VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV BIOMEDICÍNSKÉHO INŽENÝRSTVÍ
DEPARTMENT OF BIOMEDICAL ENGINEERING

VYHODNOCENÍ PODOBNOSTI PROGRAMOVÝCH KÓDŮ
PLAGIARISM DETECTION OF PROGRAM CODES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Jakub Kašpar

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Martin Vítek, Ph.D.

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Biomedicínské inženýrství a bioinformatika**
Ústav biomedicínského inženýrství

Student: Bc. Jakub Kašpar ***ID:*** 144050
Ročník: 2 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Vyhodnocení podobnosti programových kódů

POKYNY PRO VYPRACOVÁNÍ:

1) Seznamte se podrobně s definicí plagiátorství z pohledu programových zdrojových kódů a nastudujte možnosti současných metod jeho detekce posouzením podobnosti kódů. 2) Vytvořte vlastní databázi vhodnou pro testování plagiátorství. 3) Nalezněte příznaky pro detekci plagiátorství a v prostředí Matlab separátně otestujte jejich vhodnost na vytvořené databázi. Dosažené výsledky diskutujte. 4) Realizujte detektor plagiátorství založený na kombinování vhodných příznaků. 5) Navržený detektor otestujte a dosažené výsledky statisticky zpracujte. 6) Program opatřete vhodným grafickým uživatelským rozhraním.

DOPORUČENÁ LITERATURA:

[1] CHÝLA, R. Detekce plagiátorství. Ikaros [online]. 2009, roč. 13, č. 2. Dostupné z: http://ikaros.cz/node/5253

[2] SI, A., H.V. LEONG a R.W.H. LAU. CHECK: A Document Plagiarism Detection System. In Proceedings of ACM Symposium for Applied Computing. February 1997, s. 70–77.

Termín zadání: 8.2.2016 ***Termín odevzdání:*** 20.5.2016

Vedoucí práce: Ing. Martin Vítek, Ph.D.
Konzultant diplomové práce:

prof. Ing. Ivo Provazník, Ph.D., *předseda oborové rady*

ABSTRAKT

Cílem této práce bylo seznámit se s problémem plagiátorství a navrhnout program, který by byl schopen odhalit plagiátorství v programových kódech. V první části práce jsou teoreticky popsány jednotlivé druhy plagiátorství. Dále pak jsou představeny způsoby, jakým je možno k detekci přistupovat. V praktické části je pak prvně popsána fáze předzpracování a dále je navrhnut nový způsob detekce a vyhodnocení plagiátorství, pomocí adaptivních vah. Na konci práce jsou pak výsledky testování práce na databázi studentských projektů.

KLÍČOVÁ SLOVA

plagiát, plagiátorství, programové kódy, příznaky, detekce, DTW, mapování

ABSTRACT

Main goal of this work is to get acquainted with the plagiarism problem and propose the methods that will lead to detection of plagiarism in program codes. In the first part of this paper different types of plagiarism and some methods of detection are introduced. In the next part the preprocessing and attributes detection is described. Than the new method of detection and adaptive weights usage is proposed. Last part summarizes the results of detector testing on the student projects database

KEY WORDS

plagiarism, programme codes, attributes, detection, DTW, mapping

BIBLIOGRAFICKÁ CITACE

KAŠPAR, J. Vyhodnocení podobnosti programových kódů. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2016. 49 s. Vedoucí semestrální práce Ing. Martin Vítek, Ph.D..

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma *„Vyhodnocení podobnosti programových kódů“* jsem vypracoval samostatně pod vedením vedoucího práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny uvedeny v seznamu literatury na konci práce.

V Brně dne …… ……

Jakub Kašpar

PODĚKOVÁNÍ

Děkuji svému vedoucímu Ing. Martinu Vítkovi Ph.D. za cenné rady při zpracování mé diplomové práce.

V Brně dne …… ……

Jakub Kašpar

OBSAH

Obsah 5

1 Úvod 6

2 Plagiátorství 8

2.1 Definice ... 8

2.2 Druhy plagiátorství ... 9

3 Detekce v programových kódech 13

3.1 Známý software pro detekci plagiátů ... 13

3.2 Způsob detekce ... 16

3.3 Příznaky ... 17

3.4 Problémy s detekcí ... 18

4 Realizovaný detektor 19

4.1 Použitá databáze ... 19

4.2 Předzpracování ... 19

4.3 Detekce příznaků ... 22

4.4 Detekce proměnných ... 24

4.5 Vytvoření map ... 25

4.6 Vyhodnocení plagiátorství ... 29

5 Grafické uživatelské rozhraní 31

6 Vyhodnocení výsledků 38

7 Závěr 44

Literatura 46

Seznam obrázků 48

1 ÚVOD

Jedním z hlavních a velice závažných problémů vyskytujících se na akademické půdě převážně mezi studeny je takzvané plagiátorství. Existují v podstatě dva druhy, kde se může na plagiátorství narazit. Jednou z možností je plagiátorství vědeckých článků, publikací, knih, nápadů či jiných projektů. Druhým místem kde je možnost se s plagiátorstvím setkat, jsou programové kódy. Dále se tyto dva typy mohou rozdělit buďto na plagiátorství úmyslné či neúmyslné až nevědomé.

Tato práce se bude převážně věnovat plagiátorství programových kódů a jeho odhalování. Samotné plagiátorství může u studentů nabývat různých podob. Nejjednodušší a nejčastější případ je ten, že si studenti kopírují kód navzájem mezi sebou. Úplná kopie se však vyskytuje jen zřídka, neboť odhalení bývá velice snadné. V nejčastějším případě dochází ke změně názvů proměnných od originálu, pozměnění formátování kódu a doplnění vlastních komentářů. Program, který by porovnával čistě podobnost dvou kódů, by tyto úpravy dostatečně zmátly tak, aby porovnané práce nebyly vyhodnoceny jako plagiát. Avšak kdyby tyto práce porovnával člověk, bylo by mu zcela jasné, že jde v podstatě o stejný kód. Proto je velice důležité, založit detekci na porovnávání určitých příznaků, které se v programovém kódu objevují.

Druhou možností, jak může docházet k plagiátorství, je kopírování z externích zdrojů. Principiálně je detekce zcela totožná, problém však je v nalezení původního zdroje. V případě, že došlo ke zkopírování kusu kódu z internetových stránek, dá se tento zdroj ještě s větší či menší námahou dohledat. Úspěšnost však závisí na úsilí, jež student do vyhledávání cizího zdroje vložil. Použije-li kód z prvního odkazu s danou problematikou, není problém tento zdroj najít. Pokud se mu však povedlo najít kód z nějaké neznámé stránky či blogu, nemusí ji hodnotící vůbec najít a celá práce muže být tedy chybně klasifikována jako původní.

Kromě samotného plagiátorství již vytvořených kódů, existují i různé internetové stránky a vůbec služby, kde může student za poplatek nebo i zadarmo získat výsledek celé své práce od někoho s většími zkušenostmi. Ačkoliv se jedná o originální práci, která není nikde jinde zveřejněna, z pozice studenta se též jedná o plagiát. Ten je však v tomhle případě jen těžko odhalitelný.

Cílem této práce bude seznámení s obecnou teorií a definicí plagiátorství. Poté představení již dostupného softwaru pro detekci plagiátů a jejich případné porovnání. Hlavním bodem celé práce je vytvoření vlastního softwaru pro detekci plagiátů ve dvou porovnávaných programových kódech. Hlavní důraz je přitom kladen na širokou škálu příznaků, podle kterých je míra plagiátorství v porovnávaných kódech vyhodnocována, a na co nejkomplexnější nahrazení takzvaného lidského faktoru, který dokáže kvalitu

výsledků detekce značně zvýšit. Důraz je taky kladen na celkovou automatizaci detekčního softwaru. V případě porovnání pouze malého počtu prací by to nebylo nutné, ale pro jakékoliv praktické využití je automatizace absolutně nezbytnou součástí. [1]

2 PLAGIÁTORSTVÍ

2.1 Definice

Slovo plagiátorství pochází původem z latinského slova *plagiarius* a jeho význam byl *únosce*. V dnešní době je však již toto slovo spojeno spíše s pojmem *napodobenina*. Jednou z oficiálních a více popisujících definicí by mohla být: „Plagiátorství je činnost, při které jsou výsledky práce nebo nápady jiné osoby vydávány za vlastní.“ [2]

Podle DNKČR, neboli Databáze Národní knihovny České Republiky, je plagiátorství „vydávání cizího literárního nebo jiného uměleckého nebo vědeckého díla za vlastní, popřípadě převzetí části cizí práce, bez uvedení použitých zdrojů.“ [5]

Další možná definice je například podle Ústřední knihovny ČVUT, kde se píše: „Za plagiátorství se považuje nejen úmyslné okopírování (ukradení) cizího textu a jeho vydávání za vlastní, ale i nedbalé citování, neúmyslné opomenutí citace některého využitého zdroje a myšlenky či nedostatečná práce s původním textem (nedostatečná parafráze, kompilace původního textu. Plagiátorství se dopouští nejen ten, kdo takovýmto způsobem neoprávněně cizí text využije, ale i osoba, která poskytuje služby, jež plagiátorství přímo umožňují, popřípadě k němu nabádají. Tedy strana, která tyto texty zdarma či za úplatu produkuje a/nebo poskytuje.“ [4]

Důležitý je i samotný důvod, proč k plagiátorství došlo. Buď se může jednat o plagiátorství úmyslné, kdy důvodem mohl být třeba nedostatek času k vypracování práce v daném termínu, finanční krize, nebo neschopnost přijít s něčím vlastním. Patří sem také možnost zakoupení nebo získání cizí práce a vydávání ji za vlastní. Druhým případem je plagiátorství neúmyslné, kdy je příčinou nesprávně nebo nedostatečně citovaná zdrojová literatura. Poslední možností je takzvané plagiátorství nevědomé. Jedná se o případ, kdy dotyčný v minulosti někde četl nějaký článek či nápad, podvědomě si ho zapamatuje a dále ho v budoucnu někde využije jako svůj, či na něm postaví část své práce. I když zde není úmysl o poškození, stále se jedná o plagiátorství, jelikož jsou upírány nároky na nápad původnímu autoru.[2]

2.2 Druhy plagiátorství

Zde se seznámíme s jednotlivými druhy plagiátorství a důvody, proč se o plagiátorství jedná, případně i způsob, jak se jim vyhnout.

1) **Opomenutí sekundárního zdroje:**

- Jedná se o problém, kdy autor využívá informace obsažené v primárním článku, který se však odkazuje na další (sekundární) zdroj, a cituje pouze článek primární. Tento druh plagiátorství naprosto opomíjí zásluhy autorů sekundárního článku a udává falešnou představu o tom, kolik úsilí bylo do výzkumu vloženo. Pro předejití tohoto druhu plagiátorství je tedy nutné dbát zvýšené pozornosti na to, jestli je citovaná část opravdu pouze části primárního článku. [6]

2) **Chybný nebo neexistující zdroj:**

- Tento problém se vyskytuje v případech, že autor chybně uvede svůj zdroj, nebo si případně vymyslí nějaký neexistující. Důvodem tohoto typu plagiátorství může být buď pouhá nedbalost při práci, ale může se jednat i o pokus uměle zvýšit počet zdrojů a zakrýt tak vlastní nedostatečný výzkum. Také se může jednat o uvedení nepoužitého, ale obecně lépe hodnoceného zdroje, namísto použitého, o kterém se autor domnívá, že není dostatečně validní. Tato výměna je porušením etického kodexu. [6]

3) **Duplikát:**

- O duplikát se jedná i v případě, kdy autor použije znovu část práce z jeho vlastních předchozích výzkumů a článků bez použití citace. Etická stránka tohoto typu duplikátu je sporná a většinou závisí na samotné duplikované části. Tento typ plagiátorství bývá většinou tolerován, a pokud se přijde na to, že k němu došlo, bývá autor článku požádán, aby buďto přepsal duplikovanou část, nebo aby přidal citaci své předešlé práce.[6]

4) **Parafrázování:**

- Parafrázování lze definovat jako převzetí práce někoho jiného a pouhé změnění slov natolik, aby to vypadalo, že daný nápad, nebo i práce je originální, i když ve skutečnosti pochází z necitovaného zdroje. Parafrázování může být v rozsahu od jedné věty po přepsání celé práce při zachování původního nápadu. Je tedy nutné i při pouze krátkém výňatku parafrázovaném z jiné práce citovat daný zdroj. [6]

5) **Opakovaný výzkum:**

- Jedná se o problém plagiátorství, kdy se opakují data nebo text z podobné studie s podobným postupem do nové studie bez řádné citace. Většinou k tomu dochází u studií s podobným tématem, kdy je daná metoda znovu zkoumána či použita na jiných datech a dojde se k podobným, ne-li stejným výsledkům. Bývá normální, že se využívá metod, které dobře fungují, stejně tak že se výsledky těchto metod popisují velice podobně. Může být tedy obtížné prezentovat tyto data ve zcela nové formě. Pokud tomu tak není, je třeba důkladně celou metodu citovat, aby nedošlo k porušení autorských práv. [6]

6) **Replikace:**

- Replikace je případ, kdy je autorův jeden konkrétní článek zároveň uvedený ve více jak jedné publikaci. To může být etický problém zvláště v případě, tvrdí-li autor, že článek je nový, i když už byl publikován někde jinde. Většina odborných časopisů požaduje jedinečnost publikovaných článků a mohlo by zde docházet k problémům s copyrightem. Neúmyslná replikace může nastat například v případě, kdy se autor rozhodne rozeslat svoji práci do více časopisů s úmyslem zvýšit svoji šanci na publikování. V takové situaci je však nutné ihned po přijmutí článku do jedné z publikací informovat o tomto faktu všechny ostatní původně oslovené vydavatele. [6]

7) **Nepřesné uvedení autorů:**

- Jedná se o nepřesné, nebo neúplné uvedení seznamu autorů, kteří se na daném článku podíleli. Prvním případem je situace, kdy jsou autorovi odepřeny zásluhy na práci, na které se částečně, nebo i významně podílel. Opačným případem je uvedení autorů, kteří však nemají s daným článkem vůbec nic společného. Například student, který se podílí se na práci svého profesora a není uveden jako spoluautor, může nahlásit tento druh plagiátorství. Jedná se však spíše o etický problém a většinou bývá vydavatelem přehlížen. [6]

8) **Neetická spolupráce:**

- Jde o případ, kdy člověk podílející se na týmové práci poruší etiku a vydává nápady a výsledky, které jsou výstupem jejich společné práce ve vlastním článku, bez uvedení citace, či zmínění ostatních autorů, se kterými se na výzkumu podílel. Společná práce může znamenat spoustu etických problémů při individuálním zpracování, například u přidělování podílu odvedené práce jednotlivým lidem, používání nápadů ostatních, nedostatečné ocenění a jiné. [6]

9) **Doslovné plagiátorství:**

- Jedná se o případ, kdy je z jiné práce zcela zkopírovaná její část bez řádného uvedení citace. Může nabývat dvou forem. V prvním případě může autor uvést, že se jedná o myšlenky z jiného zdroje, ale neprozradí již, že se jedná o doslovnou citaci. V druhém případě není uvedena žádná citace a autor používá cizí práci za vlastní. Jde o jeden z nejvážnějších ale zároveň nejsnadněji detekovatelných případů plagiátorství. [6]

10) Kompletní plagiát:

- Extrémní případ plagiátorství, kdy autor vezme práci někoho jiného a beze změny ji vydává za svou. Je to nejtěžší případ plagiátorství, kdy je původní autor doslova okraden o jakékoliv zásluhy na své práci. [6]

11) Autoplagiátorství:

- Jde o speciální typ plagiátorství, kdy autor používá své vlastní, již dříve použité práce, bez řádné citace. Nejedná se sice o porušení autorských práv, ale stejně jako u využití práce jiných lidí, je autor povinen použít citace. [2]

3 DETEKCE V PROGRAMOVÝCH KÓDECH

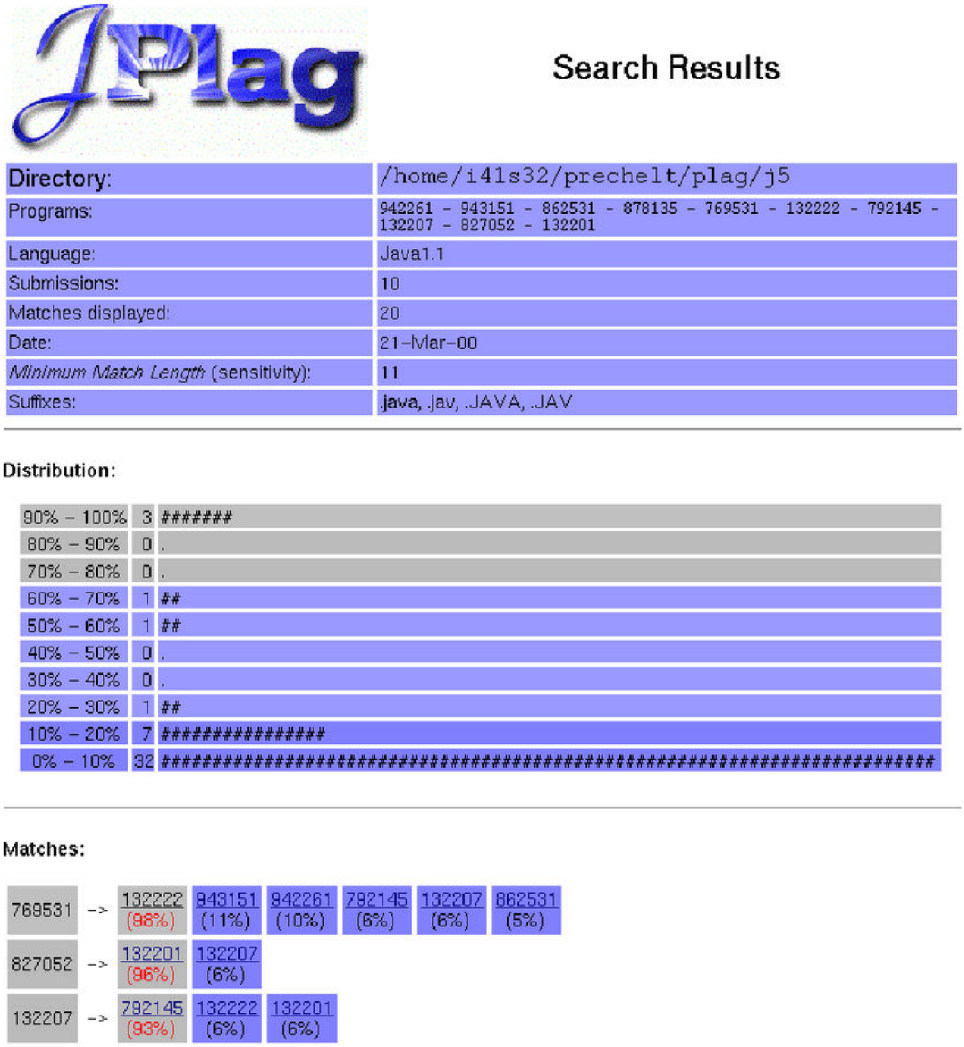
Doposud bylo rozebíráno plagiátorství obecně. V této a dalších kapitolách bude hlavním cílem představit plagiátorství v programových kódech a způsoby, kterým se může detekovat.

3.1 Známý software pro detekci plagiátů

V dnešní době existuje na trhu celá řada dostupných softwarů pro detekci plagiátorství a to jak pro vědecké články, tak pro programové kódy. V této části práce budou představeny některé z nich.

1) JPlag:

- Program vytvořený pro porovnávání studentských prací. Nachází podobnosti mezi dvěma nebo i více soubory najednou a tím je schopen odhalit plagiáty. Celý program je založen na znalosti syntaxe programovacích jazyků. To umožňuje odhalit i složitější pokusy a maskování, že jde o plagiát. JPlag momentálně podporuje programovací jazyky *Java, C#, C, C++ a Scheme* a dokáže pracovat i s přirozeným textem. Krom srovnání celých prací, dokáže také identifikovat malé části programu v rozsáhlém (a třeba i z velké části původním) kódu, které byly převzaty odjinud. Celý program je zdarma. [1, 9] Příklad výstupu z programu JPlag je ukázán na *Obrázek 1*.



JPlag

Search Results

Directory:	/home/i41s32/prechelt/plag/j5
Programs:	942261 - 943151 - 862531 - 878135 - 769531 - 132222 - 792145 - 132207 - 827052 - 132201
Language:	Java1.1
Submissions:	10
Matches displayed:	20
Date:	21-Mar-00
Minimum Match Length (sensitivity):	11
Suffixes:	.java, .jav, .JAVA, .JAV

Distribution:

90% – 100%	3	#######
80% – 90%	0	.
70% – 80%	0	.
60% – 70%	1	##
50% – 60%	1	##
40% – 50%	0	.
30% – 40%	0	.
20% – 30%	1	##
10% – 20%	7	################
0% – 10%	32	##

Matches:

769531	->	132222 (98%)	943151 (11%)	942261 (10%)	792145 (6%)	132207 (6%)	862531 (5%)
827052	->	132201 (96%)	132207 (6%)				
132207	->	792145 (93%)	132222 (6%)	132201 (6%)			

Obrázek 1: Výstup z programu JPlag [13]

2) Moss:

- Moss neboli *Measure of Software Similarity*je volně dostupný program pro automatickou kontrolu podobnosti v programových kódech. Byl vytvořen v roce 1994 a od té doby se stále vyvíjí. Momentálně dokáže analyzovat kódy v jazycích *C, C++, C#, Java, Python, Visual Basic, Javascript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, ADA, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly a HCL2.*[1, 10] Příklad toho jak vypadá výstup programu Moss je na *Obrázek 2*.

Moss Results

Sun Mar 14 15:24:02 PST 1999

Options -l c -m 10

[Text Report | How to Read the Results | Tips | FAQ | Contact Moss | Submission Scripts | Credits]

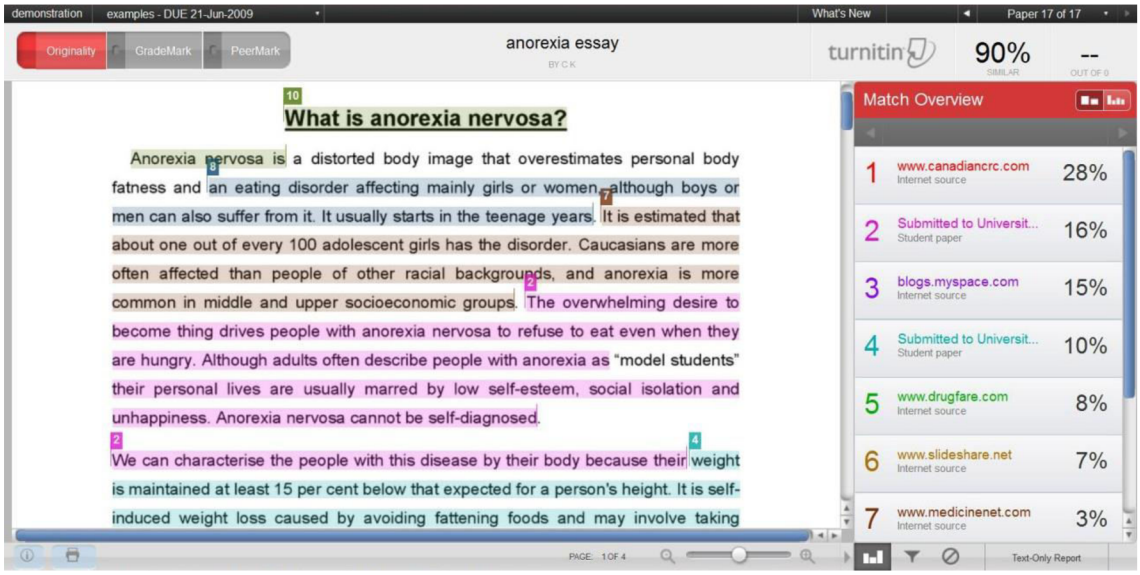
File 1	File 2	Tokens Matched	Lines Matched
mike_wolf.c (79%)	mike_fox.c (80%)	463	139
bill_smyth.c (86%)	bill_smith.c (88%)	456	133
jane_white.c (59%)	jane_blanco.c (68%)	354	111
john_doe.c (100%)	john_deer.c (100%)	220	49

Any errors encountered during this query are listed below.

Obrázek 2: Výstup z programu Moss [14]

3) Turnitin:

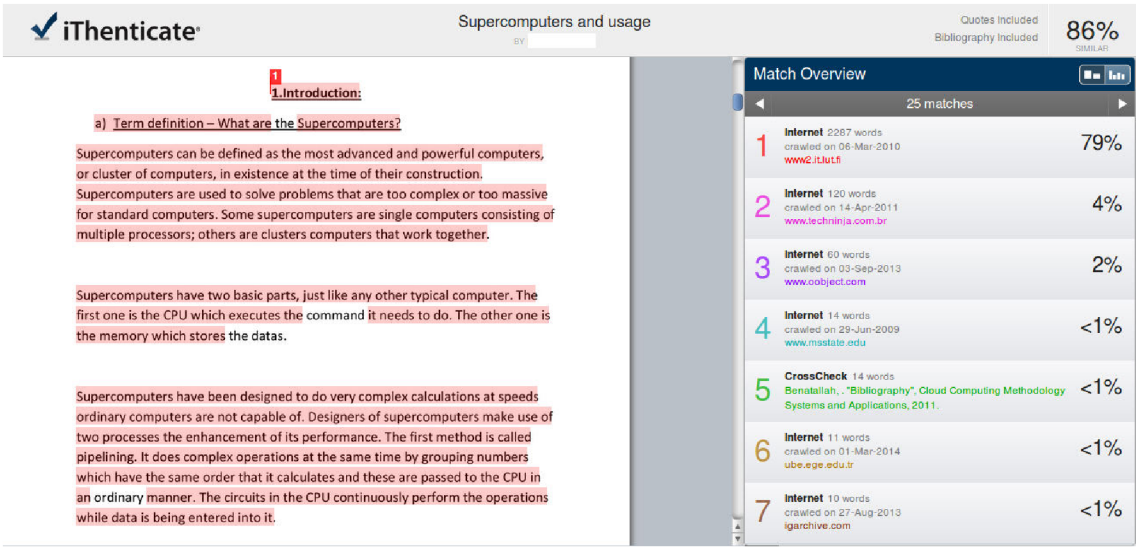
- Jedná se o placený program pro školy, který profesorovi umožňuje efektivně kontrolovat originalitu studentských prací. Kromě zpětné interakce se studenty a online známkování, má tento program velikou výhodu v tom, že dokáže nejen porovnávat práce odevzdané studenty, ale dokáže nacházet i další převzaté části na internetu. Výsledkem je pak procentuální zobrazení převzatých částí práce a to i s uvedením původních zdrojů.[8] Výstup z programu Turnitin je na *Obrázek 3*.



Obrázek 3: Výstup z programu Turnitin [15]

4) iThenticate:

- Stejně jako Turnitin, iThenticate je placený program pro kontrolu plagiátorství. Na rozdíl od něj je však zaměřen spíše na firmy a vědecké prostředí. V krátkém čase dokáže porovnat texty a navíc je kontroluje i s obsahem umístěným na internetu. Program je určen pouze pro texty. [7] Příklad výstupu z programu iThenticate je na *Obrázek 4*.



Obrázek 4: Výstup z programu iThenticate [7]

3.2 Způsob detekce

K detekci plagiátorství v programových kódech se dá přistupovat různými způsoby. Asi nejjednodušším způsobem je porovnávání dvou kódů slovo od slova, řádek po řádku. Tento způsob má ale hned několik velkých nevýhod. Jednou z nich je určitě časová náročnost. Jako u všech technik úplného prohledávání, i zde by detekce touto metodou zabrala výrazně víc času než jiné příznakové metody.

Celkově hlavní nevýhodou tohoto způsobu detekce je velice snadný způsob, jak ho obejít. Pokud by se porovnával program jako celek, stačí udělat v kopii pár drobných změn a detektor plagiát neodhalí. I v případě porovnávání shodnosti jednotlivých řádků nemůže ve většině případů docházet k příliš dobrým výsledkům. Ten, kdo plagiát vytváří, často změní názvy proměnných a popřehazuje řádky kódu, u kterých nezáleží

na pořadí. Tím je schopen tento způsob detekce spolehlivě překonat.

Z tohoto důvodu existují detekční metody založené na porovnávání jednotlivých příznaků, které jsou schopny v programovém kódu přetrvat i po jeho upravení. Ty jsou pak dále porovnávány mezi sebou a jejich podobnost pak určuje mírů podezření na plagiát. Každý příznak má taky v celkovém určení plagiátu jinou váhu. [3]

3.3 Příznaky

Jak již bylo řečeno, příznak je určitý parametr programového kódu, který by měl být pro originál i jeho plagiát stejný, nebo při nejmenším podobný. Mohou to být buď fyzické vlastnosti kódu, jako jsou například:

- velikost souboru
- počet slov
- počet řádků
- počet znaků

Pokud se tyto vlastnosti získávají na předzpracovaném kódu, je malá pravděpodobnost, že by mezi sebou porovnávané soubory příliš lišily. Předzpracování je ovšem klíčové, protože bez odstranění komentářů, volných řádků a různých druhů formátování, by v hodnotách byly značné rozdíly. Právě změna a natahování komentářů k programovému kódu bývá častý způsob snahy o zamaskování plagiátorství.

Druhým typem příznaků jsou klíčová slova. Je to seznam hledaných slov v kódu, které jsou pro daný programovací jazyk typické. Porovná se zejména počet těchto slov. Klíčová slova mají výhodu v tom, že i když je plagiát částečně změněn, například v názvech proměnných nebo popřehazováním řádků, hlavní funkce většinou zůstávají stejné, z důvodu správné funkce programu. Čím větší počet různých klíčových slov se hledá, tím je větší schopnost přesněji určit podobnost mezi kódy. Nevýhodou klíčových slov je fakt, že i když mají dva programy stejný počet klíčových slov, nemusí se jednat o plagiát. Pokud by se vzaly v úvahu dva jednoduché kódy, s jedním *for* cyklem a jednou *if* podmínkou, oba programy mohou dělat úplně něco jiného, ale přesto budou vyhodnoceny jako plagiát. [3]

Z toho důvodu je při detekci nutné používat oba typy příznaků. Problémem ale zůstává, že i při kombinaci obou druhů příznaků, popřípadě i s přidáním metody úplného prohledávání, nejsou výsledky v tom, co je či není plagiát, zcela dostačující. Bude tedy nutné vymyslet další typy příznaků, které celý proces detekce zpřesní. Příkladem by mohlo být porovnávání pozice klíčových znaků v obou kódech s nějakou přípustnou směrodatnou odchylkou, či sledování délky jednotlivých řádků v celém kódu nebo aspoň řádků s klíčovými slovy.

3.4 Problémy s detekcí

Hlavním problémem při detekování plagiátu pomocí automatického programu, je program samotný. Program totiž neobsahuje velice důležitý, takzvaný lidský faktor. Porovnává-li dva programy člověk, schopnost odhalení plagiátu je ve většině případů mnohem vyšší než u programu. Úspěšnost samozřejmě závisí na úsilí vynaloženém plagiát skrýt, ale lidský mozek si velice snadno dokáže všimnout přeházených řádků, přejmenovaných proměnných, celkových změn ve formátování, případně i nahrazení některých funkcí jinými. Například programové zjištění toho, jestli je v jednom programu použita funkce *sum* a v druhém je v důsledku maskování nahrazena za obyčejné sčítání a pro úplnou jistotu ještě leží na jiném řádku, by bylo v rámci porovnávání textu velice obtížné, ne-li nemožné.

Ideálním řešením tohoto problému by bylo nějakým způsobem zahrnout porovnávání reálných výsledků a jednotlivých kroků programu. Tím se však celý způsob detekce a vůbec řešení problému dostává úplně do jiné dimenze.

Způsob jakým zůstat u metody porovnávání dvou textů, ale zároveň řešit funkčnost operací a ne jejich formu, by mohla vyřešit konverze porovnávaných kódů do podoby, v jaké ji získává a zpracovává procesor. Jednalo by se příkazy shromážděné v takzvaném assembleru, neboli fáze před konverzí do strojového kódu, obsahujícího pouze nuly a jedničky. Problém s tímto řešením je v tom, že samotný MATLAB, ve kterém je celý praktická část vypracována, není schopen příkazy v assembleru zobrazit. Bylo by tedy nutné použít externích programů, což je pro jednoduchost obsluhy a praktické použití nežádoucí.

4 REALIZOVANÝ DETEKTOR

4.1 Použitá databáze

Pro testování a obecně práci s detektorem, popřípadě jeho částmi, byly použity projekty studentů oboru BTBIO z předmětu APRG za posledních sedm let. Celá databáze tedy čítá 553 různých projektů. Jedná se o zdrojové kódy z programového prostředí MATLAB. Projekty jsou vytvářeny v různých verzích programu a individuálně formátovány. Celá databáze je rozdělená jednak podle roku vypracování a dále pak podle tématu zadání. Z důvodu ochrany autorských práv a prevence z případného nařčení z veřejného obvinění z plagiátorství, jsou všechny ukázky projektů v této práci anonymizovány.

4.2 Předzpracování

Pro správnou detekci plagiátu je velice důležitá fáze předzpracování. Bez ní by bylo z důvodu možnosti různého formátování nadmíru obtížné pomocí programu rozpoznat, že se jedná o dva totožné kódy. Příklad dvou totožných kódu s různým formátováním je na *Obrázek 5* a *Obrázek 6*.

```
1 -    clear all; %vymazání všech proměnných
2 -    close all; %zavření všech otevřených oken
3 -     v = [1 2 3 4 5]; %proměnná pro nalezení pozice minima
4 -     MiNim = []; %pomocná proměnná
5 -    for i = 1:3 %začátek for cyklu
6 -     [MiNim(i), Pozice] = min(v); %nalazení minima a jeho pozice
7 -     v(Pozice) = []; %ulození pozice aktuálního minima
8 -    end %konec cyklu
9
```

Obrázek 5: Příklad zdrojového kódu pro vyhledávání pozice minima s formátováním '1'

```
1  -    clear all;
2  -    close all;
3
4  -     v=[1 2 3 4 5]; %data pro zpracování
5
6  -     MiNim = []; %proměnná MiNim
7
8       % Začátek vyhledávání pozice
9  -    for i=1:3
10
11          %hledání pozice
12 -        [MiNim(i),Pozice]=min(v);
13
14          %uložení pozice
15 -        v(Pozice)=[];
16
17      %konec
18 -    end
19
```

Obrázek 6: Příklad zdrojového kódu pro vyhledávání pozice minima s formátováním '2'

Oba výše zobrazené zdrojové kódy jsou funkčně naprosto totožné. Na první pohled je však vidět rozdíl v počtu řádku a celkovém formátování kódu. Navíc v obou případech se v kódu vyskytuje mnoho pro funkčnost nepotřebných věcí. Detektor založený na porovnání čisté podobnosti by tyto dva kódy označil jako rozdílné. I v případě porovnávání klíčových znaků by nemuselo dojít k přesným výsledkům. Z tohoto důvodu je klíčové, dostat všechny porovnávané kódy do stejné podoby. Toho se docílí provedením následujících operací:

- odstranění všech mezer mezi slovy
- odstranění veškerého odsazení před prvním slovem a za posledním slovem na řádku
- odstranění všech prázdných řádků
- odstranění všech komentářů
- změna všech velkých písmen na malá

Tyto operace pochopitelně celý kód znehodnocují, ale to samotné detekci nijak nevadí, naopak je pro to pro ni nezbytné.

Výsledkem předzpracování jsou tedy pouze řádky s kódem bez mezer. Samotný program pak má dva výstupy, kdy jedním je matice Xx1, kde X je počet užitečných řádků programu. Každému řádku je tedy přiřazen jeden řádek v matici. Tento formát je vhodnější pro jednodušší vyhledávání klíčových znaků a obecné porovnávání jednotlivých řádků. Druhým výstupem je pak vektor, ve kterém je celý program zapsán na jednom řádku. Tento výstup je vhodnější pro porovnávání velikosti a celkového počtu znaků. V obou případech se ovšem jedná jen o zjednodušení pro zjišťování daných znaků. Rozdíl mezi původním kódem a kódem po předzpracování je vidět na *Obrázek 7* a *Obrázek 8*.

```
clear all
list = dir('K:\Vyška\Ing\Diplomka\Moje');
i = 1;
data = struct();
while i <= length(list)
    if isempty(strfind(list(i).name,'.m')) == 1
        list(i) = [];
    else
        i = i+1;
    end
end

for k = 1:length(list)
t = fileread(list(k).name);
name = list(k).name;
t(t==char(10)) = [];  %zbavi se umele udelanych radku ve skriptu
pozice = find(t==char(13));   %najde vsecky rucne vytvorene radky
t = lower(t);  %vsecko na maly pismena (mozna muze kazit dalsi zpracovani coz Case sensitive)
radky = {};
for i = 1:length(pozice)+1
    if isempty(radky) == 1
        radky{1} = t(1:pozice(1));  %napise prvni radek
        pom = strfind(radky{end},'%');
        if isempty(pom) ~= 1
```

Obrázek 7: Příklad zdrojového kódu bez předzpracování

	1
1	clearall
2	list=dir('k:\vyška\ing\diplomka\moje');
3	i=1;
4	data=struct();
5	whilei<=length(list)
6	ifisempty(strfind(list(i).name,'.m'))==1
7	list(i)=[];
8	else
9	i=i+1;
10	end
11	end
12	fork=1:length(list)
13	t=fileread(list(k).name);
14	name=list(k).name;
15	t(t==char(10))=[];
16	pozice=find(t==char(13));
17	t=lower(t);
18	radky={};
19	fori=1:length(pozice)+1
20	ifisempty(radky)==1
21	radky{1}=t(1:pozice(1));
22	pom=strfind(radky{end},'
23	ifisempty(pom)~=1

Obrázek 8: Příklad zdrojového kódu po předzpracování a vložení do matice podle řádků

Celý program pro předzpracování je pak ještě doplněn o počátky automatizace. Na začátku programu je vytvořen seznam všech *mfile* souborů obsažených v pracovní složce. Poté každý jednotlivý soubor prochází předzpracováním a zároveň jsou u něj zjišťovány příznaky pro následnou detekci.

4.3 Detekce příznaků

Celý detektor by měl pracovat na bázi porovnávání klíčových znaků z každého zdrojového kódu. Jejich podobnost posléze určí, jedná-li se u porovnávaných souborů o plagiát či nikoliv. Použité příznaky jsou následující:

- velikost programového kódu v bytech
- počet řádků kódu
- počet znaků kódu
- počet klíčových slov

V případě prvních tří příznaků není způsob detekce nijak složitý. Pro jejich zjištění existují jednoduché příkazy. Problém může nastat u některých klíčových slov. Seznam všech klíčových slov které byly použity je v *Tabulka 1*.

Tabulka 1: Použitá klíčová slova

Seznam klíčových slov				
function	plot	size	fft	cell2mat
if	stem	length	fft2	mat2cell
elseif	xlabel	ones	ifft	mat2gray
else	ylabel	zeros	ifft2	rgb2gray
for	title	sum	fftshift	find
while	imshow	diff	ifftshift	strfind
switch	pause	mean	num2str	strcmp
case	disp	mod	str2num	isempty
figure	end	abs	str2double	fastaread

Tato klíčová slova by se dala rozdělit do několika kategorií, v závislosti na tom, jak jsou detekována. U většiny z nich je totiž problém, že nemohou být vyhledávána pouhým příkazem z celého programového kódu.

První skupinou jsou slova jako *for*, *if* a *while*. U těchto funkcí se při vyhledávání musí kontrolovat pozice na řádku. Pokud je některé z těchto slov detekováno, musí být zároveň potvrzeno, že leží na samém začátku řádku. Na jiném místě se totiž v programu tyto funkce vyskytovat nemohou. Dále je kontrolován fakt, jestli je za detekovaným slovem závorka. V takovém případě není slovo zaznamenané pro další detekci.

Druhou skupinou jsou slova jako *figure*, která mají volitelnou možnost použití závorky. Je proto nutné kontrolovat oba tyto stavy.

Samostatnou skupinou je slovo *end*. U této funkce je kontrolováno, zda-li je na řádku úplně sama. V opačném případě se jedná pravděpodobné o část jiného slova.

Poslední skupinou jsou slova jako *fft*, nebo *fftshift*. Názvy těchto funkcí jsou totiž součástí jejich reverzních funkcí, kterými jsou v tomto případě *ifft* a *ifftshift*. Proto je u těchto funkcí nutná kontrola, jestli se nejedná funkci inverzní. S tímto případem souvisí i kontrola, která se provádí u ostatních klíčových slov. Zjišťuje se, jestli nejsou součástí například nějakého názvu proměnné, tedy jestli se opravdu jedná o používanou funkci. Příkladem může být název proměnné *forcount*, jejíž název je z části totožný s funkcí *for*. [11]

Všechny takto nedetekované klíčové znaky jsou ukládány do buňkového pole, ze kterého je část uvedena na *Obrázek 9*.

	1	2	3
1	'Název'	'Počet'	'Pozice'
2	'function'	1	1
3	'if'	2	[4 19]
4	'elseif'	0	[]
5	'else'	0	[]
6	'for'	1	9
7	'while'	0	[]
8	'switch'	0	[]
9	'case'	0	[]
10	'figure'	0	[]
11	'plot'	1	18
12	'stem'	0	[]
13	'xlabel'	1	22
14	'ylabel'	1	23
15	'title'	0	[]
16	'imshow'	0	[]
17	'pause'	0	[]
18	'disp'	2	[5 25]
19	'end'	3	[7 11 21]
20	'size'	0	[]
21	'length'	3	[9 13 14]
22	'ones'	0	[]
23	'zeros'	0	[]
24	'sum'	1	13
25	'diff'	0	[]
26	'mean'	0	[]
27	'mod'	0	[]

Obrázek 9: Příklad detekovaných klíčových znaků

Jak je možné vidět, kromě množství jednotlivých detekovaných znaků, jsou detekovány i pozice řádků, na kterých se detekovaná slova vyskytují. Těchto pozic je využito jako prostředku k vytvoření vah pro jednotlivé znaky, což bude popsáno dále v práci.

4.4 Detekce proměnných

Zvláštní skupinou klíčových slov jsou proměnné, které byly použity ve zpracovávaném kódu. Zaznamenávat nebo porovnávat názvy použitých proměnných je zbytečné, protože první věcí, kterou každý plagiátor udělá, je přepsání názvů všech proměnných v kódu. Z toho důvodu byl vytvořen algoritmus, který hledá pouze místa, kde byly proměnné použity.

Dále je důležité definovat, co se přesně myslí proměnnou. V celém kódu by se daly proměnné zjednodušeně rozdělit na dva typy. Na proměnné nalevo od rovnítka a na proměnné napravo od rovnítka. Pro účely detektoru byly zjišťovány pouze proměnné nalevo od rovnítka. Jako u ostatních klíčových slov se zaznamenává jejich počet a pozice. Jako proměnná je označen každý řádek, na kterém není žádný *for* nebo *while* cyklus, *if*, *switch*, funkce pro zobrazení obrázků nebo grafů jako je *figure*, *imshow* nebo *plot*, nebo kde se nevyskytuje *end*. Tímto způsobem dochází i k tomu, že pokud je v kódu nějaká funkce, vyskytující se na začátku řádku a která není v seznamu detekovaných slov, tak je stále aspoň zaznamenaná jako proměnná. I když se jedná o chybnou detekci, dojde ke stejné chybě u všech zpracovávaných kódů a odhalení skutečných plagiátů to neovlivní.

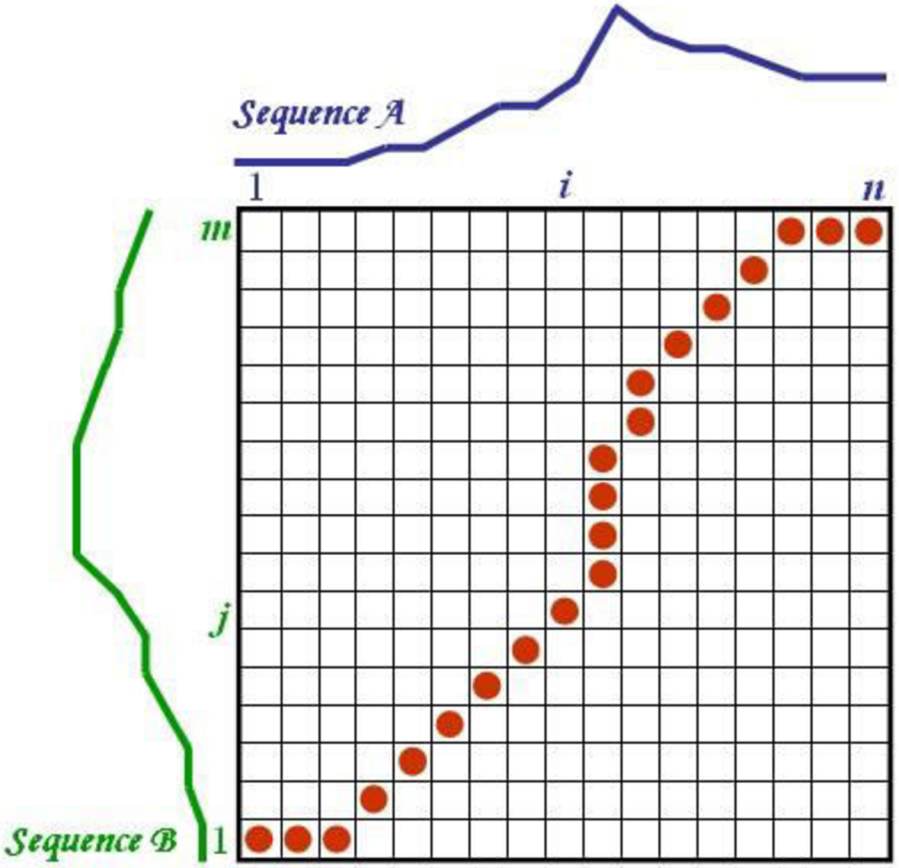
4.5 Vytvoření map

Jak již bylo řečeno dříve, kromě množství jednotlivých znaků, jsou také zaznamenávány jejich jednotlivé pozice. Vytváří se tím ke každému typu klíčového slova vektor, kde každé jeho číslo znamená řádek, kde se znak nacházel. Délka vektoru by tedy měla být stejně dlouhá, jako je počet detekovaných znaků.

Tyto vektory vytváří takzvané mapy. Mapa pro každý typ znaku u jednoho kódu je porovnána s mapou pro druhý zpracovávaný kód. Dochází tak ke zjištění podobnosti těchto map. Toto je velmi účinné například pro detekované proměnné. I když plagiátor změní názvy, nebo pořadí proměnných mezi sebou, pořád bude detekovaná stejná mapa a dojde k odhalení plagiátu.

Porovnávání jednotlivých map je prováděno pomocí výpočtu korelačního koeficientu. Ne vždy však jsou mapy stejně dlouhé, což by znamenalo nemožnost použití této metody. Z toho důvodu je před výpočtem korelačního koeficientu použita na obě mapy metoda dynamického borcení časové osy (DTW). Tato metoda se běžně používá pro synchronizaci měřených signálů se zachováním vnitřního pořadí měřených bodů. Zvláště je pak vhodná pro synchronizaci neúplných sekvencí s chybějícími úseky.

Metoda DTW spočívá v porovnávání vzdáleností jednotlivých úseků obou zpracovávaných signálů. Vytvoří se matice mxn kde m je délka prvního signálu a n délka druhého signálu. Nalevo od matice se přiloží signál délky m směřující od spodu nahoru a nad matici se přiloží signál n směřující z leva doprava. Pro názornost je vše zobrazeno na *Obrázek 10*. [12]

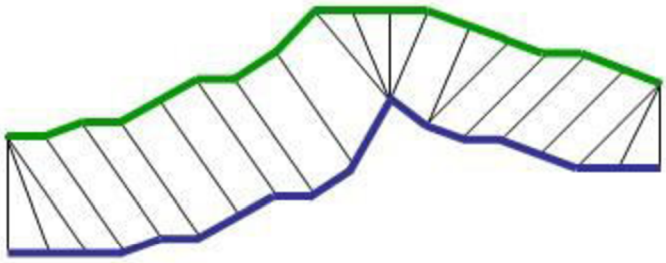


Obrázek 10: Výpočetní matice pro metodu DTW [12]

Po přiložení obou signálů se vypočítají jednotlivé vzdálenosti obou signálů pro všechny prvky matice. K tomu aby se našlo správné zarovnání, je po výpočtu vzdáleností najít v matici cestu, která minimalizuje vzdálenost mezi oběma signály. Toho je dosaženo získáním všech možných cest skrze matici a z těch se vybere cesta s nejmenší sumou všech svých prvků. Celková cesta může být ještě násobena váhovací funkcí. Kromě toho se DTW řídí následujícími podmínkami. [12]

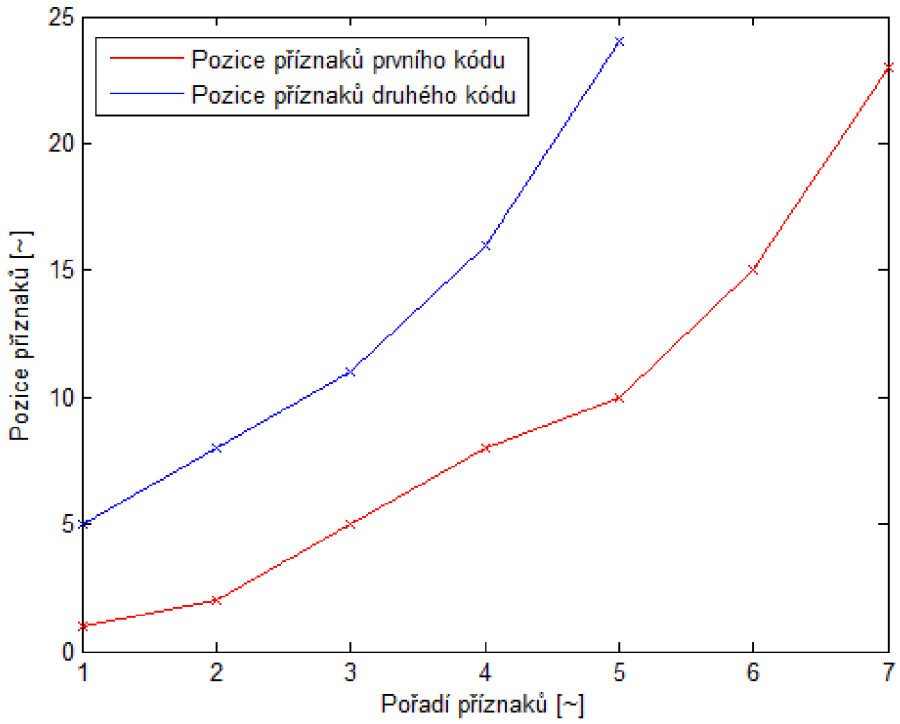
- vytvářená cesta skrze matici se nesmí vracet zpět
- cesta se muže vždy posunout jen o jeden krok, nesmí přeskakovat
- cesta začíná v levém dolním rohu a končí v pravém horním rohu
- výsledná cesta by se neměla odchylovat daleko od diagonály
- výsledná cesta by neměla být moc strmá ani moc plochá

Výsledné zarovnání signálů z *Obrázek 10* je na *Obrázek 11*.

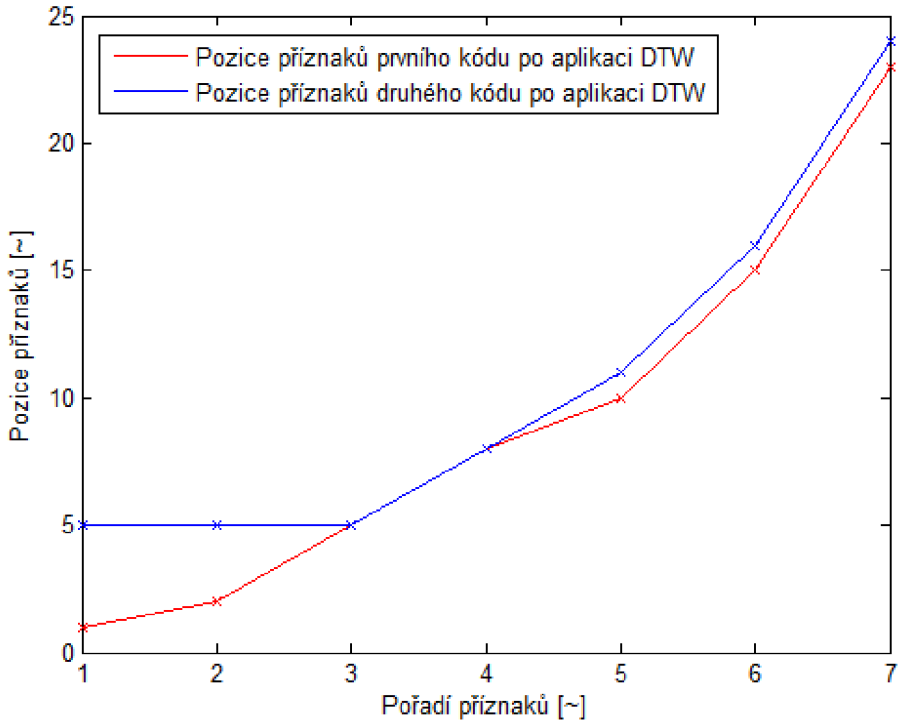


Obrázek 11: Výsledné zarovnání dvou signálů pomocí DTW [12]

V případě použití DTW na dvě nestejně dlouhé mapy dojde k tomu, že kratší mapa se co nejpřesněji přiloží na místo, kde jsou si mapy nejvíce podobné a zbytek se doplní. To je výhodou v případech, že plagiátor smazal například malé množství funkcí, nebo více řádků srazil do jednoho. Zbytek mapy stále zůstává nezměněn a podobnost tedy bude vysoká. Příklad použití DTW na poziční mapy je na *Obrázek 12* a *Obrázek 13*.



Obrázek 12: Mapa dvou příznaků před použitím DTW



Obrázek 13: Mapa dvou příznaků po použití DTW

Tato metoda je užitečná nejen v případě odstraněných či přidaných klíčových slov, ale i tehdy, pokud plagiátor změní pořadí jednotlivých funkcí. I když už na první pohled vypadá kód jinak a mapy při přímém porovnání by byly rozdílné, DTW je opět dokáže sobě velice přiblížit. Příklad plagiátů, které je možno odhalit pomoci map porovnaných metodou DTW je na *Obrázek 14* a na *Obrázek 15*.

```
1    n = 10;
2    x=a:0.1:b; % hodnoty x v rozsahu a-b s krokem 0.1
3    y=eval(fce); % hodnoty y zadané fce
4    plot(x,y,'-r'); % zobrazení grafu zadané fce
5    xlabel('x') % popis osy x
6    ylabel('y')% popis osy x
7    title('Průběh zadané funkce')% název grafu
8    for i=1:n-1 % cyklus for probíhá od 1 do n-1 (dle vzorce)
9        x=vektor(i);
10       vektor(i)=eval(fce);
11   end
12   vektor(1)=vektor(1)/2;       %prvni prvek vektoru
13   vektor(end)=vektor(end)/2;   %posledni prvek vektoru
14   soucet=0;
```

Obrázek 14: První z dvojice plagiátů odhalených pomocí DTW

```
1    x=a:0.1:b; % hodnoty x v rozsahu a-b s krokem 0.1
2    for i=1:9 % cyklus for probíhá od 1 do n-1 (dle vzorce)
3        x=vektor(i);
4        vektor(i)=eval(fce);
5    end
6    y=eval(fce); % hodnoty y zadané fce
7    vektor(1)=vektor(1)/2;       %prvni prvek vektoru
8    vektor(end)=vektor(end)/2;   %posledni prvek vektoru
9    soucet=0;
```

Obrázek 15: Druhý z dvojice plagiátů odhalených pomocí DTW

Při jednoduchém porovnání map těchto dvou kódů, dojde k chybnému závěru, že jsou rozdílné. První kód má na čtvrtém až sedmém řádku vnořené funkce pro zobrazení grafu a jeho popsání, ale z funkčního hlediska není do kódu nic přidáno. Dále proměnná *y* je posunuta nad *for* cyklus. Ten sám o sobě je také posunut. U těchto kódů je mapa pro *for* cyklus jednorozměrná, protože se v každém vyskytuje pouze jeden. V takových případech se nepoužívá k porovnání DTW, ale podobnost je vyhodnocena pouze na základě vzdálenosti míst, kde se klíčová slova v kódu vyskytují.

4.6 Vyhodnocení plagiátorství

Celkové vyhodnocení zda dva zpracovávané kódy jsou plagiáty, stojí na jednoduché metrice. Z obou kódů se berou jednotlivé nedetekované příznaky a navzájem se porovnávají. Porovnání se provádí prostým vydělením detekovaných počtů klíčového znaku u obou kódů, jak uvádí rovnice *(1)* kde P_i znázorňuje výsledný poměr, A_i znázorňuje počet znaků u prvního kódu a B_i počet znaků u druhého kódu.

$$P_i = \frac{\min(A_i, B_i)}{\max(A_i, B_i)} \tag{1}$$

Použití funkce pro minimum v čitateli a funkce pro maximum ve jmenovateli zajišťuje to, že je menší počet znaků dělen větším počtem znaků a celkový poměr tak bude vždy v intervalu od nuly do jedné.

Každý tento poměr je dále váhován, v závislosti na své důležitosti. Vzhledem k tomu, že nastavování vah na konstantní hodnoty vede ve většině případů, a to i v jiných oblastech než je detekce plagiátorství, k nepřesným výsledkům, byla pro tento detektor navržena nová metoda adaptivního váhování.

Váha V_i pro váhování poměru P_i je určená z velké části porovnáním nedetekovaných map pro daný klíčový znak, pomocí již dříve zmíněné metody DTW. Po aplikaci metody je z takto upravených map vypočten korelační koeficient, určující jejich celkovou podobnost. Tato hodnota je ještě dále upravena pomocí kombinace poměrů klíčových znaků, které určují velikosti zpracovávaných kódů. Celé vytvoření váhy V_i popisuje rovnice (*2*), kde P_m je korelační koeficient upravených map pomocí DTW, P_{bi} je poměr velikostí kódů v bitech, P_{ri} je poměr počtu řádků obou kódů a P_{ci} je poměr počtu znaků.

$$V_i = P_{mi} \cdot \frac{4 \cdot P_{bi} + 5 \cdot P_{ri} + P_{ci}}{10} \tag{2}$$

Jednotlivé poměry určující velikost zpracovávaných kódů jsou váhovány konstantními hodnotami, určených podle důležitosti jednotlivých poměrů a dohromady jsou zprůměrovány, vytvářejíce tak jednu hodnotu, zastupující celkový poměr velikosti kódů.

Poslední vahou, která se k vyhodnocení plagiátorství používá je váha V_p popsaná rovnicí (*3*).

$$V_p = \frac{C_{mi}}{C_i} \tag{3}$$

Znázorňuje poměr klíčových znaků použitých pouze v jednom ze dvou porovnávaných kódů, hodnota C_{mi}, oproti celkovému počtu klíčových znaků použitých v obou kódech, hodnota C_i. Celkově tedy tato váha určuje jedinečnost obou kódů. V případě že se hodnota blíží jedné, jsou kódy zcela rozdílné, naopak pokud se blíží nule, kódy používají veliké množství stejných funkcí, což však ještě nemusí znamenat, že se jedná o plagiát.

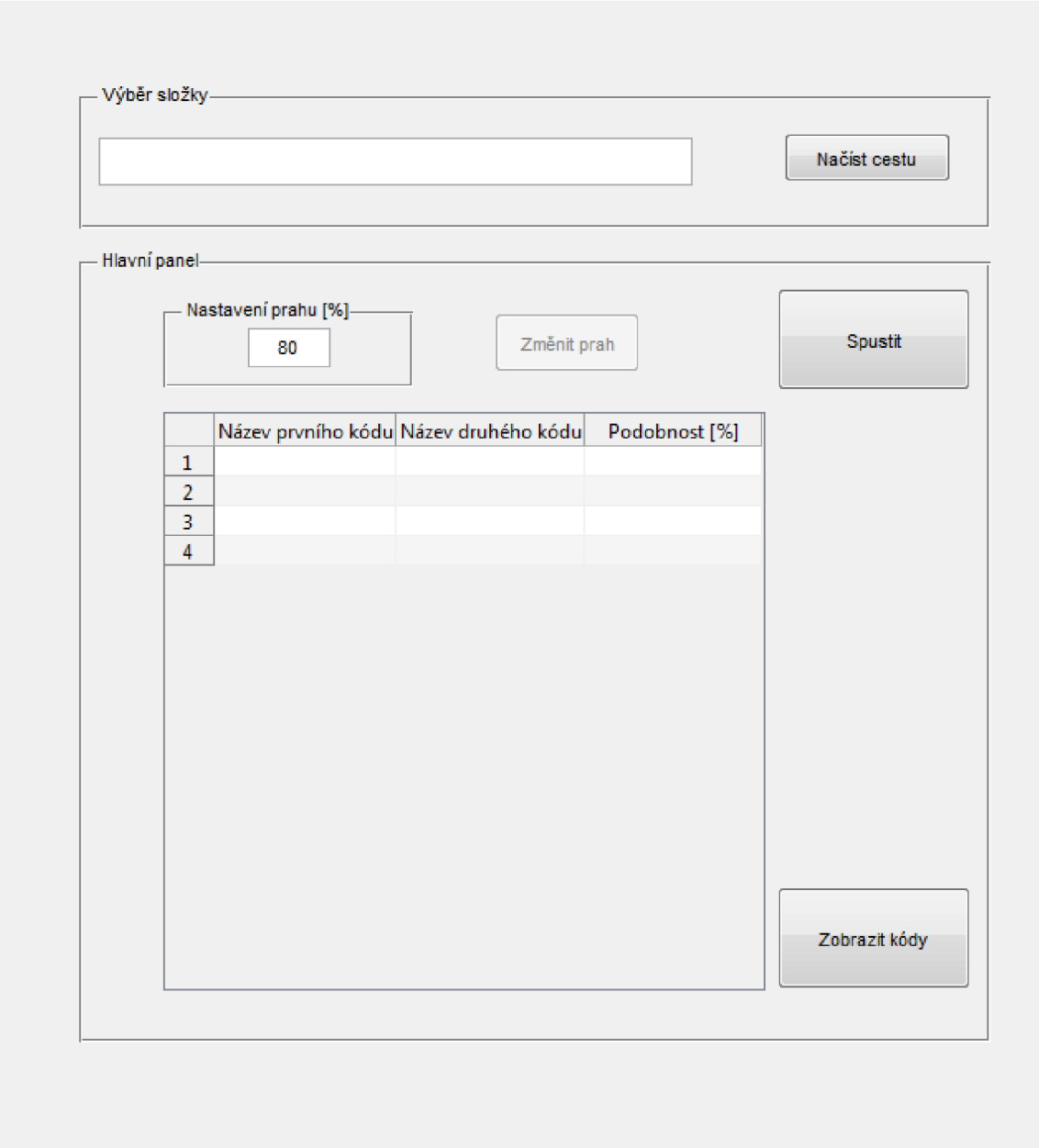
Celkové vyhodnocení podobnosti dvou zpracovávaných kódů udává rovnice (*4*).

$$P = \frac{\sum_{i=1}^{N} P_i \cdot V_i}{\sum_{i=1}^{N} V_i} \cdot (1 - V_p) \tag{4}$$

P_i tedy udává poměr dvou příznaků, V_i znázorňuje váhu získanou porovnáním map a výpočtem korelačního koeficientu a V_p je vahou jedinečnosti obou kódů. Všechny jednotlivé prvky rovnice byly vytvořeny tak, aby dosahovaly velikostí od nuly do jedné, stejných hodnot dosahuje i výsledná podobnost. [11]

5 GRAFICKÉ UŽIVATELSKÉ ROZHRANÍ

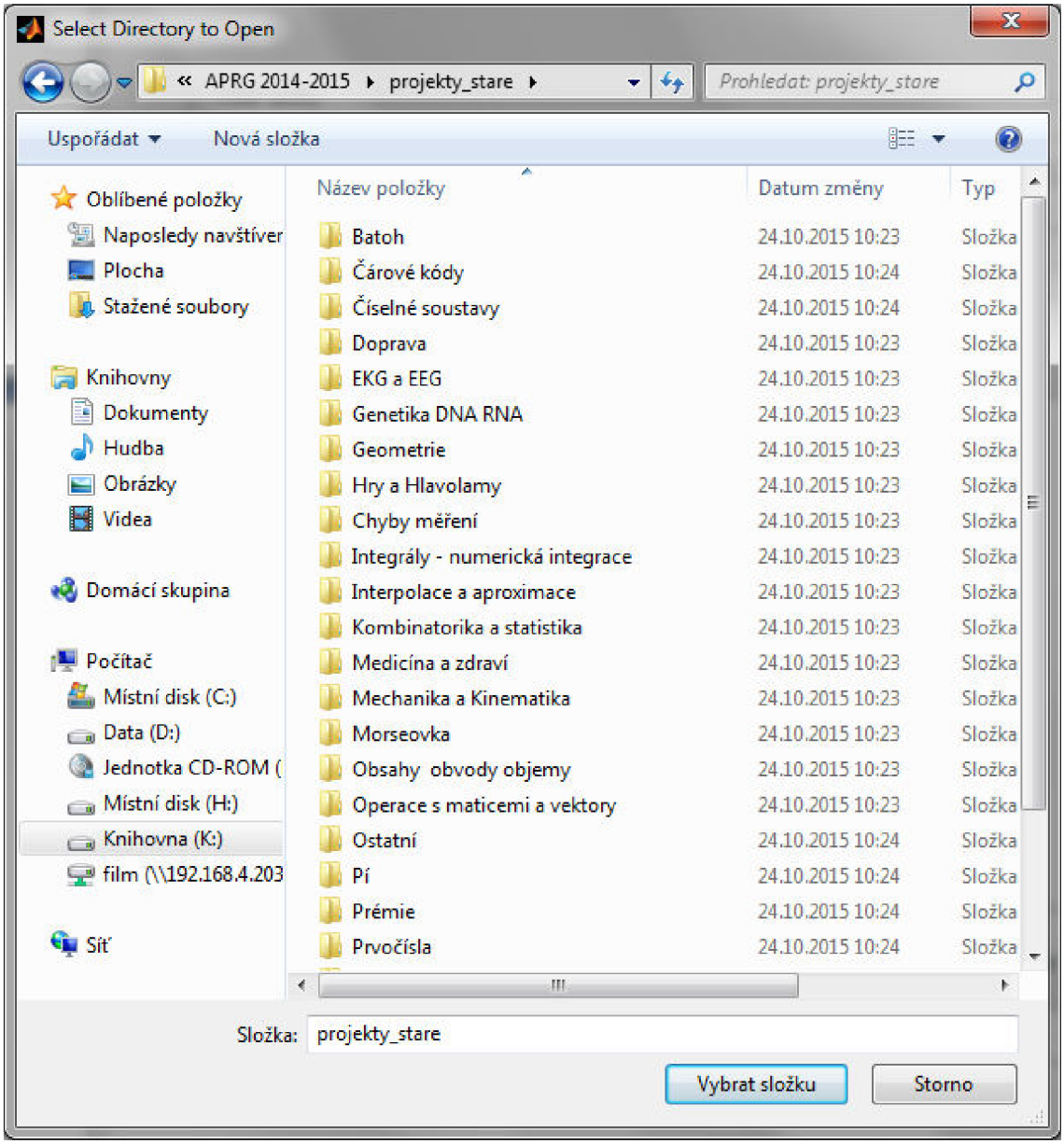
Pro přehledné a jednoduché zacházení byl program opatřen grafickým uživatelským rozhraním (GUI). Po spuštění programu se zobrazí hlavní panel, ukázaný na *Obrázek 16*.



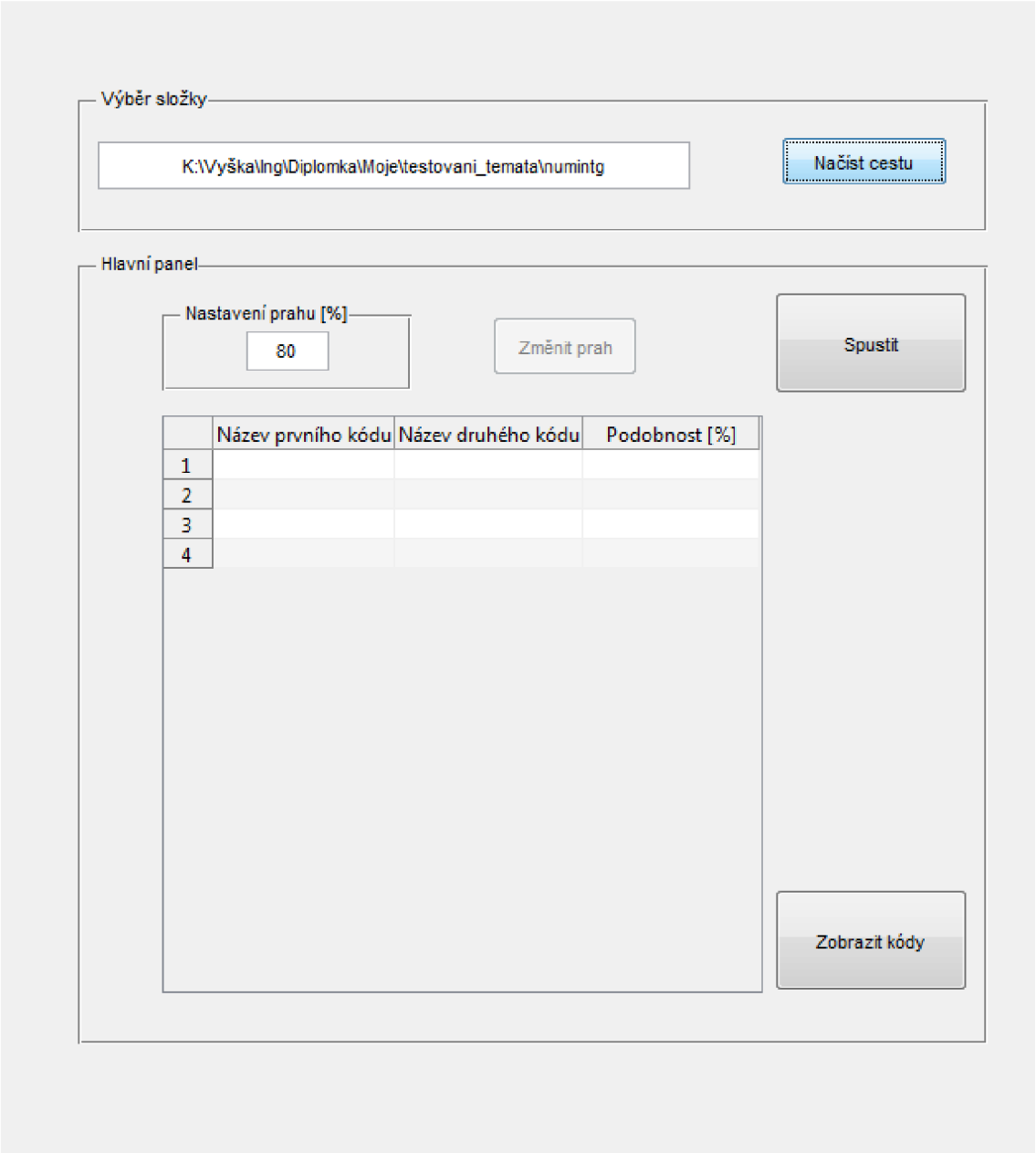
Obrázek 16: Hlavní panel detektoru

Po kliknutí na tlačítko *Načíst cestu* se otevře výběrové okno viz. *Obrázek 17*. Celý program je vytvořen tak, že dokáže porovnávat více kódů najednou, takže při výběru

stačí vybrat složku, kde jsou uloženy všechny kódy, určené pro porovnání. Program zároveň pracuje i se všemi podadresáři, které jsou ve vybraném adresáři, což výrazně zlehčuje práci při porovnávání velkého množství kódů, získaných z různých zdrojů, s různě strukturovaným ukládáním.

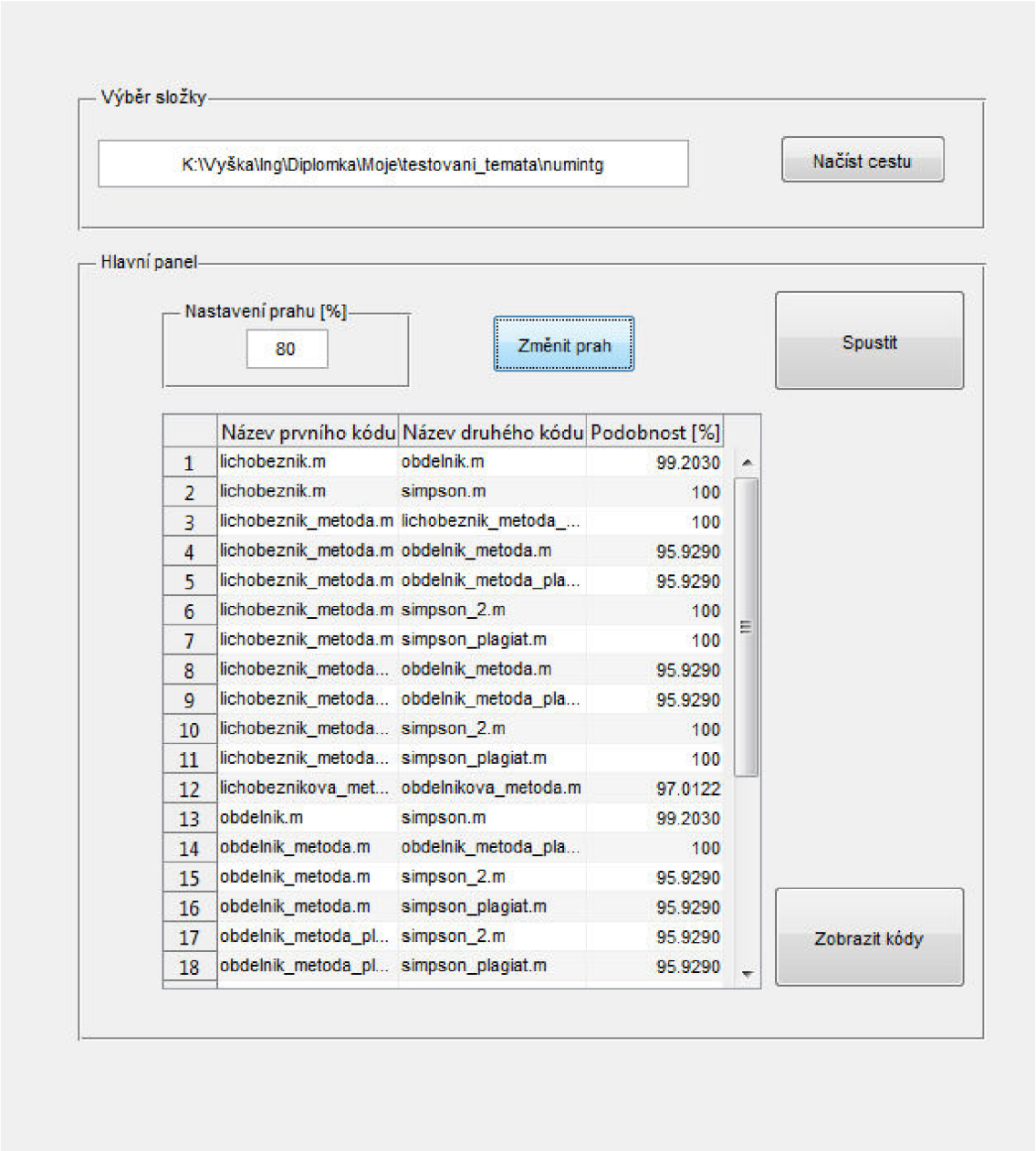


Obrázek 17: Okno na výběr složky se zpracovávanými kódy



Obrázek 18: Hlavní panel po nastavení cesty

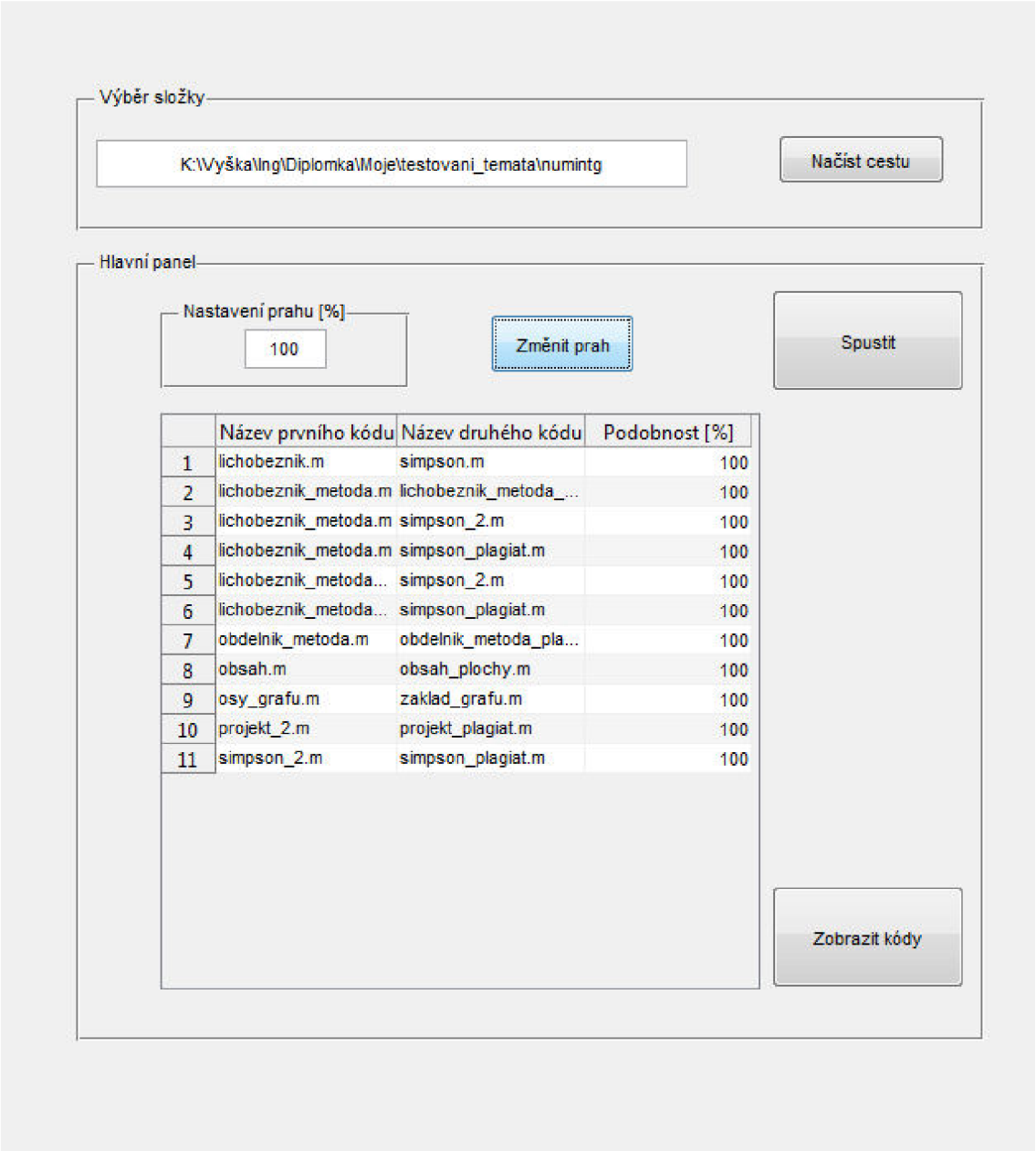
Poté co je nastavena cesta k složce s kódy na zpracování, se tato cesta pro názornost vypíše do kolonky, jak je uvedeno na *Obrázek 18* a je možné celý program spustit. Pokud by byla snaha spouštět vyhodnocení před nastavením cesty, program upozorní uživatele dialogovým oknem. Před samotným spuštěním je také možné nastavit práh, podle kterého se bude určovat, jak moc podobné kódy budou po vyhodnocení zobrazovány. Defaultně je hodnota prahu nastavena na 80 %. Pro přenastavení prahu stačí přepsat číslo v příslušné kolonce.



	Název prvního kódu	Název druhého kódu	Podobnost [%]
1	lichobeznik.m	obdelnik.m	99.2030
2	lichobeznik.m	simpson.m	100
3	lichobeznik_metoda.m	lichobeznik_metoda_...	100
4	lichobeznik_metoda.m	obdelnik_metoda.m	95.9290
5	lichobeznik_metoda.m	obdelnik_metoda_pla...	95.9290
6	lichobeznik_metoda.m	simpson_2.m	100
7	lichobeznik_metoda.m	simpson_plagiat.m	100
8	lichobeznik_metoda...	obdelnik_metoda.m	95.9290
9	lichobeznik_metoda...	obdelnik_metoda_pla...	95.9290
10	lichobeznik_metoda...	simpson_2.m	100
11	lichobeznik_metoda...	simpson_plagiat.m	100
12	lichobeznikova_met...	obdelnikova_metoda.m	97.0122
13	obdelnik.m	simpson.m	99.2030
14	obdelnik_metoda.m	obdelnik_metoda_pla...	100
15	obdelnik_metoda.m	simpson_2.m	95.9290
16	obdelnik_metoda.m	simpson_plagiat.m	95.9290
17	obdelnik_metoda_pl...	simpson_2.m	95.9290
18	obdelnik_metoda_pl...	simpson_plagiat.m	95.9290

Obrázek 19: Hlavní panel po spuštění programu

Po dokončení porovnávání všech vybraných kódu, jsou kódy, jejichž podobnost překračuje zvolený práh, vypsány do tabulky, jak ukazuje *Obrázek 19*. Vzhledem k tomu, že vyhodnocení může zabírat při práci s velkým počtem souborů dost času, byla kromě oken zobrazujících průběh celé práce programu, na hlavní panel přidáno i tlačítko pro změnu prahu. Pokud chce uživatel po dokončení programu zobrazit soubory překračující jiný práh, stačí tuto hodnotu přepsat a kliknout na tlačítko *Změnit práh*. Výsledná tabulka se poté změní podle nastaveného prahu, bez toho, aby celý program musel provádět vyhodnocení znovu. Hlavní panel po změně prahu ukazuje *Obrázek 20*.

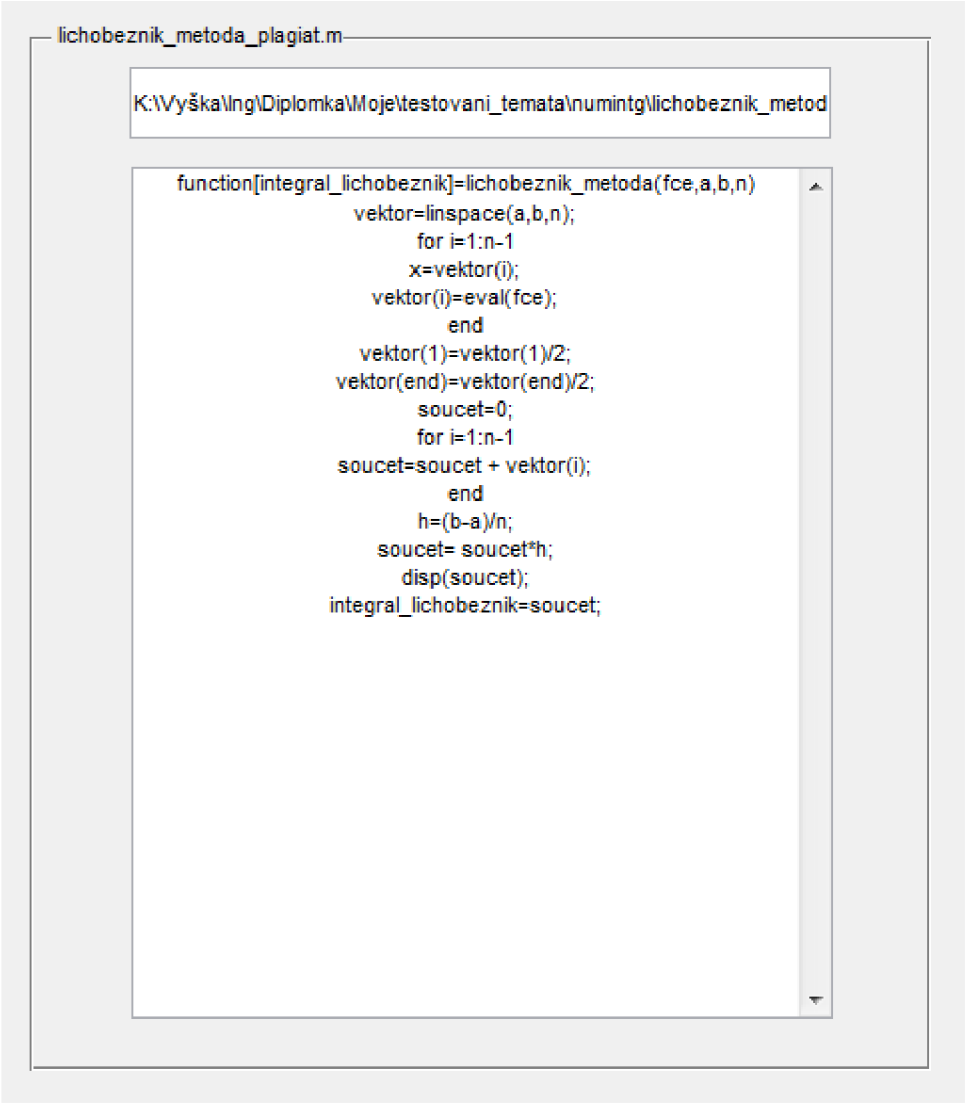


	Název prvního kódu	Název druhého kódu	Podobnost [%]
1	lichobeznik.m	simpson.m	100
2	lichobeznik_metoda.m	lichobeznik_metoda_...	100
3	lichobeznik_metoda.m	simpson_2.m	100
4	lichobeznik_metoda.m	simpson_plagiat.m	100
5	lichobeznik_metoda...	simpson_2.m	100
6	lichobeznik_metoda...	simpson_plagiat.m	100
7	obdelnik_metoda.m	obdelnik_metoda_pla...	100
8	obsah.m	obsah_plochy.m	100
9	osy_grafu.m	zaklad_grafu.m	100
10	projekt_2.m	projekt_plagiat.m	100
11	simpson_2.m	simpson_plagiat.m	100

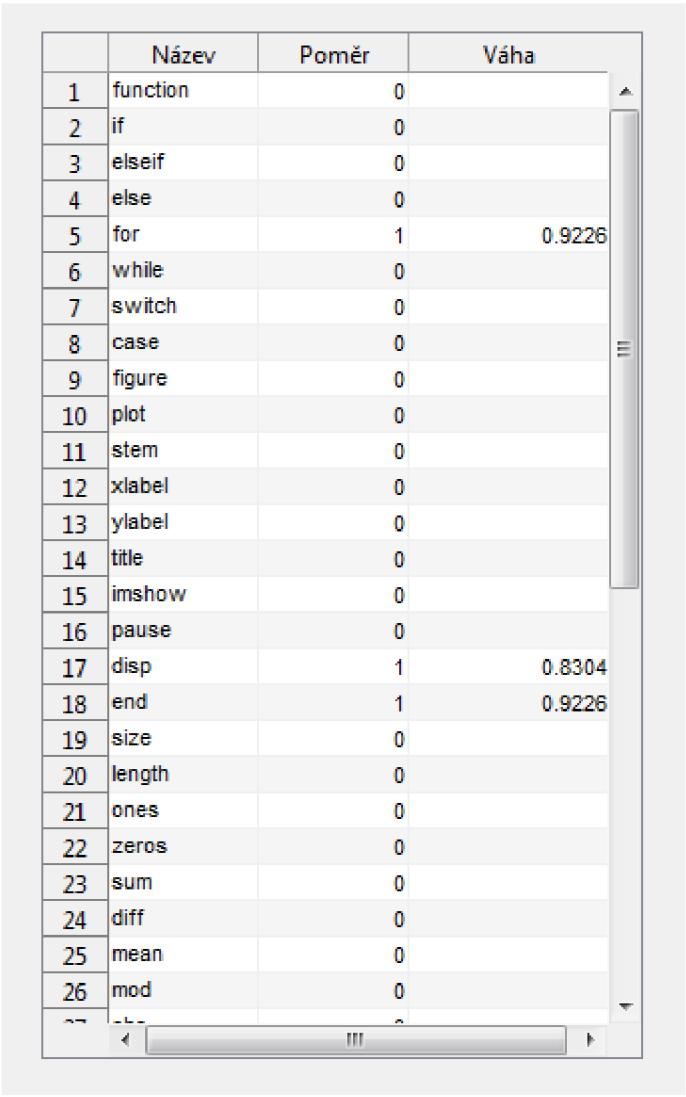
Obrázek 20: Hlavní panel po změně prahu na vyšší hodnotu

Pokud by výstupem byla pouze tabulka s názvy a podobností porovnávaných kódů, mohl by nastat problém při zpracovávání velkého počtu kódu se složitou ukládací strukturou, protože by bylo složité dohledat, o který kód se jedná. Může se i vyskytnout případ, kdy se ve vyhodnocení objeví více kódů se stejným názvem. Z toho důvodu bylo do hlavního panelu přidáno tlačítko *Zobrazit kódy*. Po vybrání určitého řádku v tabulce výsledků a kliknutí na zmíněné tlačítko se otevře nové okno. V tomto okně jsou vedle sebe zobrazeny oba předzpracované kódy k případnému osobnímu porovnání a u každého je i zapsána přesná cesta, kde se daný kód ve struktuře složek nachází. Odtud se dá cesta lehce zkopírovat a vložit do prohlížeče. Příklad tohoto okna u jednoho

kódu je na *Obrázek 21*. Kromě možnosti porovnat jak vypadají samotné kódy, je v pomocném okně k dispozici i tabulka s daty, ze kterých se určila výsledná podobnost kódů. Tuto tabulku ukazuje *Obrázek 22*.



Obrázek 21: Část pomocného okna s předzpracovaným kódem a cestou umístění



	Název	Poměr	Váha
1	function	0	
2	if	0	
3	elseif	0	
4	else	0	
5	for	1	0.9226
6	while	0	
7	switch	0	
8	case	0	
9	figure	0	
10	plot	0	
11	stem	0	
12	xlabel	0	
13	ylabel	0	
14	title	0	
15	imshow	0	
16	pause	0	
17	disp	1	0.8304
18	end	1	0.9226
19	size	0	
20	length	0	
21	ones	0	
22	zeros	0	
23	sum	0	
24	diff	0	
25	mean	0	
26	mod	0	

Obrázek 22: Tabulka v pomocném okně s uvedeným poměrem detekovaných klíčových znaků a jednotlivými vahami

6 VYHODNOCENÍ VÝSLEDKŮ

Celý program byl testován na databázi studentských projektů. Tyto projekty byly rozděleny do skupin podle témat. K dispozici byl i seznam již objevených plagiátů. Rozdělení a počet jednotlivých kódů udává *Tabulka 2*.

Tabulka 2: Rozdělení kódů do skupin

Témata	Počet kódů	Počet známých případů plagiátorství
Hry a hlavolamy	183	30
Numerické integrování	106	8
Genetika - DNA a RNA	56	0

Z výsledků dosažených detektorem jsou spočítány dvě hodnoty hodnotící jeho účinnost. Jedná se o senzitivitu (SE) a specificitu (SPE). SE udává kolik procent z celkového počtu plagiátů, bylo skutečně detekováno. SPE udává kolik procent z porovnávaných kódů byly správně neoznačeny jako plagiáty. Vzorce pro výpočet těchto hodnot určují rovnice (*5*) a (*6*), kde *TP* je množství správně detekovaných plagiátů, TN je množství kódů, které nejsou plagiáty, *FN* je množství plagiátů které nebyly detekovány a *FP* je počet kódů, které byly chybně za plagiáty označeny.

$$SE = \frac{TP}{TP + FN} \cdot 100\,, \tag{5}$$

$$SPE = \frac{TN}{TN + FP} \cdot 100\,, \tag{6}$$

Výsledky tohoto testování ukazuje *Tabulka 3*. Jak je vidět, detektor byl schopen odhalit všechny známé případy plagiátu. Dále byl schopen ve všech skupinách odhalit nové případy plagiátorství, o kterých se dosud nevědělo. Hodnoty SE byly počítány pouze na základě známých případů plagiátorství, proto u tématu genetiky, kde žádný plagiát původně nebyl, je tato hodnota vynechána.

Tabulka 3: Statistické vyhodnocení detektoru

Statistická hodnota	Hry a hlavolamy	Numerické integrování	Genetika - DNA a RNA
SE	100 %	100 %	x
SPE	99,46 %	99,47 %	98,76 %
Počet známých případů plagiátorství	30	8	0
Počet nových případů plagiátorství	16	6	10
Počet falešně pozitivních detekcí	18	30	20
Celkový počet porovnání	3377	5671	1596

Jak je z *Tabulka 3* vidět, detektor ve všech skupinách označil několik kódů mylně za plagiáty. Většinou se tomu však stávalo v případech, kdy oba kódy psal jeden člověk, nebo když měly oba kódy stejné zadání. Nejvíce tomu tak bylo ve skupině numerického integrování. Velké množství kódů tam bylo zaměřeno na lichoběžníkovou a obdélníkovou metodu. Obě metody, ač využívají jiného matematického vzorce, se dají naprogramovat v podstatě úplně stejně. Pokud je navíc programuje jeden člověk, nemá potřebu program nějak výrazně měnit a přepíše jen minimum řádků.

V případě ukázaném na *Obrázek 23* a *Obrázek 24*, jsou oba kódy ve skutečnosti skoro totožné. Oba však plní lehce jinou matematickou funkci a oba byly napsány stejným člověkem, takže se nejedná o plagiát. Detektor však vyhodnocuje pouze celkovou podobnost kódu a takovéto detaily už není schopen odhalit.

```
function [i,n,chyba]=lichobeznik(fce,a,b,eps)
disp('lichoběžníková metoda:')
disp('**********************')
n=2;
h=(b-a)/n;
x=a:h:b;
y=eval(fce);
i=h*((y(1)+y(n+1))/2+sum(y(2:n)));
d=['integrál: ',num2str(i)];
disp(d)
d=['n: ',num2str(n)];
disp(d)
chyba=1e6;
while chyba>=eps
disp('_____________________')
is=i;
n=2*n;
h=(b-a)/n;
x=a:h:b;
y=eval(fce);
i=h*((y(1)+y(n+1))/2+sum(y(2:n)));
chyba=abs(i-is)/3;
d=['integrál: ',num2str(i)];
disp(d)
d=['n: ',num2str(n)];
disp(d)
d=['chyba: ',num2str(chyba)];
disp(d)
end
```

Obrázek 23: První příklad kódu od jednoho člověka se stejným zadáním

```
function [i,n,chyba]=obdelnik(fce,a,b,eps)
clc
disp('obdélníková metoda:')
disp('********************')
n=2;
h=(b-a)/n;
x=a:h:b;
y=eval(fce);
i=h*(sum(y(1:n)));
d=['integrál: ',num2str(i)];
disp(d)
d=['n: ',num2str(n)];
disp(d)
chyba=1e6;
while chyba>=eps
disp('_____________________')
is=i;
n=2*n;
h=(b-a)/n;
x=a:h:b;
y=eval(fce);
i=h*(sum(y(1:n)));
chyba=abs(i-is)/3;
d=['integrál: ',num2str(i)];
disp(d)
d=['n: ',num2str(n)];
disp(d)
d=['chyba: ',num2str(chyba)];
disp(d)
end
```

Obrázek 24: Druhý příklad kódu od jednoho člověka se stejným zadáním

Druhým případem kde se hodně vyskytovaly problémy se správnou detekcí, byly projekty na téma genetiky. V mnoha případech tomu bylo ze stejného důvodu jako u numerického integrování, ale byly i případy, kdy si kódy byly velice podobné použitými funkcemi a rozdíl byl pouze v pár proměnných. Příklad zobrazuje *Obrázek 25* a *Obrázek 26*.

```
function bases=basecount(seq)
d=length(seq);
a=0;
c=0;
g=0;
t=0;
for k=1:d
switch seq(k)
case 'a'
a=a+1;
case 'c'
c=c+1;
case 'g'
g=g+1;
case 't'
t=t+1;
end
end
bases.a=a;
bases.c=c;
bases.g=g;
bases.t=t
```

Obrázek 25: První příklad kódu se stejnými funkcemi

```
function kodujici_vlakno_dna=generovani(pocet_nukleotidu);
kodujici_vlakno_dna=[];
for i=1:length(pocet_nukleotidu)
pom=floor(4*rand);
switch pom
case 0
kodujici_vlakno_dna=[kodujici_vlakno_dna 'a'];
case 1
kodujici_vlakno_dna=[kodujici_vlakno_dna 'g'];
case 2
kodujici_vlakno_dna=[kodujici_vlakno_dna 'c'];
case 3
kodujici_vlakno_dna=[kodujici_vlakno_dna 't'];
end
end
```

Obrázek 26: Druhý příklad kódu se stejnými funkcemi

V obou kódech je použito stejné množství funkcí *for*, *switch*, *case* a *length*. Vzhledem k tomu jak jsou oba kódy krátké, tak jsou v podstatě i na dost podobných místech. Funkce *case,* jsou navíc v obou dvou případech vzdáleny ve stejných intervalech od sebe. Jediné co se liší je distribuce proměnných a ta sama o sobě nebyla dostatečně silná na vyvrácení podobnosti kódů. Tyto případy falešné detekce se

vyskytují celkově hlavně u krátkých kódů, s malým množstvím použitých funkcí.

Vzhledem k tomu, že detektor nedokáže vyhodnotit funkčnost programu a pracuje pouze na porovnávání klíčových znaků a map, dochází často ke špatné detekci při hromadném porovnání kombinace různých témat. Stejně jako v předchozím případě u kódu s genetikou stačí, aby byly kódy krátké a používaly aspoň trochu podobné funkce a jsou vyhodnoceny jako podobné. Tomuto problému lze lehce předejít kontrolováním kódů po tématech, nebo případně nastavením vyššího prahu.

Posledním nedostatkem, který se u detektoru vyskytl, byl velice ojedinělý problém u předzpracování. To je nezávislé na úpravách, které dělá programátor, ale počítá s jistým způsobem zpracování, které dělá samotný Matlab. To je ve většině případů naprosto totožné, ale objevilo se asi pět výjimek, kdy Matlab vynechal jisté úpravy a předzpracování tedy nebylo možné správně provést. Přitom ani po bližším zkoumání daných kódů nebyl nalezen žádný rozdíl, který by je odlišoval od všech ostatních kódů. Pokud tedy takováto situace nastane, kód je stále uveden ve výsledném porovnání, ale označí se jako poškozený a musí být porovnán manuálně.

Oproti zmíněným nedostatkům stojí fakt, že navržený detektor byl schopen odhalit celkem 32 zatím nezaznamenaných plagiátů. Jeden z těchto případů ukazuje *Obrázek 27* a *Obrázek 28*. Jedná se o příklad typického plagiátorství, kdy jsou změněny pouze názvy proměnných a případně formátování programu.

```
function rna_bez_intronu=eliminace_intronu(rna)
krok=1;
k=0;
rna_bez_intronu=[];
while krok<=length(rna)
rna_bez_intronu=[rna_bez_intronu rna(krok)];
krok=krok+1;
if k==2
k=0;
if ((rna_bez_intronu(end) == 'a') &(rna_bez_intronu(end-1) == 'g')
&(rna_bez_intronu(end-2) == 'u')) | ...
((rna_bez_intronu(end) == 'a') &(rna_bez_intronu(end-1) == 'a')
&(rna_bez_intronu(end-2) == 'u')) | ...
((rna_bez_intronu(end) == 'g') &(rna_bez_intronu(end-1) == 'a')
&(rna_bez_intronu(end-2) == 'u'))
if length(rna)<(krok+3)
krok=krok+10;
end
while length(rna)>=(krok+2) & ~(rna(krok)=='a' & rna(krok+1)=='u' &
rna(krok+2)=='g')
krok=krok+1;
if length(rna)<(krok+3)
krok=krok+10;
end
end
end
k=-1;
end
k=k+1;
end
```

Obrázek 27: První příklad nově odhaleného plagiátorství

```
function
mediatorova_rna=posttranskripcni_uprava(primarni_transkript);
krok=1;
k=0;
mediatorova_rna=[];
while krok<=length(primarni_transkript)
mediatorova_rna=[mediatorova_rna primarni_transkript(krok)];
krok=krok+1;
if k==2
k=0;
if ((mediatorova_rna(end) == 'a') &(mediatorova_rna(end-1) == 'a')
&(mediatorova_rna(end-2) == 'u')) | ...
((mediatorova_rna(end) == 'g') &(mediatorova_rna(end-1) == 'a')
&(mediatorova_rna(end-2) == 'u')) | ...
((mediatorova_rna(end) == 'a') &(mediatorova_rna(end-1) == 'g')
&(mediatorova_rna(end-2) == 'u'))
if length(primarni_transkript)<(krok+3)
krok=krok+10;
end
while length(primarni_transkript)>=(krok+2) &
~(primarni_transkript(krok)=='a' & primarni_transkript(krok+1)=='u' &
primarni_transkript(krok+2)=='g')
krok=krok+1;
if length(primarni_transkript)<(krok+3)
krok=krok+10;
end
end
end
k=-1;
end
```

Obrázek 28: Druhý příklad nově odhaleného plagiátorství

Celkově detektor dosáhl kvalitních procentuálních výsledků, a tak i přes drobné nedokonalosti je možné ho reálně využívat. Hlavní výhodou je jeho plná automatizace, takže je schopen porovnání velkého množství kódů zároveň. Navíc celé porovnání je provedeno v relativně krátkém čase. 120 kódů porovnaných navzájem, což znamená 7260 operací, je provedeno za necelou půl minutu. V porovnání s časem potřebným k manuálnímu vyhodnocení, je detektor jistě lepší alternativou.

7 ZÁVĚR

Cílem této diplomové práce bylo seznámit se s tím, co všechno je považováno za plagiát a navrhnout algoritmus na porovnávání programových kódů. Celá práce je rozdělená do několika částí. V úvodu je nastíněn přibližný obraz toho, kde a v jaké formě se můžeme s plagiátorstvím setkat. V druhé části jsou popsané definice plagiátorství a dále rozepsané jeho konkrétní druhy. V další části je pak seznámení s volně dostupnými i komerčními detektory plagiátu programových kódů i prostého textu.

Vzhledem k tomu, že se většina detektorů plagiátorství používá komerčním způsobem, nejsou k tomuto tématu v podstatě žádné články popisující zhotovení těchto detektorů. Všechny jsou spíše na bázi vyhodnocení a celkového přehledu. Z toho důvodu není v teoretické části popsán žádný, již navržený postup detekce plagiátorství.

V praktické části práce pak byl ze stejného důvodu navržen vlastní postup detekce plagiátorství. První část praktické části pojednává o předzpracování kódů a jak je důležité pro následnou detekci. V další části pak byla použita jednoduchá metoda porovnání klíčových znaků, která však byla zdokonalena o algoritmus porovnávající umístění jednotlivých znaků v takzvaných mapách pomocí metody DTW. Toto porovnání je dále součástí další navržené metody, a to nového způsobu váhování, kdy se místo konstantních vah používají váhy adaptivní. Z toho důvodu není v práci uvedeno vyhodnocení kvality jednotlivých příznaků, protože v každém jednotlivém porovnání dvou kódů jsou všechny váhy jiné a jejich hodnoty záleží pouze na daných kódech.

Výsledný detektor byl otestován na databázi studentských projektů z kurzu programování. Tyto projekty byly rozděleny do skupin a porovnány mezi sebou. Z výsledků udaných detektorem byla určena hodnota senzitivity (SE) a specificity (SPE). Detektor byl schopen odhalit všechny již dříve známé případy plagiátorství, takže hodnota SE činí 100 %. V některých případech, většinou u programů se stejným zadáním, případně programů napsaných jedním člověkem, docházelo k falešně pozitivní detekci plagiátorství. I přesto však byla hodnota SPE průměrně 99,23 %. Tyto výsledky nebyly porovnány s žádným již dostupným softwarem pro detekci plagiátorství, protože ty volně dostupné nejsou schopny detekovat plagiátorství v programovém prostředí Matlab.

Hlavní výhodou celého programu je jeho úplná automatizace. Program není konstruován tak, že by bylo nutné vkládat každé dva kódy na porovnání. Naopak stačí pouze vybrat složku se všemi kódy určenými pro porovnání a to s jakýmkoli strukturováním. To znamená, že program najde kódy v hlavní složce i ve všech dalších podsložkách v ní vložených.

Program byl také opatřen, z důvodu jednoduchosti používání, grafickým

uživatelským rozhraním. To umožňuje uživateli jednoduchý výběr složky s daty a výsledky porovnání vypíše do přehledné tabulky. Zároveň je zde i možnost zobrazení porovnávaných kódů, pro jednoduché ověření, zda se opravdu o plagiáty jedná. Je zde i přiložená tabulka s průběžnými výsledky a vahami které byly ve vyhodnocení použity.

Výsledkem celé práce je tedy program navržený podle vlastní nové metody, který je vhodný k praktickému využití. Výsledky praktické části této práce byly také úspěšně prezentovány na konferenci EEICT.

LITERATURA

[1] ALSMADI, Izzat, Ikdam ALHAMI a Saif KAZAKZEH. Issues Related to the Detection of Source Code Plagiarism in Students Assignments. *International Journal of Software Engineering and Its Applications* (Vol.8, No.4 (2014), pp.23-34): [cit. 2015-12-30].

[2] ČERNOHLAVKOVÁ, Kateřina. *Plagiátorství na vysokých školách.* Brno: Masarykova univerzita, Filozofická fakulta, Kabinet knihovnictví, 2004, 108 s. Vedoucí diplomové práce Mrg. Petra Šedinová

[3] NEČADOVÁ, A. *Detekce plagiátu programových kódů.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2015. 51 s., 2s. příloh. Vedoucí diplomové práce Ing. Martin Vítek Ph.D.

[4] Knihovna.cvut.cz. *Co vše je plagiátorství.* [online]. [cit. 2015-12-30]. Dostupné z: *http://knihovna.cvut.cz/studium/jak-psat-vskp/doporuceni/plagiatorstvi/co-je-plagiatorstvi.html*

[5] Databáze Národní knihovny ČR. *Definice plagiátorství.* [online]. [cit. 2015-12-30]. Dostupné z: *http://aleph.nkp.cz/F/?func=find-c&local_base=KTD&ccl_term=wtr%3Dplagiatorství*

[6] iThenticate. *Decoding Plagiarism and Attributions Issues*. [online]. [cit. 2015-12-30]. Dostupné z: *http://www.ithenticate.com/resources/infographics/types-of-plagiarism-research*

[7] iThenticate. *Plagiarism Software*. [online]. [cit. 2015-12-30]. Dostupné z: *http://www.ithenticate.com/products/faqs*

[8] Turnitin. *What we offer.* [online]. [cit. 2015-12-30]. Dostupné z: *http://www.turnitin.com/en_us/what-we-offer*

[9] JPlag. *Detecting Software Plagiarism.* [online]. [cit. 2015-12-30]. Dostupné z: *https://jplag.ipd.kit.edu*

*[10]*Moss. *A System for Detecting Software Plagiarism.* [online]. [cit. 2015-12-30]. Dostupné z: *http://theory.stanford.edu/~aiken/moss/*

*[11]*Kašpar, J. Plagiarism *Detection In Program Codes Using Maping Technique*, *Student EEICT*, Vol. 1, pp. 174-176,2016

*[12]*GenTXWarper. *Comtutational Biology*. [online]. 16.5.2016[cit. 2016-05-16]. Dostupné z *http://www.psb.ugent.be/cbd/papers/gentxwarper/DTWalgorithm.htm*

*[13]*ResearchGate. [online]. 16.5.2016 [cit. 2016-05-16]. Dostupné z: *https://www.researchgate.net/figure/36451198_fig1_Figure-21-The-top-of-an-example-JPlag-results-overview-page*

*[14]*How does MOSS detect plagiarism?. *Quora.* [online]. 16.5.2016 [cit. 2016-05-16]. Dostupné z: *https://www.quora.com/How-does-MOSS-Measure-Of-Software-Similarity-Stanford-detect-plagiarism*

*[15]*Turnitin Originality Report. *University of Wolverhampton.* [online]. 16.5.2016 [cit. 2016-05-16]. Dostupné z: *http://www.wlv.ac.uk/about-us/internal-departments/the-college-of-learning-and-teaching-colt/learning-and-teaching-technologies/turnitin—detecting-plagiarism/turnitin-originality-report/*

SEZNAM OBRÁZKŮ

Obrázek 1: Výstup z programu JPlag [13] 14
Obrázek 2: Výstup z programu Moss [14] 15
Obrázek 3: Výstup z programu Turnitin [15] 15
Obrázek 4: Výstup z programu iThenticate [7] 16
Obrázek 5: Příklad zdrojového kódu pro vyhledávání pozice minima s formátováním '1' 19
Obrázek 6: Příklad zdrojového kódu pro vyhledávání pozice minima s formátováním '2' 20
Obrázek 7: Příklad zdrojového kódu bez předzpracování 21
Obrázek 8: Příklad zdrojového kódu po předzpracování a vložení do matice podle řádků 22
Obrázek 9: Příklad detekovaných klíčových znaků 24
Obrázek 10: Výpočetní matice pro metodu DTW [12] 26
Obrázek 11: Výsledné zarovnání dvou signálů pomocí DTW [12] 27
Obrázek 12: Mapa dvou příznaků před použitím DTW 27
Obrázek 13: Mapa dvou příznaků po použití DTW 28
Obrázek 14: První z dvojice plagiátů odhalených pomocí DTW 28
Obrázek 15: Druhý z dvojice plagiátů odhalených pomocí DTW 29
Obrázek 16: Hlavní panel detektoru 31
Obrázek 17: Okno na výběr složky se zpracovávanými kódy 32
Obrázek 18: Hlavní panel po nastavení cesty 33
Obrázek 19: Hlavní panel po spuštění programu 34
Obrázek 20: Hlavní panel po změně prahu na vyšší hodnotu 35
Obrázek 21: Část pomocného okna s předzpracovaným kódem a cestou umístění 36
Obrázek 22: Tabulka v pomocném okně s uvedeným poměrem detekovaných klíčových znaků a jednotlivými vahami 37
Obrázek 23: První příklad kódu od jednoho člověka se stejným zadáním 40
Obrázek 24: Druhý příklad kódu od jednoho člověka se stejným zadáním 40

Obrázek 25: První příklad kódu se stejnými funkcemi 41
Obrázek 26: Druhý příklad kódu se stejnými funkcemi 41
Obrázek 27: První příklad nově odhaleného plagiátorství 42
Obrázek 28: Druhý příklad nově odhaleného plagiátorství 43