87-A No.3 pp.682-691, 2004.

[7] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. Della Pietra és J. C. Lai, „Class-based n-gram models of natural language,” *Computational Linguistics Volume 18 Issue 4,* pp. 467-479, December, 1992.

[8] S. M. Alzahrani, N. Salim és A. Abraham, „Understanding Plagiarism Linguistic Patterns, Textual Features, and Detection Methods,” IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) Volume 42, Issue 2, Page 133-149, 12 May 2012.

[9] A. K. Jain és R. C. Dubes, Algorithms for clustering data, Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1988.

[10] J. A. Hartigan és M. A. Wong, „Algorithm AS 136: A K-Means Clustering Algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics) Vol 28. No. 1.,* pp. 100-108, 1979.

[11] „PDF to Text," Trisun, [Online]. Available: http://www.pdf-helper.com/pdf-to-text/. [Hozzáférés dátuma: 22. október 2016.].

[12] „Wikipedia dumps," [Online]. Available: https://dumps.wikimedia.org/huwiki/. [Hozzáférés dátuma: 22. október 2016.].

[13] „Wikipedia Extractor," Medialab, Università di Pisa, [Online]. Available: http://medialab.di.unipi.it/wiki/Wikipedia_Extractor. [Hozzáférés dátuma: 22. október 2016.].

[14] B. Jenkins, „SpookyHash: a 128-bit noncryptographic hash," [Online]. Available: http://burtleburtle.net/bob/hash/spooky.html. [Hozzáférés dátuma: 22. október 2016.].

[15] S. Few, *Save the Pies for Dessert,* Visual Business Intelligence Newsletter, Perceptual Edge, August 2007.

[16] N. Meuschke és B. Gipp, „State-of-the-art in detecting academic plagiarism," *International Journal for Educational Integrity, vol. 9, iss. 1,* pp. 50-71, 2013.

Függelék

Kimeneti fájlok szerkezete:

```
<id_1>-<id_2> (<összes_db>)
<db> <szóhármas_1>
<db> <szóhármas_2>
…
<db> <szóhármas_n>

…
<id_n-1><id_n> (<összes_db>)
<db> <szóhármas_1>
<db> <szóhármas_2>
…
<db> <szóhármas_n>

<id_1>,<id_2>,<összes_db>;<id_n-1>,<id_n>,<összes_db>
```