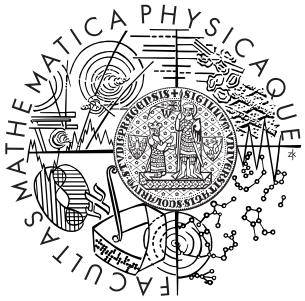
Univerzita Karlova v Prahe
Matematicko-fyzikálna fakulta

DIPLOMOVÁ PRÁCA



Matej Klaučo

Podpora rozpoznávání matematických vzorců v rámci OCR systému

Ústav teorie informace a automatizace AV ČR

Vedúci diplomovej práce: Ing. Tomáš Suk, CSc.

Študijný program: Softwarové inžinierstvo

Študijný obor: Softwarové systémy

Praha 2011

Ďakujem svojmu vedúcemu za jeho cenné poznámky a svojim rodičom a priateľom za ich podporu.

Prehlasujem, že som svoju diplomovú prácu napísal samostatne a výhradne s použitím citovaných prameňov, literatúry a ďalších odborných zdrojov. Súhlasím so zapožičiavaním práce.
Berem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona v platnom znení, najmä skutočnosť, že Univerzita Karlova v Prahe má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe dňa 20. 7. 2011 Matej Klaučo

Obsah

Zoznam obrázkov

Zoznam tabuliek

Názov práce: Podpora rozpoznávania štruktúry matematických vzorcov v rámci OCR systému
Autor: Matej Klaučo
Katedra (ústav): Ústav teorie informace a automatizace
Vedúci diplomovej práce: Ing. Tomáš Suk, CSc.
e-mail vedúceho: suk@utia.cas.cz

Abstrakt: Cieľom tejto práce je implementovať prevod naskenovaného matematického vzorca do editovateľnej podoby vo forme TEX súboru ako doplnok fungujúceho OCR systému. V práci sa venujeme podrobnej analýze tohto problému, jeho rozloženiu na niekoľko menších časti, ako rozpoznanie symbolov a rozpoznanie štruktúry vzorcu, ich riešeniu a taktiež popisujeme rôzne spôsoby riešenia. Úspešnosť našich implementácií týchto častí overujeme na nami vytvorenej databáze symbolov a vzorcov. Podstatnou časťou práce je aj vytvorenie sady komplexných aplikácií s prepracovaným grafickým rozhraním, ktoré umožnia jednoduché prispôsobenie konverzie potrebám užívateľa. Obrázky obsahujúce vzorce môžu obsahovať jemný šum spôsobený nekvalitou skeneru.

Kľúčové slová: rozpoznávanie matematických vzorcov, rozpoznávanie symbolov, podporné vektory, Kohonenova mapa

Title: Optical formula recognition support as a part of the OCR system
Author: Matej Klaučo
Department: Institute of Information Theory and Automation
Supervisor: Ing. Tomáš Suk, CSc.
Supervisor's e-mail address: suk@utia.cas.cz

Abstract: The aim of this work is to implement a conversion from the scanned math formula to the editable form as a TEX file as an extension of the working OCR system. In this work we closely analyze this problem, its division into several smaller parts, such as math symbol recognition and a recognition of structure of math formulas, and their solutions together with a description of various solutions. We test our implementations using our database of symbols and math formulas. An important part of the work is also a creation of a set of complex applications with a sophisticated graphical user interface, which allow easy accommodation of conversion to the user's needs. During the conversion we work with images, which may contain insignificant noise caused by a scanner of lower quality.

Keywords: mathematical formula recognition, symbol recognition, support vector machines, Kohonen's map

Kapitola 1

Úvod

1.1 Rozpoznávanie matematických vzorcov

Rozpoznávanie matematických vzorcov[1] je proces prekladu obrázkov, ktoré obsahujú matematické vzorce, do editovateľnej podoby. Podobne ako pri klasickom rozpoznávaní textu, rozlišujeme rozpoznávanie ručne písaného alebo tlačeného matematického textu.
Rozpoznávanie matematických vzorcov je možné rozdeliť do dvoch krokov, a to rozpoznávanie matematických symbolov a rozpoznávanie štruktúry. Vhodným výberom techniky rozpoznávania symbolov môžeme významným spôsobom pomôcť k správnemu rozpoznávaniu štruktúry [1].
Samotné rozpoznávanie symbolov je zložitý problém vzhľadom k tomu, že existuje veľký počet rôznych symbolov, mnoho symbolov sa vyskytuje v rôznych typoch fontov, prípadne veľkostí, v rámci jedného vzorca.

1.2 Súčasný stav

V súčasnosti existujú nástroje schopné rozpoznávať štandardný text bez matematických vzorcov s vysokou úspešnosťou. Tieto nástroje nie sú schopné rozpoznávať matematické vzorce. Výnimku tvorí len *Infty Reader*, ktorý je výsledkom projektu *Infty* v *Suzuki Laboratory* na *Kyushu University* v Japonsku, ktorý bol zahájený v roku 2003. Tento software slúži na rozpoznávanie textu z článkov z matematických žurnálov a jeho úspešnosť rozpoznania symbolov je viac ako 99% [2]. Text je rozpoznávaný použitím dvoch navzájom doplňujúcich sa nástrojov na rozpoznávanie symbolov, pričom prvý je zameraný na rozpoznávanie štandardného textu a druhý na rozpoznávanie matematických symbolov. Súčasťou projektu *Infty* je aj voľne dostupná databáza symbolov *InftyCDB-1* [3]
Klasické OCR programy pracujú so vzorcami ako s obrázkami, čo užívateľom komplikuje prácu pri digitalizácii textov, ktoré obsahujú matematické vzorce. V

[1] OFR - Optical Formula Recognition

súčasnosti neexistuje nástroj, ktorý by umožňoval rozpoznávať text a zároveň užívateľom definované štruktúry.

1.3 Cieľ

Cieľom našej práce je implementovať rozšírenie fungujúceho OCR systému o možnosť rozpoznávania matematických vzorcov a ich prevod do editovateľnej podoby. Primárne budú naskenovaný text a vzorce prevádzané do TEXovej podoby. Výstupnú formu bude možné meniť. V práci sa zameriame na obrázky obsahujúce text a matematické vzorce, ktoré budú obsahovať tmavý text na svetlom pozadí a budú pootočené o maximálne 2°, nakoľko bežné pootočenie skenovaného dokumentu činí v drvivom množstve prípadov maximálne 2°. Obrázky môžu obsahovať aj mierny šum. Vopred predpokladáme, že vzorce neobsahujú maticové zápisy. Cieľom práce nie je automatické vyhľadávanie vzorcov v texte, polohy vzorcov musia byť známe pred spracovaním vstupu.
Výsledkom implementácie bude okrem samotného rozšírenia OCR systému aj sada aplikácií, ktoré umožnia užívateľovi do značnej miery upraviť proces rozpoznávania vzorcov. Samotné rozpoznávanie bude implementované tak, aby bolo použiteľné aj na iné štruktúry ako matematické vzorce.

1.4 Postup práce

V práci sa zameriame na popis samotného rozpoznávania vzorcov. Rozpoznávanie vzorcov môžeme vykonať v niekoľkých za sebou nasledujúcich krokoch:

- predspracovanie obrázku do podoby vhodnej na rozpoznávanie textu, čo zahŕňa konverziu obrázka do odtieňov šedej, odstránenie šumu a v prípade potreby zvýraznenie písma. Fáze predspracovania sa venujeme v kapitole 2.

- získanie informácií o polohe a rozmeroch jednoduchých symbolov. Predpokladáme, že jednoduchý symbol sa skladá len z jednej súvislej oblasti tmavých pixelov. Tejto fáze sa venujeme na začiatku kapitoly 3.

- rozpoznanie jednoduchého symbolu, t.j. priradenie užívateľom definovaného kódu danému rozpoznanému symbolu. Tejto časti sa venujeme v kapitole 3

- rozpoznanie zložených symbolov a štruktúry textu, t.j. vytvorenie stromu, ktorý reprezentuje jeden alebo viac matematických vzorcov. Túto fázu bližšie popíšeme v kapitole 4. Okrem iného sa v tejto kapitole venujeme aj dostupným známym implementáciám tohto problému.

- dodatočné spracovanie, t.j. prípadné rozloženie vzorcov na viac riadkov a začlenenie vzorcov do štruktúry textu. Túto časť popisujeme na konci kapitoly 4.

V kapitole 5 sú uvedené demonštračné príklady, príklady existujúcich problémov a celkové výsledky našej implementácie.

Kapitola 2

Predspracovanie obrázku

Dôležitou súčasťou rozpoznávania textu a symbolov je fáza predspracovania vstupu. V tejto fáze dochádza k transformácii vstupného obrázku do podoby vhodnej na rozpoznávanie. Celý proces predspracovania sa dá rozdeliť na niekoľko fáz:

- konverzia vstupného obrázku do odtieňov šedej
- histogramová ekvalizácia - táto fáza je voliteľná a závisí od charakteru vstupného obrázku
- lokálne prahovanie [4], slúžiace na odstránenie šumu
- dodatočné odstránenie osamotených pixelov (angl. salt and pepper)

Tieto fázy procesu predspracovania sú naimplementované a používané pri rozpoznávaní textu vo fungujúcom OCR systéme [5] a v nasledujúcich podkapitolách ich stručne popíšeme.

2.1 Konverzia do odtieňov šedej

Prvú fázu predspracovania obrázku tvorí konverzia do odtieňov šedej. V takom obrázku obsahujú pixely informáciu o intenzite. Tú je možné spočítať podľa vzorca (2.1).

$$I = 0.31 \cdot R + 0.58 \cdot G + 0.11 \cdot B \tag{2.1}$$

kde R je veľkosť červenej, G veľkosť zelenej a B veľkosť modrej zložky. Obrázky, ktoré obsahujú tmavé písmo na tmavom podklade alebo svetlé písmo na svetlom podklade, majú po prevode do odtieňov šedej malý kontrast, čo je nežiadúce pri odstraňovaní šumu. Tento nedostatok eliminuje voliteľná druhá fáza predspracovania - histogramová ekvalizácia.

2.2 Histogramová ekvalizácia

Histogram digitálneho obrázka v odtieňoch šedej s veľkosťami intenzít k v rozsahu [0, L - 1] je diskrétna funkcia $h(r_k) = n_k$, kde r_k je k-ta hodnota intenzity a n_k je počet pixelov v obrázku s intenzitou k. Pravdepodobnosť výskytu intenzity veľkosti r_k v obrázku je aproximovaná podľa

$$p(r_k) = \frac{n_k}{MN} \qquad k = 0, 1, 2, ..., L-1 \tag{2.2}$$

kde MN je celkový počet pixelov v obrázku, n_k je počet pixelov s intenzitou r_k, a L je celkový počet hodnôt intenzít, ktoré môžu byt nadobudnuté (napríklad 256 pre 8-bitový obrázok).
Definujme funkciu

$$\begin{aligned} s_k = T(r_k) &= (L-1)\sum_{j=0}^{k} p_r(r_j) \\ &= \frac{L-1}{MN}\sum_{j=0}^{k} n_j \qquad k = 0, 1, 2, ..., L-1 \end{aligned} \tag{2.3}$$

Výsledný obrázok teda získame transformáciou hodnoty intenzity r_k každého pixelu vo vstupnom obrázku podla vzorca (2.3) na hodnotu s_k. Transformácia $T(r_k)$ v tejto rovnici sa nazýva *histogramová ekvalizácia* [4]. Transformácia má vo všeobecnosti tendenciu rozložiť hodnoty histogramu do väčšej škály hodnôt. Výsledkom je zlepšenie kontrastu potrebné pri ďalšom spracovaní obrázku - lokálnom prahovaní.

2.3 Lokálne prahovanie

Prahovanie je jednou zo základných transformácií intenzít pixelov v obrázku. Jej cieľom je oddeliť požadované objekty v obrázku od pozadia. Základnou metódou je vybrať prah T, ktorý vytvorí dve skupiny pixelov, v našom prípade pixely reprezentujúce symboly v texte a pixely pozadia:

- ak pre pixel na pozícii (x,y) platí, že $f(x, y) \leq T$, bude považovaný za pixel symbolu, inak
- ak platí, že $f(x, y) > T$, bude považovaný za pixel pozadia

kde $f(x, y)$ je pixel (x,y) vo vstupnom obrázku.

Globálne prahovanie je proces prahovania, pri ktorom je konštanta T aplikovaná na celý vstupný obrázok. Pri meniacej sa hodnote T sa jedná o *variabilné prahovanie*. Ak chceme zdôrazniť, že ide o variabilné prahovanie, pri ktorom hodnota T na ľubovoľnej pozícii (x,y) v obrázku závisí na vlastnostiach okolia (x,y), hovoríme o *lokálnom prahovaní*. Globálne prahovanie je možné použiť len na

obrázkoch v ktorých je zreteľné rozloženie intenzít pixelov objektov a pozadia. Variabilita vstupov si potom vynucuje použitie algoritmu na automatické určenie prahu pre konkrétny vstup. Z toho dôvodu používame lokálne prahovanie.

Základný postup pri lokálnom prahovaní je použitie priemeru intenzít m a smerodatnej odchýlky σ v okolí S_{xy} bodu (x,y). Bežné formy lokálnych prahov majú tvar

$$T_{xy} = a\sigma_{xy} + bm_{xy} \tag{2.4}$$

kde a a b sú nezáporné konštanty a

$$T_{xy} = a\sigma_{xy} + bm_G \tag{2.5}$$

kde m_G je globálny priemer intenzít pixelov. Výsledný čiernobiely obrázok g dostaneme nasledovne:

$$g(x,y) = \begin{cases} 1 & \text{ak } f(x,y) > T_{xy}, \text{ (pixel (x,y) je súčasťou pozadia)} \\ 0 & \text{ak } f(x,y) \leq T_{xy} \end{cases} \tag{2.6}$$

kde $f(x,y)$ je pixel vstupného obrázka na pozícii (x,y). Táto rovnosť je aplikovaná na každý pixel vstupu a na každej pozícii (x,y) je vypočítaný nový prah s použitím okolitých pixelov S_{xy}.

OCR systém používa modifikáciu špeciálneho prípadu lokálneho prahovania nazývaného *prahovanie s pohybujúcimi sa priemermi* [5]. Modifikácia algoritmu použitá v OCR systéme spočíva v počítaní priemeru intenzít pixelov v štvorci veľkosti $n \times n$ so stredom v aktuálnom bode. Pri výpočte ukladáme priemery jednotlivých stĺpcov takého štvorca tak, ako to popisuje vzorec 2.7. Nech n je veľkosť hrany štvorca, $s = \lfloor n/2 \rfloor$ a nech $z_{k-s} \ldots z_{k+s+1}$ označujú priemery stĺpcov výšky n so stredom v aktuálne prechádzanom riadku obrázku v k-tom kroku. Potom celkový priemer m v kroku $k+1$ počítame ako:

$$\begin{aligned} m(k+1) &= \frac{1}{n} \sum_{i=k-s+1}^{k+s+1} z_i \\ &= m(k) + \frac{1}{n}(z_{k+s+1} - z_{k-s}). \end{aligned} \tag{2.7}$$

Pretože je priemer počítaný pre každý pixel v obrázku, výsledný obrázok dostaneme použitím (2.6) s $T_{xy} = bm_{xy}$, kde b je konštanta a m_{xy} je priemer z (2.7) v bode (x,y) vstupného obrázku [4].

Modifikácia algoritmu použitá v OCR systéme spočíva v tom, že pri počítaní pohybujúceho sa priemeru berieme do úvahy štvorec veľkosti $n \times n$ so stredom v aktuálnom bode. Kladom tejto modifikácie je väčšia korešpondencia prahovej hodnoty ľubovoľného bodu s jeho okolím. Algoritmus sa tak vie lepšie vysporiadať s prítomným šumom a tým oddeliť pozadie od symbolov[5].

2.4 Odstránenie zbytkového šumu

Posledným, voliteľným krokom predspracovania obrazu je odstránenie osamotených čiernych pixelov. Vo všeobecnosti hovoríme o odstraňovaní tých pixelov, ktoré sú obklopené minimálne K pixelmi s intenzitou rôznou od intenzity daných pixelov (angl. K-fill filter). Ďalším krokom pri predspracovaní obrazu je odstránenie tých pixelov písmen alebo pozadia, ktorých najbližšie okolie obsahuje minimálne K pixelov s rovnakou hodnotou intenzity. Najbližšie okolie obsahuje práve 8 pixelov, platí teda $0 \leq K \leq 8$. Pri predspracovávaní nemá zmysel položiť $K < 6$. Pri $K = 8$ sú odstránené osamotené pixely a pri $K = 7$ dosiahneme odstránenie útvarov skladajúcich sa z dvoch pixelov. Pre $K = 6$ dôjde k odstráneniu čiar širokých jeden pixel. Odstránením týchto elementov sa zbavíme oblastí, ktoré by boli v ďalšom kroku chybne označené za symbol [5].

Kapitola 3

Rozpoznávanie symbolov

V tejto kapitole sa budeme venovať hľadaniu pozícií a rozmerov symbolov v obrázku a jeho rozpoznaniu. Našim cieľom je vybrať zo vstupného obrázku práve tie oblasti, ktoré reprezentujú vždy jeden symbol. Rôzne základné metódy riešenia tohto problému však spĺňajú cieľ vždy len do určitej miery:

- rekurzívne použitie profilov horizontálnej a vertikálnej projekcie na rozdelenie vzorcu na menšie časti obdĺžnikového tvaru. Tento spôsob sa používa priamo pri analýze štruktúry metódou zhora nadol, ktorá je podrobnejšie popísaná v sekcii 4.1. Nevýhodou tohto spôsobu je neschopnosť priamo rozpoznať zložený symbol, t.j. symbol skladajúci sa z viacerých oddelených častí (napríklad: $i, \Theta, \Xi, \div, ...$) a neschopnosť rozdelenia skupiny vnorených symbolov ($\sqrt{n}$).
- značkovanie súvislých oblastí, pri ktorej sú vo vstupnom obrázku vyhľadané súvislé oblasti čiernych pixelov. Každá súvislá oblasť je považovaná za jednoduchý symbol. Táto metóda úspešne rozpozná skupiny symbolov, ale zložené symboly sú po jej aplikácii rozdelené na niekoľko jednoduchých symbolov.

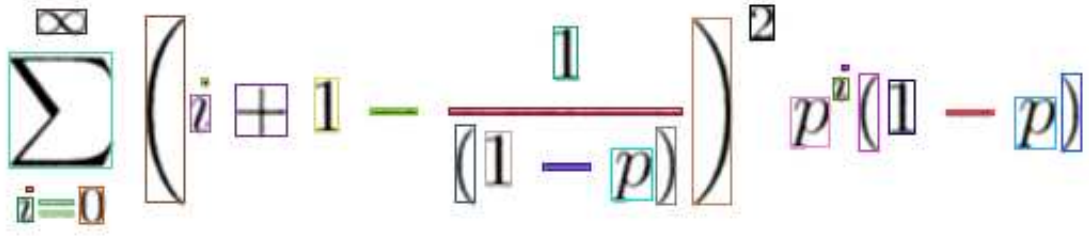
V našej práci používame druhú zmienenú metódu. Vyriešenie problému zložených symbolov prenechávame fáze rozpoznávania štruktúry vzorcov, pričom jednotlivé jednoduché symboly extrahujeme z obrázka tak, že v obdĺžniku najmenšej možnej veľkosti sa nachádza len jednoduchý symbol bez šumu a častí ostatných symbolov. V podkapitole 3.1 popíšeme spôsob získania informácii o pozícii a veľkosti symbolov. V ďalších podkapitolách detailne popíšeme spôsob klasifikácie symbolov, t.j. priradenia užívateľom definovaného kódu konkrétnemu jednoduchému symbolu.

3.1 Získanie polohy a veľkosti jednoduchých symbolov

V tejto fáze predpokladáme, že obrázok neobsahuje šum, keďže bol spracovaný postupom popísaným v kapitole 2. Pri každom symbole nás zaujíma pozícia a

veľkosť najmenšieho obdĺžniku v ktorom sa daný jednoduchý symbol nachádza. V práci používame algoritmus značkovania oblastí nazývaný *Component-Labeling Algorithm Using Contour Tracing Technique* [6] a v našej práci ho stručne rozoberieme v sekcii 3.1.1. Tento algoritmus patrí medzi jednopriechodové algoritmy, to znamená, že po prvom prejdení celého obrázka je známy výsledok. Pracuje s kontúrami značkovaných oblastí a patrí medzi najrýchlejšie zo značkovacích algoritmov. Použitím tohto algoritmu predpokladáme, že jednotlivé symboly sa nebudú navzájom dotýkať.

Príklad výsledku označovania jednoduchých symbolov je zobrazený na obrázku 3.1



Obr. 3.1: Príklad označených jednoduchých symbolov vo vzorci

3.1.1 Algoritmus označovania komponent

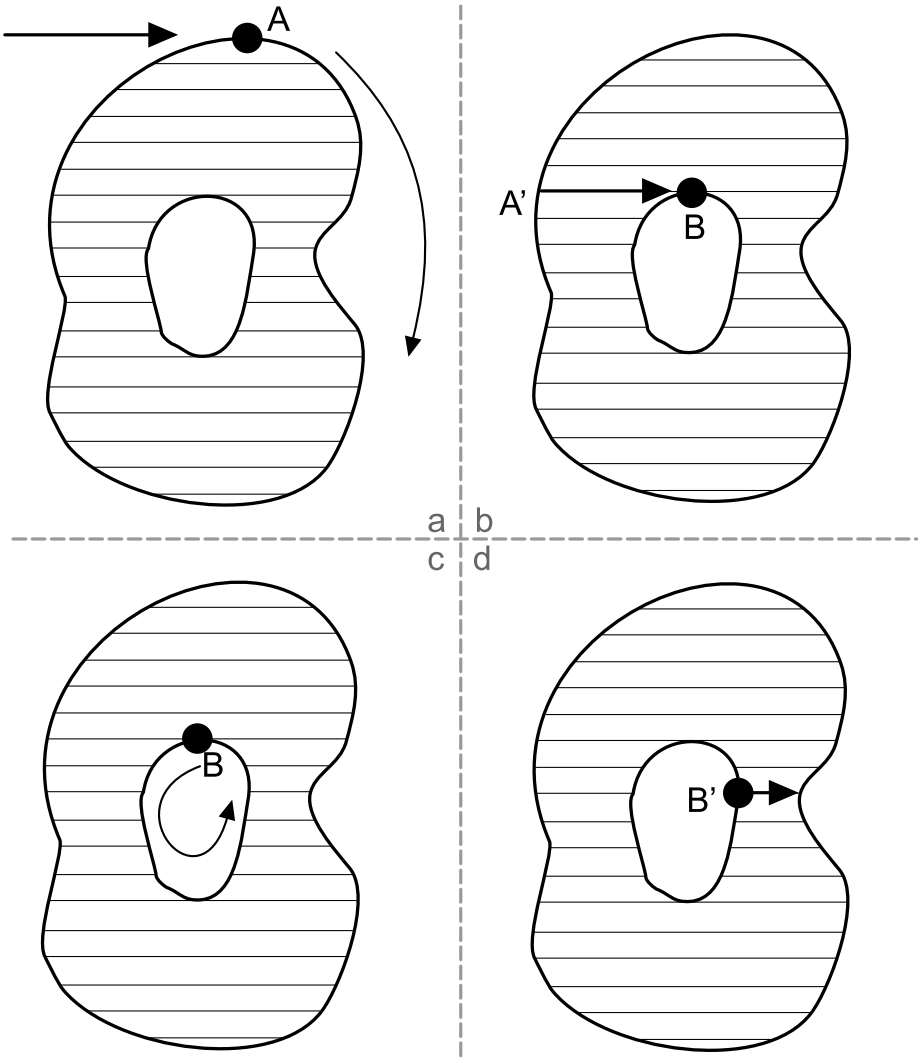
V tejto sekcii stručne popíšeme algoritmus označovania oblastí *Component - Labeling Algorithm Using Contour Tracing Technique* [6]. Algoritmus prechádza binárny obrázok zhora nadol zľava doprava postupne po riadkoch. Pri priechode obrázkom môžeme vykonávať jednu z týchto štyroch operácií zobrazených na obrázku 3.2:

1. ak prvýkrát narazíme na bod A vonkajšieho okraju, prejdeme pozdĺž celého okraju kým nenarazíme opäť na bod A tak, ako to je zobrazené na obrázku 3.2a. Bod A a všetky body okraju označíme štítkom.

2. ak narazíme na označený bod vonkajšieho okraju A', pokračujeme v skenovaní v riadku a všetkým nasledovným čiernym pixelom priradíme rovnaký štítok ako má A', viď obrázok 3.2b.

3. ak prvýkrát narazíme na bod vnútorného okraju B, priradíme mu rovnaký štítok ako majú body vonkajšieho okraju rovnakej oblasti. Potom prechádzame pozdĺž vnútorného okraju a označujeme dané body rovnakým štítkom ako má bod B, viď obrázok 3.2c.

4. ak narazíme na označený bod vnútorného okraju B', pokračujeme v skenovaní riadku a všetkým nasledovným čiernym pixelom priradíme rovnaký štítok ako má B', viď obrázok 3.2d.



Obr. 3.2: Algoritmus označovania komponent

3.2 Techniky klasifikácie symbolu

V tejto časti v stručnosti popíšeme niektoré spôsoby, akými je možné klasifikovať symboly. V prípade matematických symbolov ide o veľký počet častokrát veľmi podobných znakov, ktoré musia byť správne rozpoznané. Techniky, o ktorých je možné uvažovať, a ktorých aplikovateľnosť na tento problém už bola overená v praxi, sú nasledovné:

- klasifikácia pomocou hľadania k najbližších susedov (k-NN), t.j. k symbolov z trénovacej množiny, ktoré majú najpodobnejšie vlastnosti s predloženým symbolom. Podľa vlastností týchto symbolov je následne určená skupina, do ktorej patrí predložený symbol.
- Support Vector Machines, ktorú popíšeme v nasledujúcich častiach
- Weighted Nearest Neighbor [7] - vylepšená verzia štandardnej 1-NN. Jednotlivým symbolom z trénovacej množiny sú priradené rôzne váhy podľa toho, ako dobre reprezentujú svoju skupinu. Rôzne váhy môžu byť priradené aj extrahovaným vlastnostiam.
- Hidden Markov Models - táto metóda bola úspešne použitá pri online rozpoznávaní matematických vzorcov [8]. Nič menej, upravenú metódu je možné použiť aj na offline rozpoznávanie tak, ako je to popísané v [9].

Zmienené metódy boli otestované na databáze InftyCDB-1 [3]. Najlepšie výsledky boli dosiahnuté použitím SVM a Weighted Nearest Neighbor klasifikátorov. Najhorší výsledok dosiahla posledná zmienená metóda [1]. Úspešnosť SVM nás presvedčila o jej použití aj v našej práci. Detaily použitia popíšeme v nasledujúcich sekciách.

3.3 Základný klasifikátor

Prvý popisovaný klasifikátor symbolov je založený na dvojkrokovej stratégii. V prvom kroku je symbol na základe vonkajších vlastností zaradený do určitej kategórie symbolov. V druhom kroku sú zo symbolu extrahované ďalšie vlastnosti, na základe ktorých je identifikovaný symbol v rámci jednej kategórie. Pred samotnou klasifikáciou je nutné predspracovanie symbolu. V nasledujúcich sekciách bližšie popíšeme jednotlivé kroky.

3.3.1 Predspracovanie

Pred samotným extrahovaním akýchkoľvek vlastností symbolu je nutné znormalizovať symbol na rozmer 32×32 pixelov. Pri normalizácii používame algoritmus založený na kubickej konvolúcii a zachovávame pomer strán symbolu. Symbol je vždy umiestnený v strede štvorca.

3.3.2 Hrubá klasifikácia

Hrubá klasifikácia je založená na extrakcii obvodových vlastností symbolu. Nech obdĺžnik $(i_e - i_s) \times (j_e - j_s)$ reprezentuje symbol. Klasifikačné vlastnosti symbolu definujeme ako

$$D(i_e, i_s, j_e, j_s, L, R, T, B) \tag{3.1}$$

kde L, R, T, B sú vonkajšie vlastnosti symbolu extrahované zľava, sprava, zhora a zdola a reprezentujú počet bielych pixelov na danej strane štvorca rozmeru 32×32 pixelov, v ktorom sa nachádza normalizovaný symbol. L vypočítame nasledovne:

$$L = \sum_{i=i_s}^{i_e} \sum_{j=j_s}^{j_e} (f(i,j) \wedge flag = 0) \tag{3.2}$$

kde

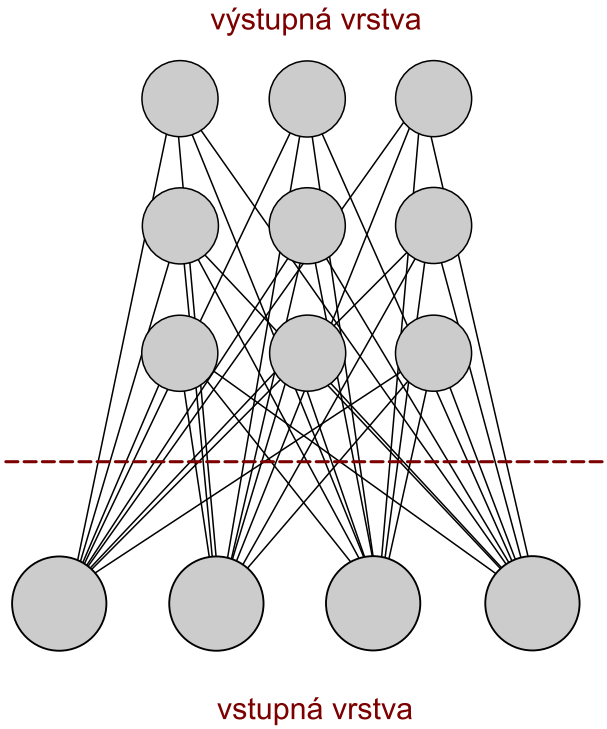
$$f(x,y) = \begin{cases} 1 & \text{ak sa na pozícii (x,y) nachádza biely pixel} \\ 0 & \text{inak} \end{cases} \tag{3.3}$$

a $flag$ je počet zmien bieleho pixelu na čierny. Na začiatku každého riadku je hodnota $flag$ inicializovaná hodnotou 0. Hodnoty R, T, B spočítame obdobne [10]. Na základe spočítaných vlastností môžeme symboly klasifikovať do niekoľkých tried. Jednotlivé triedy získame pomocou zhlukovania. Existuje množstvo techník zhlukovania, v našej práci používame Kohonenove mapy, pretože dosahuje lepšie výsledky ako tradičný algoritmus *K-means*.

Kohonenova mapa

Samoorganizujúca mapa (skr. SOM) je typ neurónovej siete, ktorá je učená bez učiteľa. Jej cieľom je vytvoriť typicky dvojdimenzionálnu diskrétnu reprezentáciu (vo všeobecnosti nízkej dimenzie) vstupného priestoru trénovacích vzoriek dimenzie N, nazývanú mapa. Samoorganizujúce mapy sa líšia od neurónových sietí tým, že používajú funkciu okolia na zachovanie topologických vlastností vstupného priestoru. Navzájom blízke vzory vo vstupnom priestore by mali byť blízko seba v mape. SOM je preto užitočná pri vizualizácii pohľadov nízkej dimenzie na vysokodimenzionálne dáta. Tento model bol prvýkrát popísaný profesorom Tuevom Kohonenom. Z jeho mena je odvodený názov Kohonenova mapa. Kohonenova mapa je zložená z N vstupných neurónov a matice výstupných neurónov. Vstupná vzorka dĺžky N je propagovaná každému výstupnému neurónu, tak ako to je zobrazené na obrázku 3.3. Do každého výstupného neurónu vstupujú vážené hrany. Váhy hrán sú inicializované malými náhodnými číslami.

Samotný proces učenia prebieha nasledovne:



Obr. 3.3: Kohonenova mapa tvorená vstupnou vrstvou veľkosti 4 a výstupnou vrstvou veľkosti 3×3. Do každého výstupného neurónu vstupujú vážené hrany zo všetkých vstupných neurónov.

Algorithm 1 Proces samoorganizácie mapy

```
inicializuj váhy pre každý výstupný neurón
while zmeny váh sú nezanedbateľné do
    for all vstupná vzorka v do
        prezentuj sieti v;
        nájdi víťazný výstupný neurón;
        nájdi všetky neuróny v okolí víťaza;
        zmeň váhové vektory všetkých nájdených neurónov;
    end for
    ak je potrebné, zredukuj veľkosť okolí výstupných neurónov;
end while
```

Víťazný výstupný neurón je neurón s váhovým vektorom, ktorý má najmenšiu Euklidovskú vzdialenosť od vstupnej vzorky. Okolie neurónu je definované ako množina neurónov nachádzajúcich sa do určitej vzdialenosti od daného neurónu na mape (nie v priestore váh). Zmena váh je počítaná podľa vzorca

$$\vec{W}_v(t+1) = \vec{W}_v(t) + \Theta(v,t)\alpha(t)(\vec{D}(t) - \vec{W}_v(t)) \tag{3.4}$$

kde $\alpha(t)$ je monotónne klesajúci koeficient učenia, $\vec{D}(t)$ je vstupná vzorka. Hodnota funkcie okolia $\Theta(v,t)$ závisí od vzdialenosti medzi víťazným neurónom a neurónom v. V najjednoduchšej podobe má hodnotu 1 pre všetky neuróny v okolí víťaza a 0 pre ostatné neuróny [12].

3.3.3 Jemná klasifikácia

Po tom, ako je neznámy symbol zaradený do jednej triedy symbolov, extrahujeme zo symbolu ďalšie vlastnosti (podrobnému popisu získania týchto vlastností sa budeme venovať v sekcii 3.3.4). Tieto vlastnosti pomáhajú rozlišovať podobné symboly v rámci jednej triedy. Každá trieda obsahuje niekoľko reprezentantov symbolov, t.j. priemerných vektorov, ktoré sú vypočítané z vektorov všetkých inštancií rovnakých symbolov, ktoré sa dostali pri trénovaní do jednej triedy.
Po extrahovaní vnútorných vlastností použijeme klasifikátor založený na *Euklidovskej vzdialenosti s odchýlkou* (skr. EDD - Euclidean distance with deviation), aby sme našli najbližšieho reprezentanta.
Nech $Y = (y_1, y_2, \cdots, y_n)$ je n-dimenzionálny vektor vlastností symbolu Y. $M = (m_1, m_2, \cdots, m_n)$ je vektor vlastností nejakého reprezentanta v triede. Potom EDD definujeme ako:

$$d_{min}(Y) = \sum_{i=1}^{n} [\max(0, |y_i - m_i| - \theta\sigma_i]^2 \tag{3.5}$$

kde σ_i je priemerná kvadratická odchýlka i-tej vlastnosti reprezentanta a θ je konštanta.

3.3.4 Extrakcia vnútorných vlastností symbolu

Extrakcia vnútorných vlastností symbolu prebieha v niekoľkých krokoch. Každému čiernemu pixelu v $[G(i,j)]_{M\times N}$, kde M a N sú rozmery normalizovaného symbolu, sú priradené štyri druhy elementov (vertikálna čiara, horizontálna čiara, čiara v uhle 45° a čiara v uhle 135°) pomocou masiek $R^1 - R^4$(3.6).

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{3.6}$$

Aplikovaním masiek získame 4 roviny vlastností $[P(i,j)]_{M\times N}$

$$\left[P^{(k)}(i,j)\right]_{M\times N} = \begin{bmatrix} P^{(k)}(0,0) & P^{(k)}(0,1) & \cdots & P^{(k)}(0,N-1) \\ P^{(k)}(1,0) & P^{(k)}(1,1) & \cdots & P^{(k)}(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ P^{(k)}(M-1,0) & P^{(k)}(M-1,1) & \cdots & P^{(k)}(M-1,N-1) \end{bmatrix}, \tag{3.7}$$

$k = 1,2,3,4, kde$

$$P^{(k)}(i,j) = \begin{cases} 1 & \rho^{(k)}(i,j) \geq 3 \\ 0 & \rho^{(k)}(i,j) < 3 \end{cases}, \tag{3.8}$$

$$\begin{aligned} \rho^{(k)}(i,j) &= \left[R^{(k)}\right]_{3\times 3} * [G'(i,j)]_{M\times N} \\ &= \sum_{m=0}^{2}\sum_{n=0}^{2} R^{(k)}(m,n)G'(i+m-1, j+n-1) \end{aligned} \tag{3.9}$$

$$G'(i,j) = \begin{cases} G(i,j) & 0 \leq i < M, 0 \leq j < N \\ 0 & \text{inak} \end{cases} \tag{3.10}$$

Každá rovina vlastností je ďalej rovnomerne rozdelená na $M'\times N'$ oblastí rozmerov $u_0 \times v_0$. Každá oblasť zasahuje u_1 pixelmi vertikálne a v_1 pixelmi horizontálne do susediacich oblastí. Z toho plynie

$$\begin{aligned} M' &= \left(\frac{M-u_0}{u_0-u_1}+1\right), \\ N' &= \left(\frac{N-v_0}{v_0-v_1}+1\right). \end{aligned} \tag{3.11}$$

Komprimované roviny vlasností $\left[E^{(k)}(i,j)\right]_{M'\times N'}, k = 1,2,3,4$ získame namapovaním každej oblasti na bod

$$E^{(k)}(i,j) = \sum_{m=0}^{u_0-1}\sum_{n=0}^{v_0-1} W^{(k)}(m,n)P^{(k)}((u_0-u_1)i+m, (v_0-v_1)j+n), \tag{3.12}$$

$$i = 0,1,\ldots,M'-1,\ j = 0,1,\ldots,N'-1, \tag{3.13}$$

kde $\left[W^{(k)}(m,n)\right]_{u_0 \times v_0}$ je matica váh:

$$W^{(k)}(m,n) = \frac{\exp\left(-\frac{(m-u_0/2)^2}{2\sigma_1^2} - \frac{(n-v_0/2)^2}{2\sigma_2^2}\right)}{2\pi\sigma_1\sigma_2},$$

$$\sigma_1 = \frac{\sqrt{2}}{\pi}u_1,$$

$$\sigma_2 = \frac{\sqrt{2}}{\pi}v_1 \tag{3.14}$$

$$m = 0, 1, \ldots, u_0 - 1,\ n = 0, 1, \ldots, v_0 - 1. \tag{3.15}$$

Výsledný $d = 4 \times M' \times N'$ dimenzionálny vektor vlastností $\mathbf{X} = [x_1, x_2, \ldots, x_d]$ dostaneme z hodnôt $\left[E^{(k)}(i,j)\right]_{M' \times N'}, k = 1, 2, 3, 4$ zoradených v poradí E^1, E^2, E^3, E^4 [15].

3.3.5 Detaily implementácie

Zo vzorcu (3.11) v sekcii 3.3.3 vidíme, že dĺžka vektoru vlastností závisí od voľby hodnôt u_0, u_1, v_0 a v_1. Od dĺžky vektoru vlastností závisí schopnosť správnej klasifikácie a schopnosť generalizácie klasifikátoru. Taktiež platí, že čím dlhší je vektor vlastností, tým náročnejšie na čas a priestor bude trénovanie klasifikátora. Preto je potrebné vybrať dostatočný počet vlastností na správnu klasifikáciu a generalizáciu tak, aby proces trénovania nebol veľmi časovo a pamäťovo náročný. V našej práci implementujeme niekoľko spôsobov extrakcie vlastností symbolu.

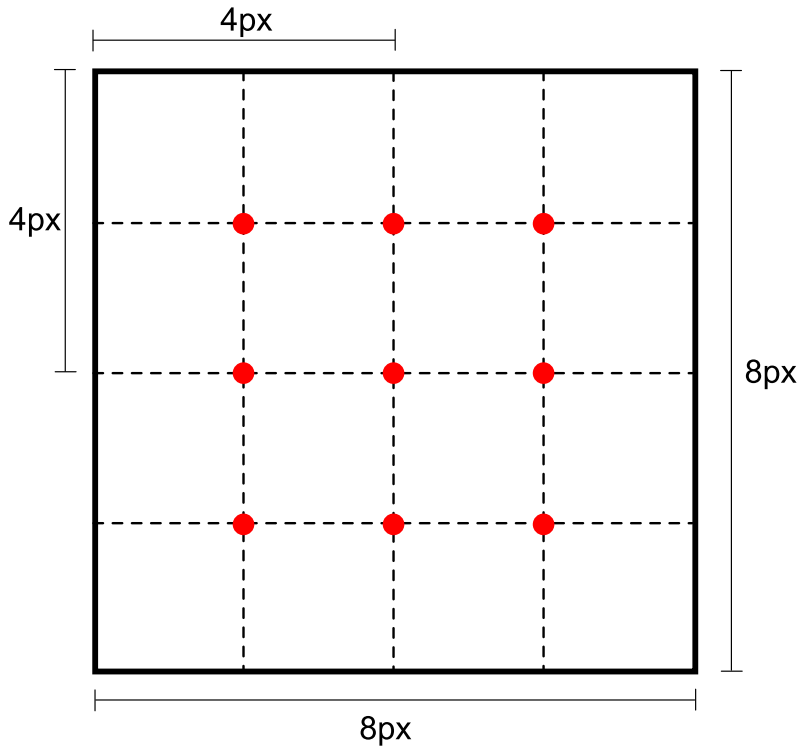
Predpokladajme, že máme normalizovaný symbol s rozmermi 32×32 pixelov. Tento štvorec rozdelíme na 16 dielov rozmeru 8×8 pixelov. V nasledujúcich odstavcoch budeme pracovať len s jedným dielom. Spôsob extrakcie vlastností z jedného dielu potom uplatníme na všetkých dieloch. Výsledný vektor vlastností F je spojením vektorov vlastností všetkých dielov.

V práci implementujeme nasledovné spôsoby extrakcie vlastností dielu:

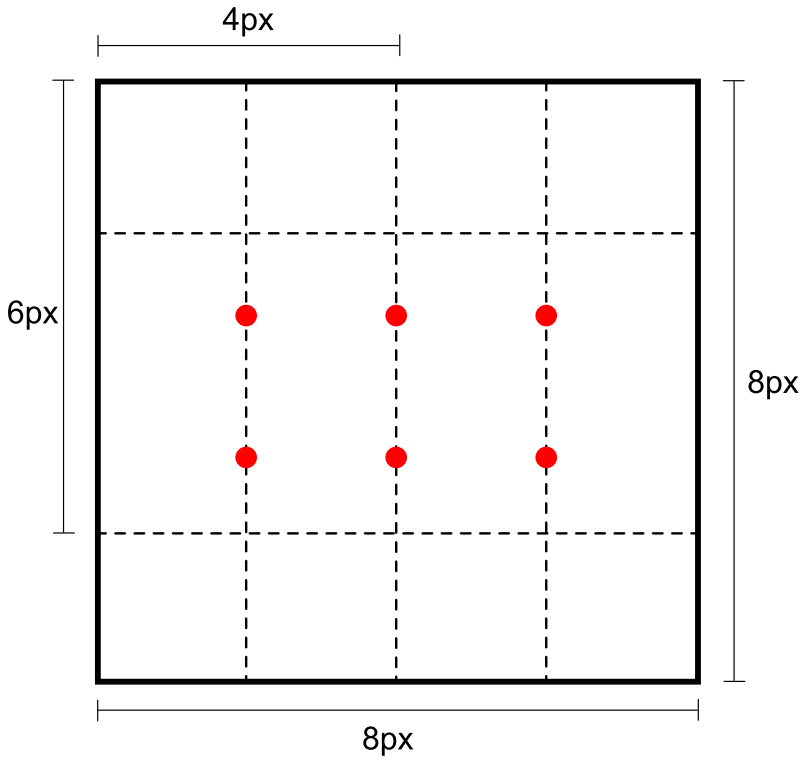
1. Extrakcia vlastností v deviatich bodoch, pri ktorej sú nastavené hodnoty $u_0 = 4, v_0 = 4, u_1 = 2, v_1 = 2$. Vznikne tak deväť oblastí, ktoré sú namapované na bod pomocou (3.12). Z toho získame deväť červených bodov znázornených na obrázku 3.4. Keďže každý bod je charakterizovaný vektorom dĺžky 4, dostávame celkovú dĺžku vektoru F rovnú $9 \times 4 \times 16 = 576$.

2. Extrakcia vlastností v šiestich bodoch, tak ako to je zobrazené na obrázku 3.5. Hodnoty parametrov sú $u_0 = 6, v_0 = 4, u_1 = 4, v_1 = 2$. Celková dĺžka vektoru F je 384.

3. Extrakcia vlastností v piatich bodoch, kde nastavujeme hodnoty $u_0 = 4, v_0 = 4, u_1 = 0, v_1 = 0$. Tento spôsob extrakcie zahŕňa položenie $\sigma_1 = \sigma_2 =$

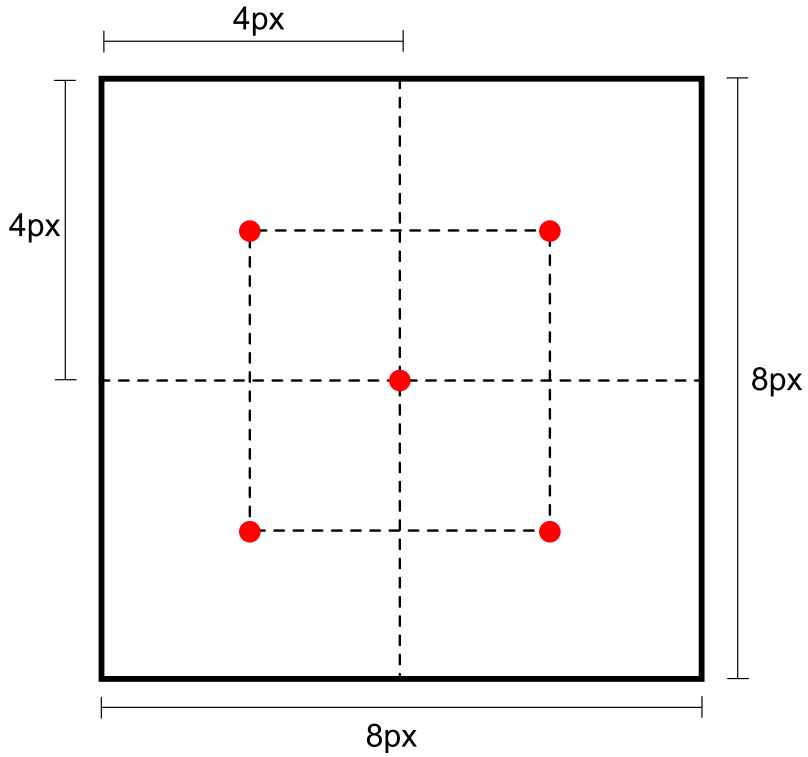


Obr. 3.4: Extrakcia vlastností deviatich bodov. Červené body sú stredmi deviatich štvorcov veľkosti 4×4 pixely



Obr. 3.5: Extrakcia vlastností šiestich bodov.

$1/(\pi\sqrt{2})$, aby nebol nulový menovateľ zlomku v (3.14). Pozície piatich štvorcových oblastí sú pevne dané tak, ako to je zobrazené na obrázku 3.6. Výsledný vektor F má dĺžku 320.

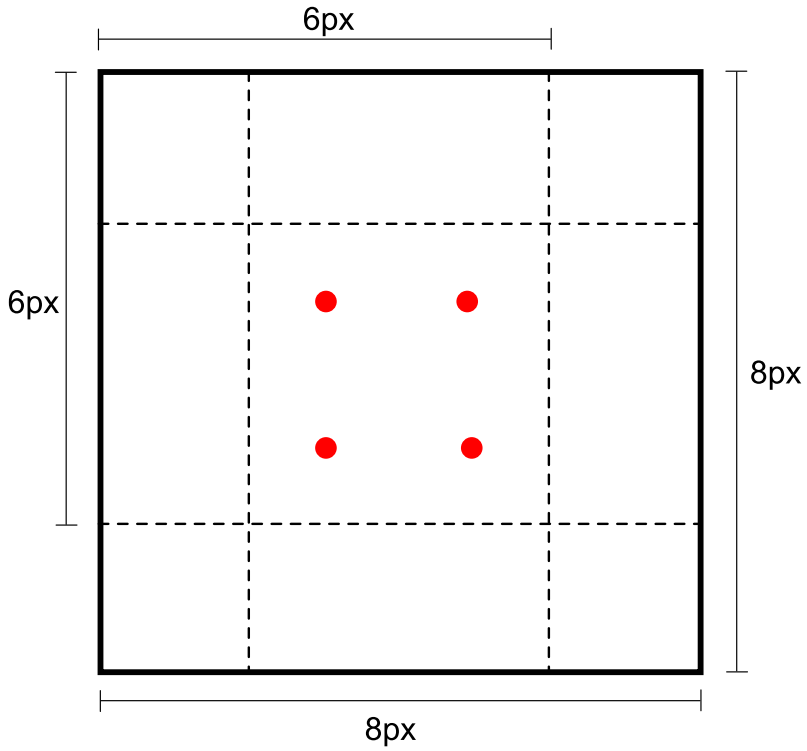


Obr. 3.6: Extrakcia vlastností piatich bodov. Červené body sú stredmi piatich štvorcov veľkosti $4px \times 4px$

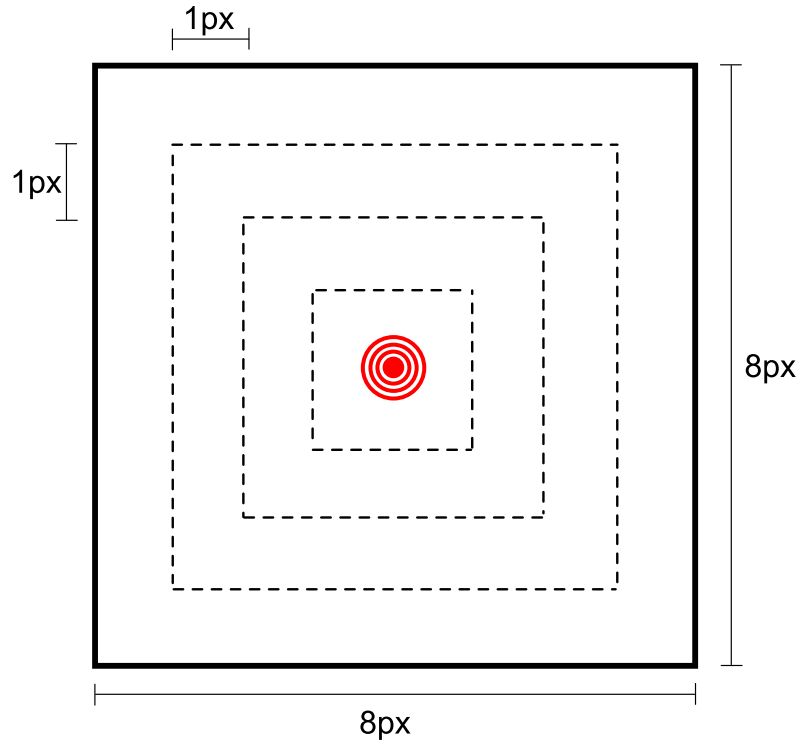
4. Extrakcia vlastností v štyroch rovnomerne rozdelených bodoch, zobrazená na obrázku 3.7. Hodnoty parametrov sú $u_0 = 6, v_0 = 6, u_1 = 4, v_1 = 4$. Dĺžka výsledného vektoru je 256.

5. Viacnásobná extrakcia vlastností v jednom bode. V jednom bode spočítame vlastnosti podľa vzorca (3.12), postupne pre hodnoty parametrov $u_0 = 8, v_0 = 8, u_1 = 0, v_1 = 0$, ďalej $u_0 = 6, v_0 = 6, u_1 = 0, v_1 = 0$, $u_0 = 4, v_0 = 4, u_1 = 0, v_1 = 0$ a nakoniec $u_0 = 2, v_0 = 2, u_1 = 0, v_1 = 0$. Podobne ako pri extrakcii vlastností v piatich bodoch, kladieme $\sigma_1 = \sigma_2 = 1/(\pi\sqrt{2})$. Spôsob extrakcie je znázornený na obrázku 3.8

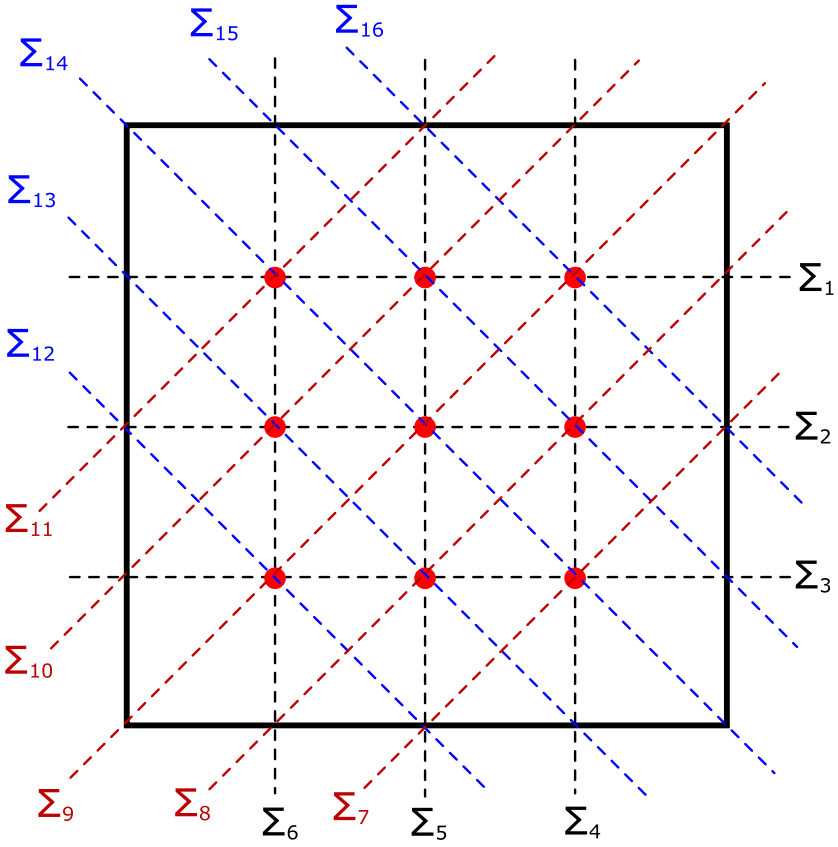
6. Extrakcia vlastností kumulovaných v rôznych smeroch s cieľom zmenšiť dĺžku výsledného vektoru. Tento spôsob je založený na vytvorení vektoru dĺžky 16, v ktorom každá položka je sumou vlastností niekoľkých bodov spočítaných v jednom smere, tak ako je to zobrazené na obrázku 3.9. Vlastnosti bodu sú počítané spôsobom popísaným pri extrakcii vlastností v deviatich bodoch. Dĺžka výsledného vektoru je 256.



Obr. 3.7: Extrakcia vlastností v štyroch bodoch.



Obr. 3.8: Viacnásobná extrakcia vlastností v jednom bode. Vlastnosti sú počítané zo zmenšujúcej sa oblasti.



Obr. 3.9: Extrakcia vlastností kumulovaných v rôznych smeroch. Σ_1 až Σ_3 sú súčty vlastností vo vodorovnom smere v červených bodoch pretínajúcich príslúšné priamky. Podobne sú spočítané v ostatných smeroch aj ďalšie sumy.

3.4 Rozšírený klasifikátor

V tejto sekcii popíšeme ďalší klasifikátor. Jeho cieľom je mať lepšiu úspešnosť rozpoznania symbolu ako je úspešnosť prvého popísaného klasifikátora. V našej práci modifikujeme klasifikátor popísaný v [13]. Proces trénovania upraveného klasifikátora vo všeobecnosti popisuje algoritmus 2.

Algorithm 2 Proces trénovania upraveného klasifikátora

for all symbol S z trénovacej množiny **do**
  klasifikuj neznámy symbol S nejakým klasifikátorom;
  označ symbol R ako výsledok klasifikácie;
  if $S \neq R$ **then**
    zaznamenaj chybu klasifikácie do matice chýb;
  end if
end for
vytvor chybové skupiny M z matice chýb;
for all chybová skupina $m \in M$ **do**
  spočítaj podporné vektory(Support Vector Machine, viď 3.4.4) pre m;
end for

Popis matice chýb a vytvárania chybových skupín popíšeme v nasledujúcich sekciách.

3.4.1 Matica chýb

Matica chýb slúži na zaznamenanie chýb, ktoré nastali pri klasifikácii v prvom kroku hlavnej slučky algoritmu 2. Na pozícii (i, j) matice sa nachádzajú všetky inštancie symbolu i, ktoré boli klasifikované ako symbol j. Jedná sa teda o štvorcovú maticu veľkosti počtu druhov symbolov trénovacej množiny.

3.4.2 Vytváranie chybových skupín

Po tom, ako máme vytvorenú chybovú maticu, môžeme pre reprezentanta symbolu S vytvoriť jemu príslušiacu chybovú skupinu, t.j. skupinu symbolov, ktoré boli výsledkom chybnej klasifikácie symbolu S. Predpokladáme, že matica chýb bude relatívne riedka, a preto ju reprezentujeme ako slovník dvojíc

$$(S, \text{hashovacia množina symbolov} H),$$

kde H obsahuje reprezentantov výsledkov chybných klasifikácií pri klasifikácii symbolu S v prvom kroku hlavnej slučky algoritmu 2. Vzhľadom k spomenutej reprezentácii vyberieme zo slovníku len tie dvojice, pre ktoré platí, že $|H| > 1$. Množiny prislúchajúce vybraným dvojiciam považujeme za chybové skupiny. V ďalších častiach textu však budeme pracovať s celými vybranými dvojicami.

3.4.3 Klasifikácia symbolu

Klasifikácia symbolu prebieha, vzhľadom k štruktúre klasifikátoru, v niekoľkých krokoch, ktoré popisuje algoritmus 3

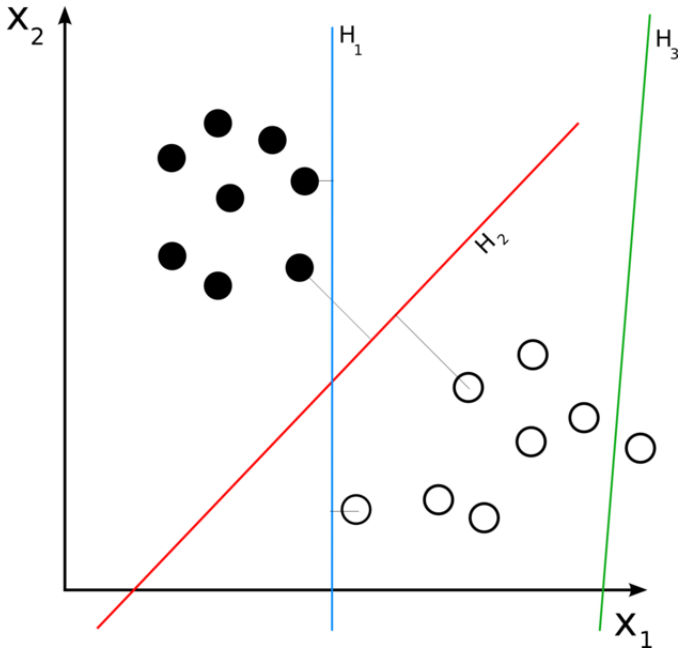
Algorithm 3 Proces klasifikácie symbolu

Require: neznámy symbol S
klasifikuj neznámy symbol S prvotným klasifikátorom;
označ symbol T ako výsledok klasifikácie;
if $\exists$ dvojica (T, H) vo vybraných dvojiciach z časti 3.4.2 **then**
V := výsledok klasifikácie symbolu S pomocou SVM,
ktorá príslúcha chybovej skupine H;
else
▷ neexistuje chybová skupina pre symbol S, t.j. môžeme použiť výsledok prvej klasifikácie
V := T;
end ifreturn V

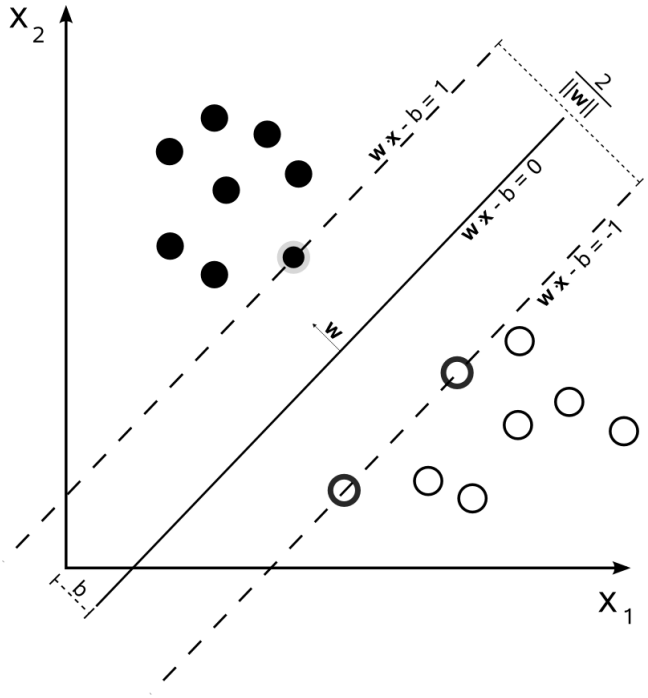
3.4.4 Support Vector Machines

Support Vector Machines (ďalej SVM) je množina príbuzných učiacich metód s učiteľom, ktoré slúžia na klasifikáciu vzorov. Štandardné SVM dostane na vstupe množinu vstupných dát a pre každú položku vstupu predpovedá, do ktorej z dvoch možných tried spadá daná položka. Hovoríme teda o binárnom lineárnom klasifikátore. Lineárny klasifikátor sa vo všeobecnosti snaží oddeliť triedy vstupov dimenzie p pomocou p-1 dimenzionálnej nadroviny.
Trénovací algoritmus vytvára model, ktorý predpovedá, či nový neznámy vstup spadá do jednej alebo do druhej triedy.
Model SVM je reprezentácia vstupov ako bodov v priestore namapovaných tak, že vstupy príslušiace daným triedam sú rozdelené najširšou možnou medzerou. Nové vstupy sú namapované do toho istého priestoru a klasifikované podľa toho, na ktorej strane medzery sa nachádza ich obraz v danom priestore, viď obrázok 3.10. Formálne, SVM vytvára nadrovinu, alebo množinu nadrovín vo vysoko alebo nekonečne dimenzionálnom priestore. Dobrú separáciu dostaneme pomocou nadroviny, ktorá má najväčšiu vzdialenosť od najbližšieho obrazu v priestore, lebo vo všeobecnosti platí, že čím väčšia je medzera, tým menšia je chyba generalizácie daného klasifikátora [14].

Často sa stáva, že problém, definovaný v priestore konečnej dimenzie, nedokážeme lineárne separovať vzhľadom k požadovaným triedam. Z dôvodu jednoduchšej separácie sa preto originálny priestor konečnej dimenzie mapuje do priestoru oveľa vyššej dimenzie. Nelineárnou transformáciou vstupného priestoru je možné použiť lineárny klasifikátor na transformovanom priestore, čo v konečnom dôsledku znamená nelineárna separácia v pôvodnom vstupnom priestore. Detaily výpočtu nebudeme v tejto práci rozoberať, je možné ich nájsť v [14].



Obr. 3.10: Oddelenie tried v priestore. Na obrázku je vidno, že priamka H_1 oddeluje dve triedy vstupov, ale veľkosť medzery je menšia ako v prípade priamky H_2. Priamka H_3 neoddeľuje triedy.



Obr. 3.11: Nadrovina vytvárajúca najväčšiu medzeru. Vzorky na okraji medzery sa nazývajú podporné vektory (angl. support vectors).

Použitie SVM na klasifikáciu do viac ako dvoch tried

SVM je neodmysliteľne binárny klasifikátor. Klasifikácia do viac ako dvoch tried (angl. multiclassification) je preto väčšinou riešená pomocou dostatočného počtu binárnych klasifikátorov. Označme C ako počet tried. Najčastejšie metódy riešenia klasifikácie do C tried sú nasledovné:

- vytvorenie C *jedna trieda - versus - zbytok* klasifikátorov a vybratie tej triedy, ktorá klasifikuje vzorku s najväčšou medzerou. Táto metóda má zaužívaný názov *one-versus-all klasifikácia*

- vytvorenie množiny *jedna trieda - versus - jedna trieda* klasifikátorov a vybratie tej triedy, ktorá bola vybraná najväčším počtom klasifikátorov. To obnáša vytvorenie $C(C-1)/2$ klasifikátorov. Celkový čas potrebný na natrénovanie klasifikátorov však môže byť menší ako pri prvej spomenutej metóde, a to vzhľadom k tomu, že tréningová množina dát pre každý klasifikátor je menšia.

- použitie tzv. štrukturálnej SVM, ktorú v tejto práci nebudeme rozoberať.

V našej práci používame knižnicu *SVM.NET* [21], ktorá umožňuje klasifikovať do viac ako dvoch tried a implementuje druhý zmienený spôsob, t.j. *jedna trieda - versus - jedna trieda*.

3.4.5 Detaily implementácie

Základný, aj rozšírený klasifikátor sú vytvorené ako samostatné zásuvné moduly. Rozšírený klasifikátor navyše podporuje načítanie ľubovoľného prvotného klasifikátoru ako zásuvného modulu, z čoho plynie jeho variabilita. Klasifikátory je možné vytvárať a trénovať pomocou aplikácie *MathSymbolTrainer*, ktorá je súčasťou práce. Vzhľadom k časovej náročnosti trénovania rozšíreného klasifikátoru je možné trénovať jednotlivé SVM prislúchajúce chybovým skupinám paralelne v dávkach zadanej veľkosti. Pri tomto spôsobe trénovania sa vytvorí potrebný počet pomocných súborov so vstupnými dátami, na ktorých treba SVM natrénovať. Na trénovanie týchto častí rozšíreného klasifikátora a následne na vytvorenie výsledného klasifikátora slúži aplikácia *DistributedPartManager*, ktorá je rovnako súčasťou práce. Vytvorenie samostatnej aplikácie umožňuje trénovanie na viacerých počítačoch. Spustením viacerých inštancií aplikácie na jednom počítači docielime vysokú mieru paralelizácie trénovania.

Kapitola 4

Rozpoznávanie štruktúry vzorcov

Posledná fáza rozpoznávania vzorcu je samotné rozpoznanie štruktúry vzorcu. V tejto fáze predpokladáme, že máme k dispozícii informácie o pozícii a veľkosti jednoduchých symbolov, a tak isto aj informáciu, o aké symboly sa jedná. V úvode tejto kapitoly rozoberieme možnosti riešenia a ich výhody a nevýhody. Následne podrobne rozoberieme vybranú metódu.

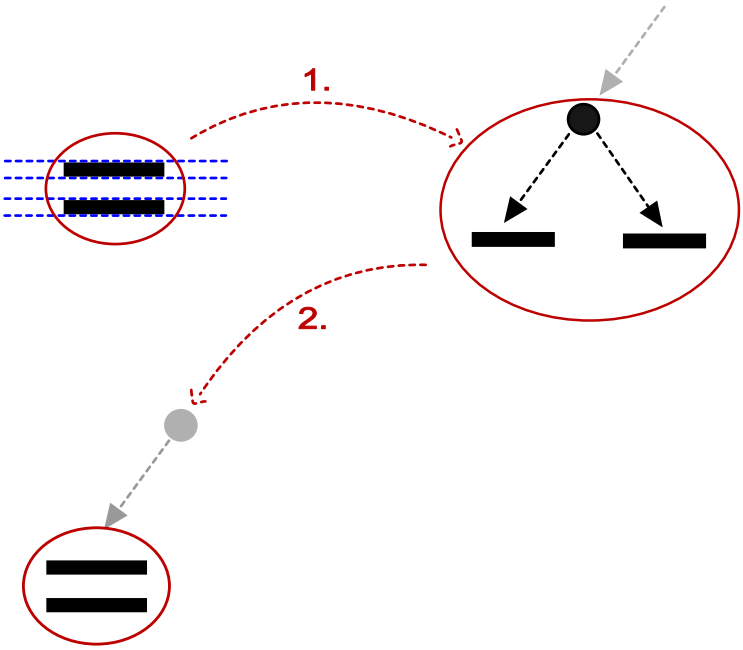
4.1 Možnosti riešenia

Rozpoznávanie štruktúry vzorcov je možné riešiť dvomi základnými spôsobmi:

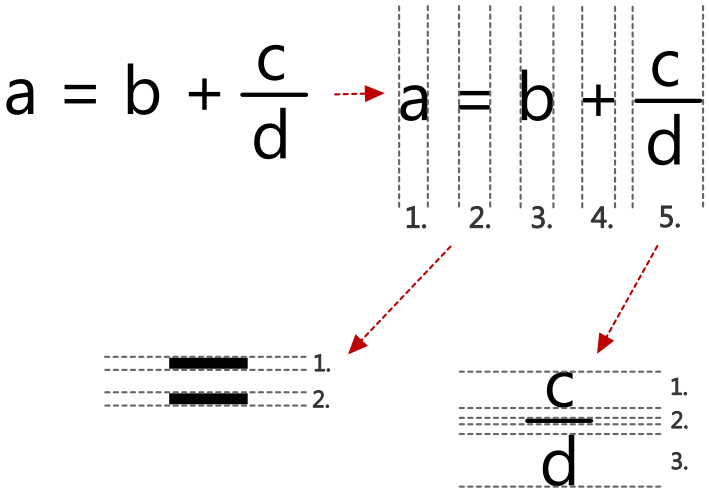
1. Riešenie metódou zhora nadol. Tento spôsob typicky znamená použitie *profilov projekcie*[1]. Veľmi často je použitie projekčných profilov spojené priamo s rozpoznávaním znakov, kedy zároveň zisťujeme štruktúru vzorca a zároveň zisťujeme polohy symbolov, ktoré môžeme rozpoznať. Tento základný spôsob riešenia však nedokáže riešiť problém horizontálne a zároveň vertikálne obklopených symbolov. Príklad takého symbolu je písmeno n vo vzorci $\sqrt{n}$. Ďalší problém tvoria zložené symboly ako napríklad $=, \leq, i, j, ...$ Takéto symboly sú pri aplikácii projekčných profilov rozdelené na viac častí tak, ako to je zobrazené na obrázku 4.1. Príklad rozpoznania štruktúry pomocou tejto metódy je znázornený na obrázku 4.2

2. Riešenie metódou zdola nahor pomocou prepisovania grafu. Prepisovanie grafu je veľmi silná technika, ktorá zovšeobecňuje prepisovanie reťazca na model prepisu grafu. Grafy sú prepisované pomocou zadefinovaných pravidiel prepisu. Tieto pravidlá špecifikujú, aká časť grafu má byť nahradená novou časťou - neterminálom. Pri rozpoznávaní vzorcov sa väčšinou pracuje s pravidlami, ktoré prepisujú časť grafu na jeden nový uzol [16]. Pri použití tejto metódy musia byť symboly vyhľadané v obrázku a rozpoznané pred aplikáciou pravidiel. Oddelením rozpoznávania symbolov od rozpoznávania štruktúry získavame modulárnejšiu štruktúru rozpoznávača vzorcov, ktorá

[1] PPC - Projection profile cutting



Obr. 4.1: Problém zložených symbolov pri použití profilov projekcie. Na obrázku je znázornená aplikácia profilov projekcie, ktorej výsledkom je časť vzorca skladajúca sa z dvoch pomlčiek. V ďalšej fáze musia byť takéto časti vzorcu objavené v celkovej jeho štruktúre, a následne nahradené za symbol =



Obr. 4.2: Príklad rozpoznania štruktúry vzorcu pomocou profilov projekcie. Po rozdelení obrázka pomocou vertikálnej projekcie sú následne rozdelené pomocou horizontálnej projekcie tie časti, ktoré obsahujú viac symbolov.

implikuje jednoduchšiu interakciu medzi časťou rozpoznávajúcou symboly a časťou rozpoznávajúcou štruktúru. To dovoľuje jednoduchší vývoj sofistikovaných techník na rozpoznanie znakov pomocou informácii o štruktúre a kontexte. Z týchto dôvodov používame v našej práci práve túto metódu. V nasledujúcich sekciách sa budeme venovať jej podrobnému popisu, predtým však popíšeme iné implementácie, a to najmä projekty InftyReader a DRACULAE.

4.1.1 InftyReader

V tejto časti stručne popíšeme spôsob rozpoznávania matematických vzorcov implementovaného v projekte InftyReader.
Problém analýzy štruktúry je reprezentovaný ako problém hľadania minimálnej kostry na váženom orientovanom grafe. Vrcholy grafu reprezentujú rozpoznané symboly, pričom graf môže obsahovať viac vrcholov (kandidátov na rozpoznané symboly) pre každý znak nachádzajúci sa v rozpoznávanom vzorci. Susedné vrcholy vytvoreného grafu sú spojené váženou hranou s popisom vzájomnej polohy dvoch vrcholov. Váha hrany korešponduje s mierou neistoty vyplývajúcej zo spojenia dvoch vrcholov. Na takom grafe je použitá modifikácia hľadania minimálnej kostry, nakoľko pre jeden znak sa v kostre musí nachádzať iba jeden kandidát z danej skupiny, a teda vo výslednej kostre sa nemôžu nachádzať všetky vrcholy grafu. Úspešnosť tejto implementácie prekračuje 95% [2].

4.1.2 DRACULAE

Ďalšou implementáciou, ktorá sa odlišuje od bežných implementácií, je *Digital Recognition Application for Computer Understanding of Large Algebraic Expressions*. Základom je vytvorenie stromu nazývaného *Baseline Structure Tree* (ďalej BST), ktorý reprezentuje hierarchickú štruktúru základných línií vo vzorci. Strom obsahuje dva druhy vrcholov:

- *vrchol reprezentujúci symbol*, ktorý je vrcholom podstromu v BST. Priami potomkovia takého vrcholu V sú vrcholy reprezentujúce oblasti v obrázku, ktoré obsahujú vnorené základné línie vzhľadom k symbolu vrcholu V.

- *vrchol reprezentujúci oblasť*, a to časť obrázka, ktorá obsahuje základnú líniu, prípadne spolu s vnorenými základnými líniami. Tento vrchol si uchováva aj informáciu o relatívnej polohe vzhľadom k nadradenému vrcholu, ktorý reprezentuje symbol.

BST je vytvorený efektívne vyhľadaním základnej línie, ktorá nie je vnorená a prechodom symbolov zľava doprava, pričom sa berie ohľad na vnorené základné línie. Takýto prístup je časovo veľmi efektívny, keďže nie je nutné používať backtracking, ako je to nutné pri použití štandardných techník prepisu grafu [17].

4.2 Vytvorenie grafu

Veľmi dôležitým krokom pri rozpoznávaní štruktúry pomocou prepisovania grafu je spôsob vytvorenia počiatočného grafu. Ak obsahuje príliš veľa hrán, tak výsledná zložitosť rozpoznania štruktúry môže byť príliš vysoká. Na druhú stranu, príliš málo hrán v grafe môže znamenať, že daný graf nebude obsahovať dôležité hrany medzi niektorými symbolmi. Z takého grafu potom nie je možné zistiť celkovú štruktúru vzorca. V našej práci hľadáme pre každý symbol jeho k najbližších susedov. Pri vhodnej voľbe k tak dosiahneme rozumný kompromis čo sa týka počtu hrán v grafe. Na vyhľadanie najbližších susedov používame 2D stromy, ktoré bližšie rozpíšeme v nasledujúcej sekcii.

4.2.1 2D stromy

2D strom je špeciálny prípad *kD stromu*. *kD strom* je dátová štruktúra slúžiaca na organizáciu bodov v k-dimenzionálnom priestore. Je to binárny strom, kde každý uzol reprezentuje bod k-dimenzionálneho priestoru. Každý vnútorný uzol vytvára deliacu nadrovinu, ktorá rozdeľuje priestor na dva podpriestory (v prípade 2D stromu rovinu na 2 polroviny). Body, ktoré sa nachádzajú na ľavej strane nadroviny, sa nachádzajú v ľavom podstrome, body napravo v pravom podstrome. Každý koreň podstromu je asociovaný s jednou z dimenzií tak, že nadrovina je kolmá na vektor danej dimenzie. V prípade 2D stromov, ak je s uzlom asociovaná prvá dimenzia, tak sú body v rovine rozdelené do dvoch skupín: skupina s x-ovou súradnicou menšou ako je x-ová súradnica daného uzlu a skupina s väčšou x-ovou súradnicou ako je x-ová súradnica daného uzlu.

Vytvorenie 2D stromu

Existuje veľa spôsobov konštrukcie tohto typu stromu, keďže existuje veľa možností, v akom poradí vyberať smer nadroviny v uzloch. Základná metóda spĺňa nasledujúce podmienky:

1. Pri prechode od koreňa k listu sú uzlom priradené deliace nadroviny podľa striedajúcich sa osí.

2. Body sú vkladané do stromu výberom mediánu z aktuálne vkladaných bodov zoradených podľa aktuálnej súradnice použitej pri vytvorení deliacej nadroviny. To vedie k tvorbe vyváženého stromu.

Vytvorenie 2D stromu popisuje algoritmus 4.

Algorithm 4 Vytvorenie2Dstromu

Input: ZoznamBodov *body_roviny*;
Input: TypOsi *aktualna_os*;
if *body_roviny* je prázdny zoznam **then return** NULL;
else
zoradˇ *body_roviny* podľa *aktualnej_osi*;
Bod prostrednyBod := **median**(*body_roviny*);
Uzol novyUzol;
novyUzol.bod := prostrednyBod;
TypOsi novaOs := **VratDalsiuOs**(aktualna_os);
novyUzol.lavySyn := **Vytvorenie2Dstromu**(body z *bodov_roviny* pred medianom, novaOs);
novyUzol.pravySyn := **Vytvorenie2Dstromu**(body z *bodov_roviny* za medianom, novaOs); **return** novyUzol;
end if

Vyhľadanie najbližších susedov v 2D strome

Vyhľadanie najbližšieho suseda daného bodu v 2D strome prebieha nasledovným spôsobom:

Algoritmus začína v koreni stromu a postupuje ním rekurzívne smerom k listu tak, ako by bol daný bod do stromu vkladaný, t.j. zostúpi do pravého alebo ľavého podstromu podľa toho či je hodnota aktuálnej súradnice zadaného bodu menšia alebo väčšia ako hodnota aktuálnej súradnice aktuálneho uzlu. Keď dorazí algoritmus do listu, zapamätá si bod, ktorý daný list reprezentuje ako doteraz najbližší nájdený bod. Pri vynáraní sa z rekurzie sú v každom uzle vykonané tieto kroky:

1. Ak je aktuálny uzol bližšie ako doteraz najbližší nájdený bod, potom sa stane doteraz najbližším nájdeným bodom.

2. Algoritmus zistí, či sa môžu nachádzať nejaké body, ležiace v druhej polrovine, bližšie k zadanému bodu ako aktuálne najbližší nájdený bod. Predpokladáme, že deliace nadroviny sú rovnobežné s osami, a tak je tento krok implementovaný ako test, či rozdiel deliacej súradnice uzlu a tej istej súradnice zadaného bodu je menší ako vzdialenosť zadaného bodu od doteraz najbližšieho bodu.

   - Ak je rozdiel menší, môžu sa nachádzať v druhej polrovine ešte bližšie body, takže algoritmus musí pokračovať priechodom druhého podstromu aktuálneho uzlu rovnakým spôsobom, ako funguje vyhladávanie.
   - Ak rozdiel nie je menší, algoritmus pokračuje vo vynáraní sa z rekurzie. Tým eliminuje celý nepreskúmaný podstrom aktuálneho bodu.

Vyhladávanie je dokončené, ked algoritmus spracuje koreň stromu [18].

V práci používame modifikáciu popísaného algoritmu. Pri vyššie zmienenom postupe by algoritmus vrátil pre ľubovoľný symbol ten istý symbol ako najbližšieho suseda, pretože sa daný symbol vo vytvorenom strome vždy nachádza. Ďalej potrebujeme nájsť k najbližších susedov bez toho, aby sme zmenili štruktúru stromu. Oba problémy riešime tak, že použijeme pomocné pole symbolov, ktoré nemôžeme pri hľadaní najbližšieho suseda použiť. Pred vyhľadaním prvého najbližšieho suseda nejakého symbolu vložíme do tohto poľa daný symbol. Tak vyriešime prvý problém. Po každom vyhľadaní najbližšieho suseda vložíme do poľa aj nájdený symbol. Tým zabezpečíme nájdenie k rôznych susedov.

4.3 Definícia pravidiel prepisu

Na to, aby sme mohli postupne vytvárať celkovú štruktúru vzorcu, musíme mať zadefinované prepisovacie pravidlá. Pravidlo nesie informácie o vzťahoch medzi susediacimi vrcholmi grafu, ktoré musia byť splnené, aby mohlo byť aplikované. Každé pravidlo má svoj názov, prioritu a identifikátor, či je pravidlo dominantné. Podobne, každý vrchol grafu, ktorý reprezentuje jeden rozpoznaný jednoduchý symbol (terminál), je pomenovaný identifikátorom daného symbolu. Vrchol reprezentujúci aplikované pravidlo je pomenovaný názvom daného pravidla. Každé pravidlo je reprezentované grafom, kde každý vrchol definuje podmienky, ktoré musia byť splnené nejakým vrcholom v reálnom grafe, a každá hrana definuje podmienky ktoré musia spĺňať vrcholy v reálnom grafe spojené hranou. Graf reprezentujúci pravidlo vždy obsahuje jeden hlavný vrchol, slúžiaci ako štartovací vrchol pri pokuse o napárovanie pravidla na podgraf reálneho grafu. V skutočnosti je graf pravidla orientovaný vzhľadom k tomu, že podmienky definované na hranách sú závislé od vlastností vrcholu z ktorého hrany vychádzajú a od vlastností vrcholu do ktorého hrany vstupujú.

4.3.1 Podmienky pre hrany

Každá hrana pravidla môže definovať podmienky na vzájomnú relatívnu pozíciu dvoch vrcholov, ďalej podmienky na relatívnu vzdialenosť a rozmery dvoch vrcholov. Definujme *zdrojový vrchol* ako vrchol z ktorého hrana grafu vychádza a *cieľový vrchol* ako vrchol do ktorého hrana grafu vstupuje. Vrcholy v reálnom grafe nesú informácie o veľkosti symbolu alebo časti vzorcu a jeho pozícii v rovine. Pod pojmami *horná hrana, spodná hrana, pravá hrana a ľavá hrana* myslíme hrany najmenšieho obdĺžniku v ktorom sa nachádza symbol alebo časť vzorcu.
Podmienky na vzájomnú relatívnu pozíciu môžu byť nasledovné (podmienky uvádzame anglickými názvami tak, ako sú definované v aplikácii na vytváranie pravidiel, viď 4.8):

- *Higher* - horná hrana cieľového vrcholu, nie však spodná, je vyššie ako zdrojový vrchol
- *Lower* - spodná hrana cieľového vrcholu, nie však horná, je nižšie ako zdrojový vrchol
- *TopLower* - horná hrana cieľového vrcholu je nižšie ako horná hrana zdrojového vrcholu
- *BottomLower* - spodná hrana cieľového vrcholu je nižšie ako spodná hrana zdrojového vrcholu
- *Above* - spodná hrana cieľového vrcholu je vyššie ako horná hrana zdrojového vrcholu
- *Below* - horná hrana je nižšie ako spodná hrana zdrojového vrcholu
- *AboveCenter* - spodná hrana cieľového vrcholu je vyššie ako horizontálna os prechádzajúca stredom zdrojového vrcholu
- *BelowCenter* - horná hrana je nižšie ako horizontálna os prechádzajúca stredom zdrojového vrcholu
- *Left* - pravá hrana cieľového vrcholu naľavo od ľavej hrany zdrojového vrcholu.
- *Right* - ľavá hrana cieľového vrcholu je napravo od pravej hrany zdrojového vrcholu
- *PartLeft* - cieľový vrchol nespĺňa podmienku *Left*, ale jeho ľavá hrana je naľavo od ľavej hrany zdrojového vrcholu
- *PartRight* - cieľový vrchol nespĺňa podmienku *Right*, ale jeho pravá hrana je napravo od pravej hrany zdrojového vrcholu
- *Inside* - cieľový vrchol sa nachádza vo vnútri zdrojového vrcholu.
- *SameOuterNode* - zdrojový aj cieľový vrchol sa nachádzajú vo vnútri toho istého vrcholu, t.j. existuje vrchol spojený hranou so zdrojovým aj cieľovým vrcholom taký, že zdrojový aj cieľový vrchol splňujú podmienku *Inside* pre daný vrchol. Táto podmienka je splnená, aj keď oba vrcholy nemajú žiaden nadradený vrchol. Pri vytváraní grafu je pri každom vrchole zaznamenaný jeho vizuálne najmenší nadradený vrchol spojený s ním hranou. Túto podmienku je možné použiť napríklad pri pravidle na matchovanie binárneho operátora, kedy sa oba operandy musia nachádzať na logicky rovnakej úrovni. Príklad: vo vzorci $\sqrt{a+b}+c$ nedôjde k namatchovaniu pravidla na symboloch $b+c$, pretože nadradeným symbolom symbolu b bude symbol odmocniny, zatiaľ čo symbol c nemá nadradený symbol. Naopak, pravidlo bude aplikované na symboloch $a+b$.

Hrana grafu pravidla môže mať definovaných viac podmienok na pozíciu, pričom pri každej podmienke je špecifikované, či musí, alebo nesmie byť splnená.

Ďalšie podmienky, ktoré je možné nastaviť pre hranu, sú nasledovné:

- *VerticalConnectionLengthRatio* - pomer vertikálnej vzdialenosti dvoch najbližších hrán vrcholov (hornej,spodnej) a výšky zdrojového vrcholu
- *HorizontalConnectionLengthRatio* - pomer horizontálnej vzdialenosti dvoch najbližších hrán vrcholov (ľavej,pravej) a šírky zdrojového vrcholu
- *WidthRatio* - pomer šírky zdrojového vrcholu k šírke cieľového vrcholu
- *HeightRatio* - pomer výšky zdrojového vrcholu k výške cieľového vrcholu
- *VerticalCenterDiffRatio* - pomer vertikálnej vzdialenosti stredov vrcholov a výšky zdrojového vrcholu
- *VerticalCenterDiffAndWidthRatio* - pomer vertikálnej vzdialenosti stredov vrcholov a šírky zdrojového vrcholu
- *HorizontalCenterDiffRatio* - pomer horizontálnej vzdialenosti stredov vrcholov a šírky zdrojového vrcholu

Pre každú z týchto podmienok sa nastavuje rozsah hodnôt, ktoré môžu byť nadobudnuté a ďalej je možné nastaviť, či sa má alebo nemá brať daná podmienka do úvahy. V prípade, že sa nemá, nedochádza k počítaniu hodnôt potrebných na overenie podmienky.

4.3.2 Podmienky pre vrcholy

Každý vrchol v grafe pravidla špecifikuje, aké terminály, alebo neterminály sa môžu vyskytovať v reálnom grafe. Ako sme spomenuli, terminály sú pomenované podľa identifikátora symbolu a neterminály podľa pravidla, ktoré vytvorilo daný neterminál. Špecifikácia prebieha pomocou štandardných regulárnych výrazov. Taktiež je možné nastaviť, či má byť matchovanie regulárneho výrazu úspešné alebo neúspešné. Okrem toho je možné nastaviť druhý regulárny výraz, ktorý slúži na vylúčenie prípadov, kedy nechceme aplikovať prvý regulárny výraz. Pri zavedení konvencie, že mená pravidiel sa budú začínať konštantným reťazcom, napr. *Rule*, môžeme efektívne písať regulárne výrazy ktoré budú matchovať všetky terminály, prípadne len neterminály, a podobne.

4.4 Rozpoznanie štruktúry grafu

Predtým, než popíšeme algoritmus rozpoznania štruktúry, definujeme:

- Dve pravidlá sú *konfliktné*, ak prienik vrcholov, na ktoré boli aplikované, je neprázdny.
- *Množina konfliktných pravidiel* je definovaná induktívne takto:
  1. Ak sú pravidlá p, p' konfliktné, tak $\{p, p'\}$ je množina konfliktných pravidiel.
  2. Ak K je množina konfliktných pravidiel a p je pravidlo, ktoré je konfliktné s nejakým pravidlom $p' \in K$, tak potom $\{p\} \cup K$ je množina konfliktných pravidiel.
- *Množina konfliktných pravidiel* $K \subseteq A$ je *maximálna* v A, ak každé pravidlo $p \in A \backslash K$ nie je konfliktné so žiadnym pravidlom z množiny K.

Algoritmus prechádza postupne všetky vrcholy grafu a overuje aplikovateľnosť všetkých pravidiel na každom vrchole postupom popísaným v 4.4.2. Pri prechode si ukladá informácie o aplikovateľných pravidlách spolu s podgrafmi, na ktorých boli aplikované.
Pri prechode vrcholov algoritmus vytvára maximálne množiny konfliktných pravidiel. Všetky pravidlá, ktoré sa nachádzajú v novovzniknutých množinách obsahujúcich len jedno pravidlo, môžu byť bezpečne aplikované (nie sú konfliktné so žiadnym iným aplikovateľným pravidlom). Tieto pravidlá sú priradené do *skupiny bezkonfliktných pravidiel*. Ostatné maximálne množiny konfliktných pravidiel sú označené ako *konfliktné skupiny* a pravidlá v nich obsiahnuté sú zoradené podľa priority zostupne.
Po vytvorení konfliktných skupín algoritmus tieto skupiny prechádza, a v prípade, že skupina obsahuje dominantné pravidlá, ponechá v nej iba dominantné. Ak po tomto kroku existuje skupina, kde sú dominantné pravidlá, odstráni z množiny konfliktných skupín tie, ktoré neobsahujú dominantné pravidlá. V ďalej popisovanej fáze prepisu tak dôjde k aplikovaniu iba dominantných pravidiel. Dôvody používania dominantných pravidiel popisujeme v podsekcii 4.4.1.
V ďalšej fáze sa algoritmus pokúsi vyriešiť konflikty tak, že vyberie vždy jedno pravidlo z každej konfliktnej skupiny, aplikuje ho na grafe a rekurzívne vykoná celý algoritmus na novom grafe, kým nenájde žiadne pravidlo, ktoré by mohol aplikovať. V takom grafe spočíta počet vrcholov. Z n konfliktných skupín je vybraná tá n-tica pravidiel, po aplikácii ktorej má finálny graf (na ktorom sa nedajú aplikovať žiadne pravidlá) najmenší počet vrcholov. Vyskúšaním všetkých možností tak dostávame graf najviac kompatibilný so zadanými pravidlami. Skúšanie všetkých možností je časovo náročné, a preto využívame fakt, že ak aktuálne spočítaný graf má práve jeden vrchol, môžeme výpočet ukončiť, pretože boli na ňom aplikované pravidlá s najvyššou prioritou a nie je nutné skúšať pravidlá s nižšou prioritou.

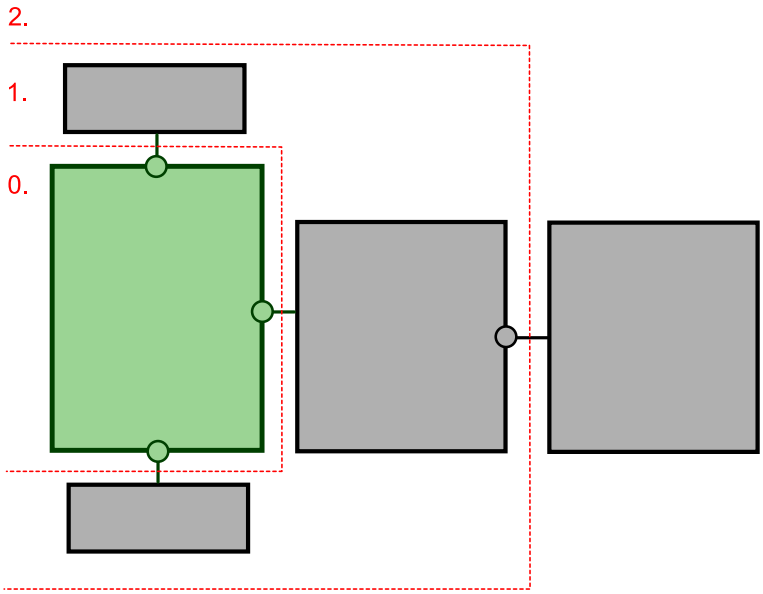
4.4.1 Dominantné pravidlá

Matematické vzorce obsahujú matematické funkcie, ako napríklad *sin* alebo *cos*. Pre každú takú funkciu musí byť vytvorené pravidlo prepisu. Ak pri rozpoznávaní

štruktúry grafu narazíme na možnosť aplikácie takého pravidla, nemusíme v jeho konfliktnej skupine uvažovať o aplikácií nedominantných pravidiel, nakoľko budú viesť k nežiadúcemu výsledku. Pri možnosti aplikácie dominantných pravidiel neaplikujeme nedominantné pravidlá, čím dôjde po konečnom počte krokov ku vytvoreniu grafu, na ktorom sa nedajú uplatniť žiadne dominantné pravidlá. Ten si môžeme v prípade funkcií vo vzorci predstaviť ako graf, ktorý má vrcholy definujúce funkcie (napr. cos) namiesto vrcholov reprezentujúcich jednotlivé znaky(c,o,s). Okrem funkcií môžu byť dominantné pravidlá aj tie, ktoré slúžia na zlúčenie symbolov zložených z viacerých častí, napríklad $i, \geq, \leq, =$.

4.4.2 Overenie aplikovateľnosti pravidla

Predpokladajme, že pravidlo je reprezentované súvislým orientovaným grafom, ktorý obsahuje jeden hlavný vrchol. Potom je tento graf možné rozdeliť na niekoľko vrstiev tak, ako to je zobrazené na obrázku 4.3.



Obr. 4.3: Príklad pravidla prepisu. Hlavný vrchol je označený zelenou farbou. Pravidlo je reprezentované súvislým orientovaným grafom, ktorý je možné rozdeliť na 3 vrstvy. V nultej vrstve sa vždy nachádza len hlavný vrchol.

Algoritmus 5 popisuje pokus o aplikáciu zadaného pravidla, a v prípade úspechu vracia všetky podgrafy, ktoré spĺňajú všetky podmienky. Algoritmus dostáva na vstupe podgraf, na ktorom bola úspešne aplikovaná nultá vrstva pravidla, t.j. hlavný vrchol. Ďalej dostáva na vstupe vrchol z grafu pravidla, ktorý je považovaný za štartovací vrchol v danej vrstve. Algoritmus pracuje rekurzívne, kde v každom jeho kroku hľadá všetky možnosti napárovania podgrafu reálneho grafu na aktuálnu vrstvu (match vrstvy) tak, aby boli splnené všetky podmienky pravidla pre danú vrstvu. Podgraf je v tomto prípade tvorený všetkými susedmi

vrcholov napárovaných s vrcholmi pravidla predchádzajúcej vrstvy. Pred samotným behom algoritmu je otestovaná aplikovateľnosť hlavného vrcholu grafu pravidla na aktuálne spracovávaný vrchol reálneho grafu.

Algorithm 5 NajdiMatche

Input: GraphMatch aktualnyMatch
Input: GraphRule pravidlo
Input: GraphNode aktualnyVrcholRealnehoGrafu
Input: GraphNode aktualnyVrcholPravidla

▷ pri porovnávní berieme do úvahy počet vrcholov aj počet namatchovaných hrán
if —aktualnyMatch— == —pravidlo— **then return** aktualnyMatch
end if
        ▷ susedný vrchol je reprezentovaný dvojicou (hrana do vrcholu vedúca, vrchol)
for all susedný vrchol (E,V) aktualnehoVrcholuRealnehoGrafu **do**
    for all susedný vrchol (F,R) aktualnehoVrcholuPravidla **do**
        if R je aplikovateľné na V a F je aplikovateľná na E **then**
            pridaj V do skupiny symbolov M_R na ktoré je aplikovateľný R
        end if
    end for
end for
vytvor matchVrstvy z tých vrcholov V reálneho grafu, pre ktoré platí
že $|M_R| = 1$ pre R z množiny susedných vrcholov *aktualnehoVrcholuPravidla*
if —aktualnyMatch ∪ matchVrstvy— == —pravidlo— **then return** aktualnyMatch + matchVrstvy
end if
M := dvojice (R,skupiny symbolov M_R obsahujúce viac ako jeden vrchol);
        ▷ vygenerujeme všetky možnosti, ako môžu byť napárované vrcholy aktuálnej vrstvy pravidla a vrcholy reálneho grafu
vsetkyMatcheVrstvy := VratVsetkyVariacie(aktualnyMatch,matchVrstvy,M);
vysledok := ∅;
if $|vsetkyMatcheVrstvy| > 0$ **then**
    for all $mv \in vsetkyMatcheVrstvy$ **do**
        for all dvojica $(R, M_R) \in M$ **do**
            for all vrchol V reálneho grafu $\in M_R$ **do**
                vysledok := vysledok ∪ NajdiMatche(aktualnyMatch ∪ mv,pravidlo, V,R);
            end for
        end for

```
27:            ▷ vyskúšame aplikovať match vrstvy, kde počiatočný vrchol bude z
   aktuálneho matchu vrstvy
28:        for all (vrchol V reálneho grafu,vrchol P pravidla) ∈ matchVrstvy do
29:            vysledok := vysledok ∪ NajdiMatche(aktualnyMatch ∪
   mv,pravidlo, V,P);
30:        end for
31:    end for
32: else
33:            ▷ vyskúšame aplikovať aktuálny match vrstvy, kde počiatočný vrchol
   bude z aktuálneho matchu vrstvy
34:    for all (vrchol V reálneho grafu,vrchol P pravidla) ∈ matchVrstvy do
35:        vysledok := vysledok ∪ NajdiMatche(aktualnyMatch ∪
   matchVrstvy,pravidlo, V,P);
36:    end for
37: end if
```

4.4.3 Pravidlá obsahujúce cyklus

Pravidlá môžu obsahovať cyklus, prípadne viac hrán môže viesť do toho istého vrcholu. Kvôli týmto prípadom sa pri vytváraní matchu vrstvy kontroluje, či namatchované vrcholy reálneho grafu sú zhodné s už namatchovanými v predchádzajúcich krokoch algoritmu, t.j. či namatchovanému vrcholu prislúcha taký istý vrchol pravidla, ako v predchádzajúcich krokoch. V prípade cyklov program *RuleCreator* neumožňuje vytvoriť priamy cyklus medzi dvomi vrcholmi.

4.5 Textová reprezentácia štruktúry

Cieľom práce nie je len rozpoznať štruktúru vzorcu, ale ju aj vypísať v lineárnej podobe. K tomuto účelu slúži vlastnosť každého definovaného pravidla - textová reprezentácia. Hodnota tejto vlastnosti je reťazec, ktorý môže obsahovať špeciálne premenné. Premenné v reťazci začínajú znakom $. Názov premennej určuje vrchol v rámci daného pravidla, ktorého textová reprezentácia má nahradiť danú premennú. Vrchol je určený podľa jeho mena.

Uvedieme príklad na prevod pravidla do TEXu:
Máme definované pravidlo na zlomky. Pravidlo obsahuje tri vrcholy *citatel*, *zlomkova_ciara* a *menovatel*. Textová reprezentácia pravidla bude vyzerať nasledovne: $nfrac\{\$citatel\}\{\$menovatel\}$. Ak je čitateľ alebo menovateľ reprezentovaný jednoduchým symbolom, nahradí sa premenná reprezentáciou tohto symbolu. Reprezentácie všetkých jednoduchých symbolov sú uložené v jednom súbore. Ak nejaký vrchol reprezentuje ďalšie aplikované pravidlo, nahradí sa príslušná premenná textovou reprezentáciou daného pravidla. Ak bol nejaký symbol rozpoznaný nesprávne, nemusí byť výstupný kód preložiteľný. V tom prípade musí užívateľ

manuálne upraviť chyby vzniknuté nesprávnym rozpoznaním.
Pri prevode do textovej reprezentácie v našej práci pre zjednodušenie neuvažujeme medzery vo vzorcoch.

4.6 Rozklad matematického textu na riadky

Predpokladajme, že v aplikácii užívateľ označil niekoľko riadkov matematického textu. Aby pri prezeraní štruktúry rozpoznaného vzorca mal užívateľ väčší prehľad, pokúšame sa rozdeliť tento text na jednotlivé riadky. Fáza rozdeľovania nasleduje po fáze rozpoznania štruktúry vzorcov v označenej oblasti. Predpokladáme, že sú pravidlá prepisu definované tak, aby nedochádzalo k splňaniu podmienok medzi vzorcami, ktoré sa nachádzajú v rôznych riadkoch. Zároveň však predpokladáme, že podmienky sú zadané tak, aby boli splniteľné v rámci jedného vzorcu. Preto môžeme predpokladať, že v jednom vytváranom riadku sa môže nachádzať jeden a viac vzorcov, prípadne ich častí. Tento predpoklad značne uľahčuje fázu rozkladu textu na riadky. Stačí nájsť vertikálne prekrývajúce sa vzorce alebo ich časti, a z nich vytvoriť riadok matematického textu. Algoritmus funguje nasledovne:

Algorithm 6 RozdelNaRiadky

Input: zoznam segmentov L, kde jeden segment reprezentuje jeden rozpoznaný vzorec
zoraď L podľa výšky vzorcov zostupne;
zoraď L podľa pozície zhora nadol;
pridaj prvý element do výsledného zoznamu segmentov V;
for all segment S zo zoznamu L okrem prvého prvku **do**
    Segment T := posledný element z V;
                                ▷ SpojSegmenty(X,Y) viď algoritmus 7
    Segment U := SpojSegmenty(T,S);
    if U ¡¿ NULL **then**
        pridaj S na koniec zoznamu V;
    end if
end for

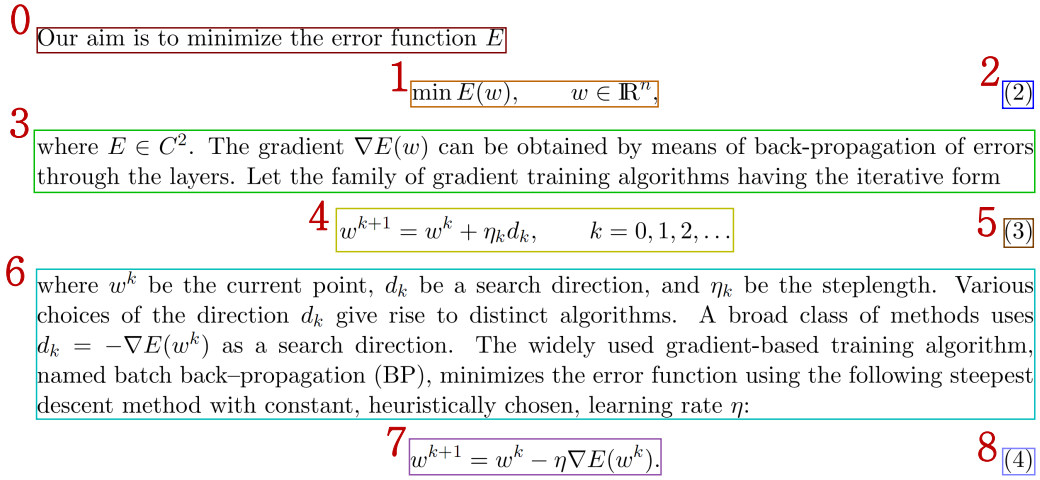
Nasledovný algoritmus dostane na vstupe dva segmenty. Ak sa tieto segmenty vertikálne prekrývajú, znamená to, že sa nachádzajú v tom istom riadku matematického textu. Také segmenty potom spojí a vráti NULL. Ak sa vstupné segmenty nenachádzajú v tom istom riadku, algoritmus vráti druhý zadaný segment.

Algorithm 7 SpojSegmenty

Input: segment X
Input: segment Y
  if X a Y sa vertikálne prelínajú **then**
    pridaj elementy segmentu Y k zoznamu elementov segmentu X; **return** NULL;
  else
    return Y;
  end if

4.7 Začlenenie vzorcov do štruktúry dokumentu

Pri rozpoznávaní štruktúry dokumentu je potrebné pracovať aj s rozpoznanými riadkami matematického textu. Podotýkame, že oblasti textu, kde sa nachádzajú matematické vzorce, musí užívateľ ručne označiť pred rozpoznávaním dokumentu podľa návodu popísanom v B.2.1. Každý rozpoznaný riadok matematického textu je reprezentovaný ako jeden odstavec a pri procese zoraďovania odstavcov podľa toho, ako by užívateľ daný text čítal, sa s nimi plnohodnotne pracuje. Algoritmus zoraďovania odstavcov predpokladá stĺpcovú štruktúru textu a je detailne popísaný v [5]. Na obrázku 4.4 je zobrazený príklad očíslovania odstavcov za predpokladu, že užívateľ pred spracovaním dokumentu príslušne označil vzorce.



Obr. 4.4: Začlenenie vzorcov do štruktúry dokumentu. Na obrázku je vidieť, že odstavce obsahujúce vzorce sú správne očíslované v takom poradí, v akom by ich užívateľ čítal.

4.8 Detaily implementácie

4.8.1 Manažment pravidiel

Na to, aby sme mohli rozpoznávať štruktúru, musíme zadefinovať pravidlá. Na to slúži aplikácia *RuleCreator*. Aplikácia umožňuje spravovať množinu pravidiel, čo zahŕňa pridávanie, odoberanie a úpravu pravidiel. Pravidlá je možné definovať veľmi jednoducho tak, že ich užívateľ nakreslí na plátno. Užívateľ môže vytvárať nové vrcholy, pridávať a odoberať hrany medzi vrcholmi a definovať vlastnosti vrcholov, hrán a pravidiel. Podrobný popis funkcionality tejto aplikácie sa nachádza v priloženej užívateľskej príručke.

4.8.2 Definované pravidlá

V práci definujeme viac ako 35 pravidiel, ktoré slúžia na rozpoznanie nasledovných štruktúr:

- binárne operátory s operandami
- zlomok
- druhá odmocnina
- k-ta odmocnina
- časť vzorca v okrúhlych zátvorkách
- horný index
- dolný index
- unárne mínus v okrúhlych zátvorkách
- integrál
- suma nenachádzajúca sa v zlomku
- produkt
- limita
- logaritmus
- sínus
- kosínus
- faktoriál
- symboly zložené z viacerých častí: $\leq, \geq, =, \cdots, i$

4.8.3 Podpora rozpoznávania vzorcov v aplikácii

OCR aplikácia môže pracovať aj bez podpory rozpoznávania vzorcov vďaka tomu, že rozpoznávanie vzorcov je voliteľný zásuvný modul. V prípade existencie viacerých modulov slúžiacich na rozpoznávanie vzorcov si môže užívateľ vybrať jeden z nich a nastaviť jeho vlastnosti. Štandardne má však k dispozícii len jeden takýto zásuvný modul, ktorého funkcionalitu popisujeme v práci.

Kapitola 5

Výsledky a príklady použitia

5.1 Úspešnosť klasifikátorov

V tejto časti sa venujeme porovnaniu úspešnosti klasifikátorov s rôznymi parametrami. Úspešnosť klasifikátorov vyhodnocujeme na testovacej množine dát. Tú tvorí 2862 symbolov vyextrahovaných z dokumentov naskenovaných v rozlíšení 300dpi. Počet symbolov v množine závisí od ich relatívneho počtu v skenovaných dokumentoch. Testovacia množina dát tak dobre zachytáva frekvenciu výskytov rôznych druhov symbolov.
V práci porovnávame úspešnosť základného klasifikátoru s rôznymi parametrami. Zaujímajú nás tie parametre, od ktorých najviac závisí úspešnosť základného klasifikátora. Konkrétne ide o použitú metódu extrakcie vlastností, rozmer výstupnej vrstvy Kohonenovej mapy a počet epôch pri jej trénovaní, kde epocha je predloženie všetkých vstupných dát mape. Okrem iného experimentálne zisťujeme prínos použitia detekcie hrán, kde namiesto samotného symbolu používame jeho obrysy. Na detekciu hrán používame Cannyho detektor hrán [19]. Všetky inštancie klasifikátora trénujeme na množine, ktorá obsahuje 273 rôznych symbolov spolu v približne 40000 výskytoch (počet výskytov rôznych druhov symbolov v databáze je rovnaký a zadaný počet inštancií nemusí byť deliteľný počtom druhov symbolov, preto uvádzame približnú veľkosť databázy). V nasledujúcich tabuľkách je uvedená úspešnosť klasifikátorov s konkrétnymi parametrami. V tabuľke 5.1 uvádzame úspešnosť klasifikátorov, ktoré nepoužívajú detekciu hrán symbolov. Názvy metód extrakcie korešpondujú s popísanými metódami v sekcii 3.3.5.

Z tabuľky 5.1 je vidieť, že schopnosť správneho rozpoznania symbolu a zároveň schopnosť generalizácie závisí od vhodného rozmiestnenia bodov extrakcie vlastností, viď klasifikátor s extrakciou vlastností v jednom bode, ktorý dopadol najhoršie. Celková úspešnosť základného klasifikátora závisí aj od parametrov Kohonenovej mapy. Pri výstupnej vrstve rozmerov 20×20 a dvojnásobnom počte epôch sa úspešnosť klasifikátora zvýšila v prípade extrakcie vlastností v deviatich bodoch takmer o takmer 3,5%. V prípade štyroch bodov však došlo k poklesu úspešnosti o viac ako 3,5%. Pri veľkosti výstupnej vrstvy 22×22 dosiahla najvyššiu úspešnosť metóda extrakcie vlastností v šiestich bodoch. Táto metóda ako

Tabuľka 5.1: Porovnanie úspešnosti základného klasifikátora s rôznymi parametrami bez detekcie hrán

Spôsob extrakcie	**Počet epôch**	**Veľkosť mapy**	**Počet chýb**	**Úspešnosť v %**
1 bod	1000	18×17	1362	52,41%
5 bodov	1000	18×17	713	75,09%
9 bodov	1000	18×17	636	77,78%
súčty	1000	18×17	634	77,85%
6 bodov	1000	18×17	615	78,51%
4 body	1000	18×17	584	79,59%
4 body	2000	20×20	689	75,93%
5 bodov	2000	20×20	607	78,79%
6 bodov	2000	20×20	553	80,68%
súčty	2000	20×20	539	81,17%
9 bodov	2000	20×20	538	81,20%
9 bodov	2000	22×22	592	79,32%
4 body	2000	22×22	560	80,43%
súčty	2000	22×22	542	81,06%
6 bodov	2000	22×22	509	82,22%

jediná vykazovala s menenými parametrami čoraz lepšie výsledky rozpoznávania.

V práci experimentálne testujeme úspešnosť rozpoznávania základného klasifikátora, ktorý dostane na vstup obrázok, kde sú zobrazené len hrany symbolu, t.j. prechody medzi oblasťami čiernych a bielych pixelov. Úspešnosť testujeme na klasifikátoroch s rovnakými parametrami tak, ako sú zobrazené v tabuľke 5.1. Z tabuľky 5.2 vidieť, že zmena vstupu sa pri niektorých klasifikátoroch prejavila pozitívne, pri niektorých však negatívne. Úspešnosti klasifikátorov sa pohybovali v rovnakých intervaloch, z čoho plynie, že použitie detekcie hrán je len alternatíva k štandardnému vstupu, ktorá markantne neprispieva k vyššej úspešnosti klasifikátorov.

Najvyššia úspešnosť nami natrénovaného základného klasifikátora činí $82,22\%$, čo sa môže výrazne prejaviť na úspešnosti rozpoznania štruktúry vzorcov. Na to, aby sme zvýšili úspešnosť rozpoznávania, používame rozšírený klasifikátor, ktorého charakter a spôsob rozpoznávania by to mal zaručiť. Časová náročnosť trénovania tohoto klasifikátora je veľká, a preto v tabuľke 5.3 uvádzame len niekoľko málo hodnôt. Stĺpec *Pomocný klasifikátor* obsahuje parametre klasifikátora používaného na vytvorenie chybových skupín. Ako pomocný klasifikátor používame natrénovaný základný klasifikátor. Ako klasifikátor chybových skupín používame SVM s jedným parametrom - metódou extrakcie vlastností. Rozšírený klasifikátor

trénujeme na novej databáze, ktorá obsahuje približne 40000 symbolov.

Tabuľka 5.2: Porovnanie úspešnosti základného klasifikátora s rôznymi parametrami s detekciou hrán

Spôsob extrakcie	**Počet epôch**	**Veľkosť mapy**	**Počet chýb**	**Úspešnosť v %**
5 bodov	1000	18×17	957	66,56%
4 body	1000	18×17	804	71,91%
súčty	1000	18×17	741	74,11%
6 bodov	1000	18×17	720	74,84%
9 bodov	1000	18×17	522	81,76%
5 bodov	2000	20×20	931	67,41%
9 bodov	2000	20×20	667	76,69%
súčty	2000	20×20	618	78,41%
4 body	2000	20×20	529	81,52%
6 bodov	2000	20×20	518	81,90%
súčty	2000	22×22	546	80,92%
9 bodov	2000	22×22	542	81,06%
6 bodov	2000	22×22	515	82,01%
4 body	2000	22×22	512	82,11%

Tabuľka 5.3: Porovnanie úspešnosti rozšíreného klasifikátora s rôznymi parametrami

Pomocný klasifikátor (počet epôch, rozmer výstupnej vrstvy Kohonenovej mapy, metóda extrakcie, použitie detekcie hrán	**Metóda extrakcie skupinový klasifikátor SVM**	**Počet chýb**	**Úspešnosť v %**
1000, 18×17, 9 bodov, nie	6 bodov	484	83,09%
2000, 22×22, 6 bodov, nie	9 bodov	456	84,07%
2000, 22×22, 4 body, áno	6 bodov	433	84,87%
2000, 22×22, 6 bodov, nie	6 bodov	385	86,55%
1000, 18×17, 6 bodov, nie	9 bodov	274	90,43%
1000, 18×17, 9 bodov, nie	9 bodov	193	93,26%

5.2 Rozpoznávanie štruktúry vzorcov

V tejto sekcii na niekoľkých príkladoch ukážeme funkčnosť prepisovacích pravidiel a z toho plynúci prevod vzorcov do lineárnej podoby, konkrétne do TEXu. Pravidlá prepisu môžeme rozdeliť do dvoch tried:

- pravidlá vytvárajúce zložené symboly, napríklad pravidlo vytvorenia znaku = z dvoch častí rozpoznaných ako −, a podobne.
- pravidlá vytvárajúce samotné časti vzorcov, napr. pravidlo na vytvorenie zlomku, pravidlo pre binárny operátory,...

Pravidlá vytvárajúce zložené symboly sú definované veľmi konkrétne, t.j. množina symbolov, ktoré môžu jednotlivé vrcholy grafu pravidla, obsahuje len niekoľko málo prvkov. Druhá trieda pravidiel musí naopak pokrývať celé kategórie symbolov (napr. kategória Arrow, ktorá obsahuje všetky šípky), prípadne vrcholy reprezentujúce pravidlá.
Pri tvorbe pravidiel využívame regulárne výrazy a fakt, že názov každého symbolu sa skladá z dvoch častí, a to názvy kategórie a identifikácie symbolu v rámci kategórie. Ďalej využívame zavedenú konvenciu pomenovania pravidiel - názov každého pravidla sa začína na *Rule*. To umožňuje vytvárať regulárne výrazy typu $\hat{}Rule|Arrow$, ktoré povolia výskyt akéhokoľvek pravidla alebo akejkoľvek šípky.

5.2.1 Úspešné rozpoznanie štruktúry

Nasledujúce obrázky sú príkladmi bezchybného rozpoznania štruktúry. Každý symbol v obrázku bol správne rozpoznaný, a podobne aj štruktúra vzorcov bola rozpoznaná bez chýb. Na obrázku 5.1 sa nachádza jednoduchý vzorec, ktorý predpokladá použitie pravidla prepisu na zlomok, binárny operátor, zátvorky a na horný index. Výsledná reprezentácia je nasledovná:

```
\left(\mathrm{1}+\frac{\mathrm{1}}{n}\right)^{\mathrm{3}n}
```

Pre kontrolu správnosti uvádzame aj opätovne vytvorený vzorec:

$$\left(1+\frac{1}{n}\right)^{3n}$$

$$\left(1+\frac{1}{n}\right)^{3n}$$

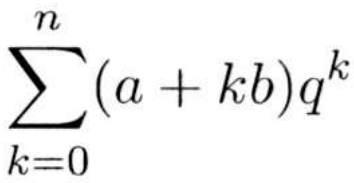
Obr. 5.1: Príklad rozpoznania štruktúry vzorcu č.1

Ďalší príklad na obrázku 5.2 obsahuje sumu, ktorý predpokladá použitie pravidla prepisu pre sumu. Štruktúra vzorcu bola rozpoznaná nasledovne:

```
\sum_{k=\mathrm{0}}^{n}\left(a+kb\right)q^{k}
```

Znova vytvorený vzorec má nasledujúcu podobu:

$$\sum_{k=0}^{n} \left(a + kb\right) q^{k}$$



Obr. 5.2: Príklad rozpoznania štruktúry vzorcu č.2

Obrázok 5.3 je príkladom rozpoznávania druhých odmocnín. Na rozpoznanie odmocnín sa používa pravidlo, ktoré využíva podmienku na polohu cieľového vrcholu typu *Inside*. viď sekcia 4.3.1. Výsledkom rozpoznávania vzorcu je nasledujúci text:

```
\sqrt{n}-\sqrt{n-\mathrm{1}}
```

Obr. 5.3: Príklad rozpoznania štruktúry vzorcu č.3

Na obrázku 5.4 sa nachádza vzorec s vnorenými odmocninami. Vo všeobecnosti algoritmus hľadania správnej štruktúry dokáže pomocou pravidla prepisu na odmocninu rozpoznať ľubovoľné zanorenie odmocnín. Výsledkom je nasledujúci text:

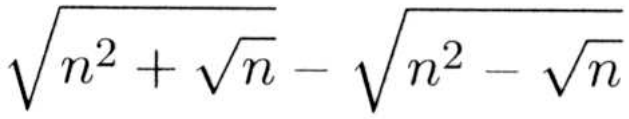
```
\sqrt{n^{\mathrm{2}} +\sqrt{n}} -\sqrt{n^{\mathrm{2}} -\sqrt{n}}
```

Pre jednoduchšie porovnanie výsledku s pôvodným obrázkom uvádzame vygenerovaný vzorec:

$$\sqrt{n^2 + \sqrt{n}} - \sqrt{n^2 - \sqrt{n}}$$

Na obrázku 5.5 uvádzame príklad rozpoznávania zlomkov s komplexným čitateľom a menovateľom. Na rozpoznanie zlomku existujú tri pravidlá prepisu, ktorých aplikácia je závislá od vizuálnej podoby zlomku, konkrétne:

- zlomok, ktorého šírka čitateľa je približne rovnaká ako šírka zlomkovej čiary a šírka menovateľa je znateľne menšia



Obr. 5.4: Príklad rozpoznania štruktúry vzorcu č.4

- zlomok, ktorého šírka menovateľa je približne rovnaká ako šírka zlomkovej čiary a šírka čitateľa je znateľne menšia
- zlomok, ktorého šírka čitateľa aj menovateľa je približne rovnaká ako šírka zlomkovej čiary

Dôvodom používania troch pravidiel je zamedzenie aplikácie pravidla zlomku na časti čitateľa, prípadne menovateľa. Výstup rozpoznávania vzorca je nasledovný:

```
\frac{n^{\mathrm{3}}+\mathrm{2}n^{\mathrm{2}}-\mathrm{4}n}
{n^{\mathrm{2}}-\sqrt{n}+\mathrm{1}}
```

Znova vygenerovaný vzorec má teda nasledujúcu podobu zhodujúcu sa so vstupom:

$$\frac{n^3 + 2n^2 - 4n}{n^2 - \sqrt{n} + 1}$$

$$\frac{n^3 + 2n^2 - 4n}{n^2 - \sqrt{n} + 1}$$

Obr. 5.5: Príklad rozpoznania štruktúry vzorcu č.5

Ďalšie príklady, ktoré uvádzame na obrázku 5.8 a 5.9, zobrazujú rozpoznanie vzorcov obsahujúcich matematické funkcie. Pravidlá rozpoznávajúce funkcie sú dominatné, a teda sú aplikované vždy ako prvé, čo pozitívne ovplyvňuje rýchlosť rozpoznávania.

$$\lim_{n \to \infty} \frac{n!}{n^n}$$

Obr. 5.6: Príklad rozpoznania štruktúry vzorcu č.6

Výstupy rozpoznávania sú nasledovné:

$$\frac{2b \log b}{1 + \log b + \log \log b}$$

Obr. 5.7: Príklad rozpoznania štruktúry vzorcu č.7

```
\frac{\mathrm{2}b\log b}{\mathrm{1}+\log b+\log\log b}
```

```
\lim{n\rightarrow\infty}\frac{n!}{n^{n}}
```

Opätovné vygenerovanie výsledkov má nasledujúcu podobu:

$$\lim_{n \to \infty} \frac{n!}{n^n}$$

$$\frac{2b \log b}{1 + \log b + \log \log b}$$

5.2.2 Neúspešné rozpoznanie štruktúry

Neúspešné rozpoznanie štruktúry vzorcov je spôsobené nasledovnými chybami, ktoré môžu pri procese rozpoznávania nastať:

- nesprávne rozpoznanie symbolu
- aplikácia nevhodného pravidla
- odstránenie symbolu, ktorý je považovaný za šum
- rozpoznávanie šumu ako symbolu[20].

Vstup na obrázku 5.8 obsahuje na ľavej strane rovnosti znak a s dolným indexom $n + 1$. Pri rozpoznávaní však nedošlo k detegovaniu dolného indexu, a tak je výsledok nasledovný:

```
an+\mathrm{1} =\frac{a_{n} +\mathrm{3}}{\mathrm{4}}
```

Pre porovnanie vstupu a výstupu uvádzame vygenerovaný vzorec:

$$an + 1 = \frac{a_n + 3}{4}$$

Obrázok 5.9 je ďalším príkladom neúspešného rozpoznania štruktúry. V tomto prípade je chyba spôsobená nesprávnym rozpoznaním symbolu (symbol 1 bol rozpoznaný ako symbol operátoru), čím došlo k nemožnosti aplikovať pravidlo na binárny operátor. Pravidlo na binárny operátor neumožňuje, aby bol operand symbol reprezentujúci operátor. Výsledok je tak nasledovný:

$$a_{n+1} = \frac{a_n + 3}{4}$$

Obr. 5.8: Príklad rozpoznania štruktúry vzorcu č.6

$$\frac{\Gamma(\alpha + n - q - 1)}{\Gamma(n + \alpha)}$$

Obr. 5.9: Príklad rozpoznania štruktúry vzorcu č.8

```
\frac{\mathrm{\Gamma}\left(\alpha +n-q\right)}
{\mathrm{\Gamma}\left(n+\alpha\right)} -\oplus
```

Pre porovnanie vstupu a výstupu uvádzame vygenerovaný vzorec:

$$\frac{\Gamma\left(\alpha + n - q\right)}{\Gamma\left(n + \alpha\right)} - \oplus$$

Všeobecným problémom používania prepisovacích pravidiel je zložitá úprava ich vlastností. Pri väčšom počte pravidiel môže úprava vlastností jedného pravidla viesť k úpravám na ďalších pravidlách. S každou úpravou vlastností je spojené testovanie jej funkčnosti náročné na čas.

5.2.3 Celková úspešnosť rozpoznávania štruktúry vzorcu

V našej práci testujeme úspešnosť rozpoznávania štruktúry vzorcu na množine obsahujúcej 90 vzorcov. Za úspešné rozpoznanie štruktúry považujeme také, pri ktorom opätovne vygenerovaný vzorec je vizuálne zhodný so vstupným vzorcom, t.j. pozície a vzájomné vzťahy medzi symbolmi sú zhodné, pričom samotné rozpoznané symboly sa nemusia zhodovať so vstupnými. Aplikácia niektorých pravidiel je však podmienená výskytom konkrétnych rozpoznaných symbolov, čo pri rozpoznávaní vzorcov častokrát znamená, že symboly musia byť vo veľkej miere rozpoznané správne.
Z celkového počtu 90 vzorcov došlo k úspešnému rozpoznaniu štruktúry v 81 prípadoch, čo tvorí 90%. Pri zvyšných prípadoch došlo k chybe pri určovaní horného alebo dolného indexu symbolov.

5.2.4 Celková úspešnosť prevodu

Celková úspešnosť prevodu závisí nielen od kvality predspracovania obrázku, úspešnosti rozpoznávania symbolov, ale z veľkej časti aj od množstva a kvality definovaných pravidiel. Pod pojmom kvalita pravidla rozumieme jeho schopnosť aplikácie na rôznych vstupoch. Vzhľadom k tomu, že naše testovacie dáta boli naskenované s rozlíšením 300dpi, došlo pri predspracovaní v niektorých prípadoch ku zlúčeniu symbolov nachádzajúcich sa v tesnej blízkosti. Nesprávne rozpoznanie takého zlúčeného symbolu sme nepovažovali za chybu. Z celkového počtu 90 vzorcov bolo bezchybne rozpoznaných 55 vzorcov, jednu chybu rozpoznania symbolu obsahovalo 20 vzorcov a dve chyby rozpoznania symbolu obsahovali 4 vzorce. Zvyšné vzorce obsahovali viac chýb, a to buď chýb rozpoznania symbolu alebo chýb rozpoznania štruktúry. Celková úspešnosť rozpoznávania vzorcov, ktoré obsahujú najviac dve chyby v rozpoznaní symbolu, teda dosiahla 87,78%.

5.2.5 Rýchlosť rozpoznávania

Nevýhodou použitia nami implementovaného spôsobu zisťovania štruktúry je jeho rýchlosť. Nami použitá heuristika, popísaná v 4.4, je vo väčšine prípadov aplikovaná a nemusia byť vyskúšané všetky možnosti prepisu. Pri zložitejších vzorcoch alebo pri obrázkoch, ktoré obsahujú viac riadkov vzorcov, môže rozpoznávanie štruktúry trvať aj niekoľko desiatok sekúnd. V takých prípadoch je vhodné rozdeliť vzorce na viac častí, prípadne riadkov, a riešiť ich samostatne. Za cenu rýchlosti rozpoznávania však užívateľovi poskytujeme možnosť rozpoznávať vlastné štruktúry.

Kapitola 6

Záver

Cieľom práce bolo rozšíriť OCR systém o prevod matematických vzorcov do editovateľnej podoby. OCR systém sme rozšírili o možnosť označenia oblasti matematických vzorcov užívateľom. Zamerali sme sa na vzorce, ktoré sú samostatne vyčlenené v texte, nakoľko takéto vzorce sú väčšinou komplexné. Matematické vzorce nachádzajúce sa priamo v texte by mali byť naopak veľmi jednoduché a v malom počte. Takéto vzorce v texte nepredpokladáme , t.j. budú reprezentované ako štandardný text. Z uvedených dôvodov nepovažujeme ignorovanie takýchto vzorcov za veľký nedostatok.
Samotný prevod je možné rozdeliť na dve časti, a to rozpoznávanie symbolov a rozpoznávanie štruktúry. Nami natrénovaný klasifikátor symbolov má úspešnosť 93,26%. Tento výsledok považujeme za vynikajúci aj napriek tomu, že existujú klasifikátory s vyššou úspešnosťou. Na rozpoznanie štruktúry sme použili metódu používajúcu pravidlá prepisu. Výsledkom našej práce je množina precízne definovaných pravidiel tak, aby správne rozpoznávali štruktúru s čo najväčšou presnosťou. Nami vytvorená množina pravidiel slúži na rozpoznávanie menej komplexných vzorcov, avšak použitá metóda, spolu s vytvorenou sadou aplikácií, umožňuje užívateľovi jednoducho pridávať nové pravidlá, pomocou ktorých je možné rozšíriť pokrytie rôznych matematických konštrukcií, prípadne prispôsobiť pravidlá na aplikáciu na matematických vzorcoch vopred známych typov. Úspešnosť rozpoznávania štruktúry menej komplexných vzorcov dosiahla 90%. Celková úspešnosť rozpoznávania vzorcov, ktoré obsahujú najviac dve chyby v rozpoznaní symbolu, dosiahla 87,78%, čo považujeme za dobrý výsledok. Tento cieľ práce preto považujeme za splnený.
Druhým cieľom práce bolo vytvorenie sady aplikácií, ktoré umožnia do veľkej miery upraviť proces prevodu. Výsledkom našej práce je sada štyroch pomocných aplikácií, ktoré slúžia na vytváranie a trénovanie klasifikátorov, vytváranie databázy symbolov, na ktorej budú klasifikátory trénované, a na vytváranie a úpravu pravidiel prepisu. Aplikácie boli navrhnuté tak, aby boli jednoducho a intuitívne ovládateľné, a pritom umožňovali komplexné úpravy prevodu. Z tohto dôvodu považujeme aj druhý cieľ za splnený.
Výsledkom našej práce je program, ktorý umožňuje okrem klasického rozpozná-

vania textu aj rozpoznávanie vzorcov, s tým, že užívateľ má možnosť namiesto vzorcov rozpoznávať iné, vlastné definované štruktúry. Takúto schopnosť nemajú ani najlepšie komerčné OCR programy súčasnosti.
V budúcej práci by sme chceli dosiahnuť vyššiu úspešnosť rozpoznávania symbolov a rozšíriť množinu pravidiel tak, aby pokrývala čo najviac matematických konštrukcií. Taktiež by sme chceli implementovať rozpoznávanie matíc nachádzajúcich sa vo vzorcoch. Samotný OCR systém by sme chceli rozšíriť o automatické rozpoznávanie pozícií vzorcov v texte, aby užívateľ nemusel označovať oblasti so vzorcami ručne. Po dosiahnutí týchto cieľov by sme chceli rozšíriť výstup o ďalšie formáty.

Literatúra

[1] Francisco Álvaro, Joan Andreu Sánchez: *Comparing Several Techniques for Offline Recognition of Printed Mathematical Symbols* 20th International Conference on Pattern Recognition (ICPR), Istanbul, 2010

[2] Masakazu Suzuki, Fumikazu Tamari, Ryoji Fukuda, Seiichi Uchida, Toshihiro Kanahori: *Infty: an integrated ocr system for matematical documents* Proceedings of the 2003 ACM symposium on Document engineering, Grenoble, 2003

[3] Masakazu Suzuki, Seiichi Uchida, Akihiro Nomura *A Ground-Truthed Mathematical Character and Symbol Image Database* Proceedings of the Eighth International Conference on Document Analysis and Recognition, IEEE Computer Society Washington, DC, USA, 2005

[4] Rafael C. Gonzales, Richard E. Woods: *Digital Image Processing*, Pearson Prentice Hall, Upper Saddle River, 2008

[5] Matej Klaučo: *Rozpoznávanie textu*, Bakalárska práca, Praha, 2009

[6] Fu Chang, Chun-Jen Chen, and Chi-Jen Lu: *A Linear-Time Component-Labeling Algorithm Using Contour Tracing Technique*, Elsevier Science Inc., New York, 2004

[7] Roberto Paredes, Enrique Vidal: *Learning Weighted Metrics to Minimize Nearest-Neighbor Classification Error* IEEE Transaction on Pattern Analysis and Machine Intelligence Volume 28 Number 7, IEEE Computer Society Washington, DC, USA, July 2006

[8] Utpal Garain, Balarko Chaudhuri: *Recognition of online handwritten mathematical expressions*, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 34, Volume 34, Issue 6, 2366-2376, New York, NY, USA, December 2004

[9] Alejandro Toselli, Alfons Juan, Enrique Vidal: *Spontaneous handwriting recognition and classification* Proceedings of the 17th International Conference on Pattern Recognition, 433-436, Cambridge, UK, August 2004

[10] Xue-Dong Tian, Hai-Yan Li, Xin-Fu Li, Li-Ping Zhang: *Research on Symbol Recognition for Mathematical Expressions* First International Conference on Innovative Computing, Information and Control, Beijing, 2006

[11] NeuronDotNet Library
http://neurondotnet.freehostia.com

[12] Tuevo Kohonen: *Self-Organizing Maps, Third Edition*, Springer, 2001

[13] Christopher Malon, Masakazu Suzuki, Seiichi Uchida: *Support Vector Machines for Mathematical Symbol Recognition* Pattern Recognition Letters Volume 29 Issue 9, Elsevier Science Inc., New York, 2008

[14] Konstantinos Koutroumbas, Sergios Theodoridis: *Pattern Recognition, Third Edition* Academic Press, London, 2006

[15] Xiaoqing Ding, Hua Wang: *Multi-Font Printed Tibetan OCR* Digital Document Processing: Major Directions and Recent Advances, Springer, 2006

[16] Amar Raja, Matthew Rayner, Alan Sexton, Volker Sorge: *Towards a Parser for Mathematical Formula Recognition* Mathematical Knowledge Management, 139-151, Springer, 2006

[17] Dorothea Blostein, James R. Cordy, Richard Zanibbi: *Recognizing Mathematical Expressions Using Tree Transformation* IEEE Transaction on Pattern Analysis and Machine Intelligence Volume 24 Number 11, 1455 - 1467, IEEE Computer Society Washington, DC, USA, November 2002

[18] Andrew W. Moore: *An introductory tutorial on kd-trees* Technical Report No. 209, University of Cambridge, 1991

[19] John Canny: *A computational approach to edge detection* IEEE Transactions on Pattern Analysis and Machine Intelligence archive Volume 8 Issue 6, IEEE Computer Society Washington, DC, USA, 1986

[20] Akio Fujiyoshi, Masakazu Suzuki, Seiichi Uchida: *Syntactic detection and correction of misrecognitions in mathematical OCR* International Conference on Document Analysis and Recognition - ICDAR , 1360-1364, Barcelona, 2009

[21] Matthew Alastair Johnson: *SVM.NET*
http://www.matthewajohnson.org/software/svm.html

Dodatok A

Obsah CD

Priložené CD obsahuje:

- *dokumentacia* - adresár s vygenerovanou dokumentáciou pomocou programu Doxygen
- *klasifikatory* - adresár obsahuje súbory s natrénovanými klasifikátormi
- *programy* - adresár obsahujúci archív *programy.zip* s vytvorenými aplikáciami
- *symboly* - adresár, ktorý obsahuje testovaciu množinu symbolov a databázu InftyCDB-1
- *text prace* - tento text vo formáte PS, DVI a PDF
- *videa* - adresár s videoukážkami rozpoznávania vzorcov
- *vzorce* - adresár, ktorý obsahuje testovaciu množinu vzorcov
- *zdrojove kody* - adresár so všetkými zdrojovými kódmi
- súbor *info.txt* - kontakt na autora

Dodatok B

Užívateľská príručka

B.1 Inštalácia balíčka programov

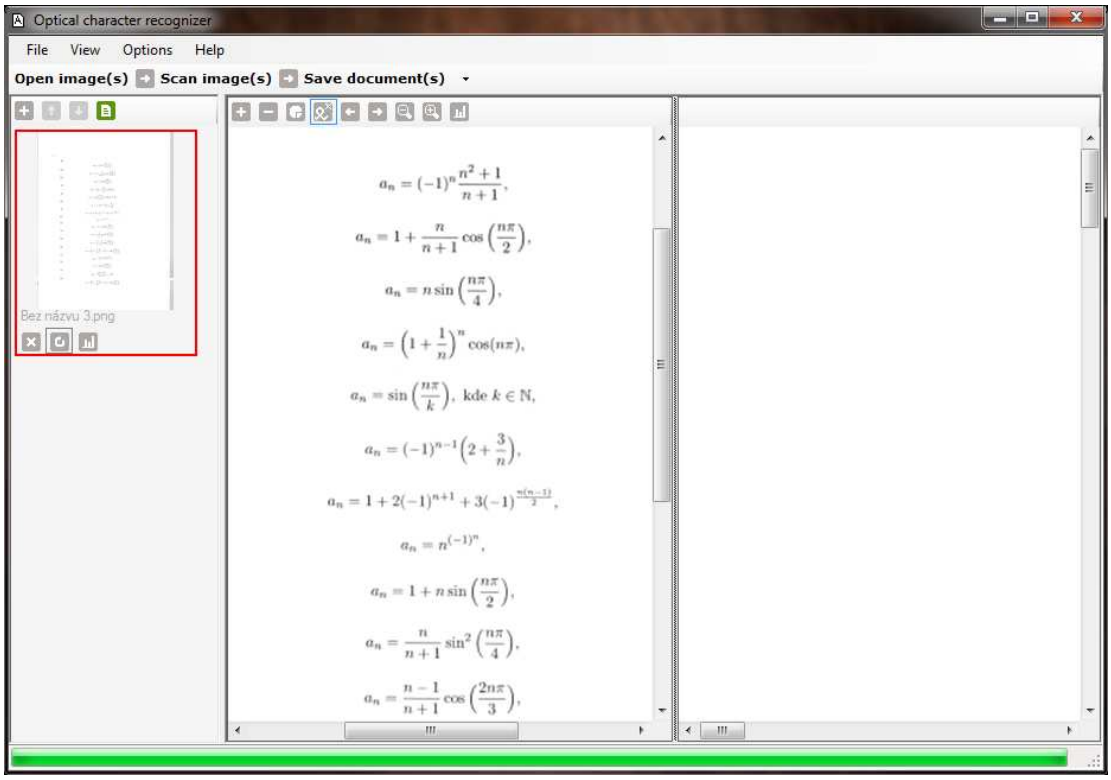
Inštalácia balíčka je veľmi jednoduchá. Stačí rozbaliť archív s programami na ľubovoľné miesto na disku. Programy vyžadujú Microsoft .NET Framework 4.0. Ak tento framework nie je nainštalovaný, je potrebné ho stiahnuť a nainštalovať. Inštalačný súbor je možné nájsť na stránkach spoločnosti Microsoft.
Doporučené požiadavky na beh programov sú nasledovné:

- procesor: minimálne 2GHz
- RAM: minimálne 2GB
- HDD: minimálne 2GB voľného priestoru
- OS: Windows Vista alebo Windows 7

B.2 Program Optical Character Recognizer

Program Optical Character Recognizer slúži na prevod skenovaných obrázkov s textom do editovateľnej podoby. Štandardne umožňuje uložiť výsledný text do textového, HTML, alebo TEX súboru. Pri ukladaní do textového a TEX súboru sú ignorované oblasti označené ako obrázok. Pri ukladaní do textového súboru sú ignorované aj oblasti označené ako matematický text. Program spustíme pomocou súboru *OCR.exe*, ktorý sa nachádza v adresári *Optical Character Recognizer*. Na obrázku B.1 je zobrazené hlavné okno súboru s načítaným obrázkom obsahujúcim matematické vzorce. Hlavné okno sa skladá z niekoľkých častí. V hornej časti sa nachádza hlavné menu. Pod hlavným menu je menu pre konverziu. V ľavej časti okna sa nachádza panel pre prácu s dávkou. Stred okna slúži na zobrazenie aktuálne vyznačeného obrázka a informácií, ktoré budú z neho počas konverzie vyextrahované. V pravej časti sa zobrazuje výsledný text približne v takom rozložení, v akom sa nachádza v rozpoznávanom obrázku. Podrobné ovládanie tejto

aplikácie je popísané v [5]. V tejto príručke popíšeme len tie časti aplikácie, ktoré sa týkajú novej funkcionality.



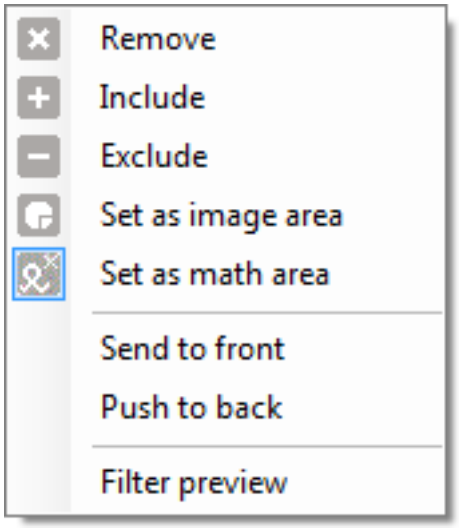
Obr. B.1: Hlavné okno programu Optical Character Recognizer

B.2.1 Označovanie rôznych typov oblastí v dokumente

Upravená OCR aplikácia umožňuje rozpoznávanie matematických vzorcov. Tie však musia byť užívateľom označené novým typom oblasti - tzv. *matematickou oblasťou*. Každá oblasť môže obsahovať jeden alebo viac vzorcov a jeden alebo viac riadkov textu. Pri očakávanom procese spracovania danej oblasti by mal byť označený text rozdelený do toľkých riadkov, koľko riadkov matematických vzorcov bolo obsiahnutých v označenej matematickej oblasti. Po vytvorení oblasti kliknutím a ťahom myši je možné zobraziť kontextové menu vyznačenej oblasti, viď obrázok B.2. Oproti kontextovému menu popísanom v B.1, obsahuje toto menu navyše aj položku *Set as math area*. Vybraním tejto položky bude zmenený typ označenej oblasti na matematickú oblasť.

B.2.2 Nastavenie rozpoznávania matematického textu

Rozpoznávanie matematických vzorcov je voliteľný doplnok aplikácie OCR. Užívateľ ho nemusí používať, alebo, v prípade viacerých implementácií rozpoznávania



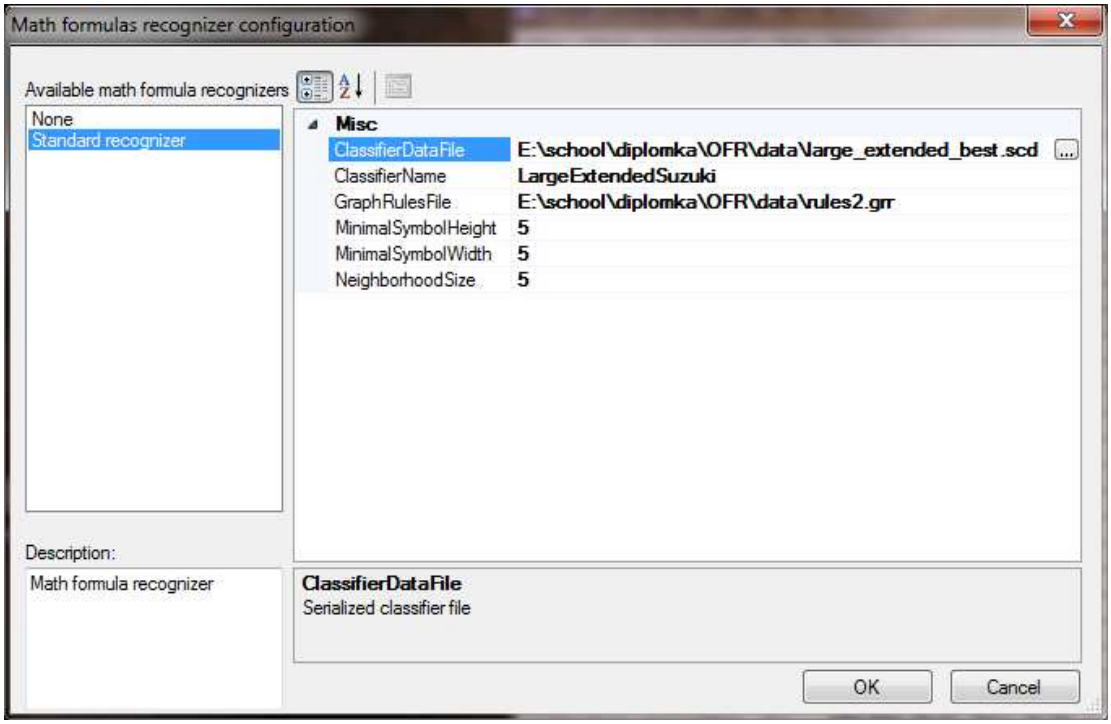
Obr. B.2: Kontextové menu pre vyznačenú oblasť

matematických vzorcov, si môže užívateľ vybrať jednu z nich a upraviť jej vlastnosti. Na správu týchto doplnkov slúži formulár na obrázku B.3. Užívateľ zobrazí formulár cez *Options* → *Math formula recognizers*. V ľavej časti formulára sa nachádza zoznam dostupných rozpoznávačov matematického textu. Každý rozpoznávač je samostatný plugin, umiestnený v adresári *MathRecognizers*, ktorý sa nachádza v hlavnom adresári aplikácie. Zoznam obsahuje položku *None*. Označením tejto položky sa nebudú rozpoznávať matematické oblasti. Pri označení inej položky zoznamu sa zobrazia v pravej časti formuláru editovateľné vlastnosti zvoleného pluginu. OCR aplikácia poskytuje jeden naimplementovaný plugin. Jeho vlastnosti popíšeme v nasledujúcej sekcii.

B.2.3 Predvolený rozpoznávač matematického textu

Predvolený rozpoznávač matematického textu umožňuje upraviť nasledovné vlastnosti:

- ClassifierName - názov klasifikátora, ktorý má byť použitý pri rozpoznávaní symbolu
- ClassifierDataFile - cesta k natrénovanému klasifikátoru, ktorý je uložený v súbore s príponou *.scd*
- GraphRulesFile - cesta k súboru s pravidlami prepisu grafu. Súbor má príponu *.grr*
- MinimalSymbolHeight - minimálna výška symbolu, aby mohol byť symbol rozpoznávaný



Obr. B.3: Okno s nastaveniami dostupných rozpoznávačov matematického textu

- MinimalSymbolWidth - minimálna šírka symbolu, aby mohol byť symbol rozpoznávaný. Symboly s rozmerom menším ako *MinimalSymbolHeight* × *MinimalSymbolWidth* px budú ignorované.
- NeighborhoodSize - počet vyhľadávaných najbližších susedov každého symbolu. Optimálna hodnota je 5, ale pri niektorých grafoch môže byť potrebné zvýšiť túto hodnotu na 6, prípadne 7. S väčším počtom susedov však rastie doba spracovania matematických textov.

B.2.4 Prehliadanie rozpoznanej štruktúry vzorcov

Po tom, ako aplikácia rozpoznala celkovú štruktúru dokumentu, môže užívateľ, okrem štandardnej práce s rozpoznaným textom, prehliadať štruktúru rozpoznaných riadkov matematického textu. Každý riadok je vykreslený v ľavej časti aplikácie jedným zeleným obdĺžnikom. Kliknutím myši na tento obdĺžnik sa zobrazí okno obsahujúce popis štruktúry v textovej podobe. Aplikované pravidlá sú rozpísané, to znamená, že užívateľ môže vidieť, ktorý vrchol grafu pravidla odpovedá ktorej časti štruktúry. Obsah pravidla je odsadený tabulátorom.

B.2.5 Ukladanie výsledného dokumentu

Výsledný dokument je možné uložiť do troch formátov - HTML, TXT a TEX. Jediný formát, ktorý podporuje ukladanie matematických vzorcov je posledný

zmienený. V nastaveniach ukladača do TEXu (*Options* → *Savers*) je možné upraviť nasledovné vlastnosti:

- AvailableTeXStrings - cesta k súboru s TEXovými reprezentáciami symbolov. Formát súboru je nasledovný:
  kategória ; *číslo kategórie* ; *hexadecimálny kód symbolu*; *názov symbolu* ; *textová reprezentácia symbolu*
- TeXHeaderFile - cesta k súboru, ktorého obsah bude vypísaný na začiatok výsledného TEXového dokumentu
- TeXFooterFile - cesta k súboru, ktorého obsah bude vypísaný na koniec výsledného TEXového dokumentu
- BeforeFormula - reťazec zapísaný do súboru pred každým rozpoznaným vzorcom
- AfterFormula - reťazec zapísaný do súboru za každým rozpoznaným vzorcom

B.3 Program MathSymbolTrainer

Program MathSymbolTrainer slúži na vytváranie a trénovanie a testovanie klasifikátorov symbolov a na vytváranie trénovacej databázy. Hlavné okno aplikácie, viď obrázok B.4, sa skladá z troch častí. Na ľavej strane sa zobrazuje zoznam symbolov databázy. Pod zoznamom je zobrazený aktuálne označený symbol zoznamu. Stredná časť slúži na zobrazenie informácií o databáze a na úpravu vlastností novovytvoreného klasifikátora. V ľavej časti sa nachádza konzola na výpis informácii pri testovaní a tlačidlá na spustenie trénovania a testovania klasifikátora.

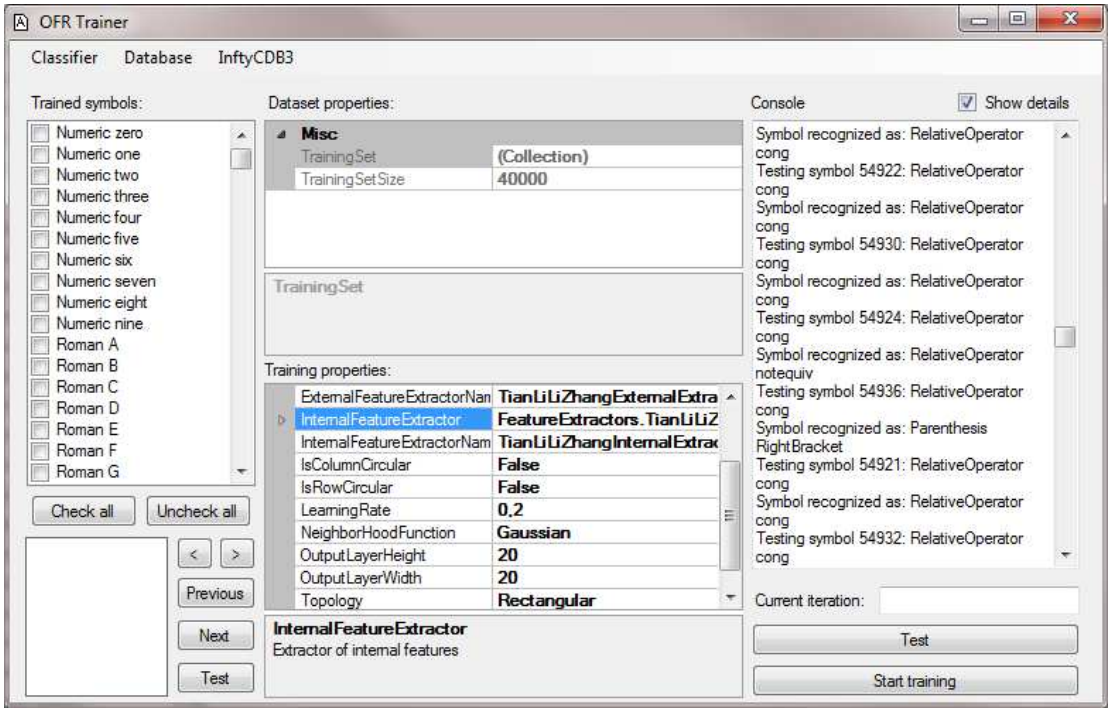
B.3.1 Vytvorenie databázy

Užívateľ môže vytvoriť databázu trénovaných symbolov z databázy typu InftyCDB3. Pomocou *Database* → *New from InftyCDB3 database* načíta súbor odvodený z databázy InftyCDB3. Formát súboru je nasledovný:

hexadecimálny kód symbolu medzera *kategória* medzera *reťazcová reprezentácia znaku* medzera *výška znaku* medzera *šírka znaku* medzera *bytový zápis znaku postupne po riadkoch*,

kde bytový zapís znamená zápis pixelov (1 ako čierny, 0 ako biely) ako 8 po sebe nasledujúcich bitov. Chýbajúce bity do násobku 8 v jednom riadku sú reprezentované ako nuly.

V strednej časti hlavného okna sa po načítaní databázy typu InftyCDB3 zobrazia upraviteľné vlastnosti vytváranej databázy. Užívateľ môže nastaviť jej



Obr. B.4: Hlavné okno programu Math Symbol Trainer

celkovú veľkosť, t.j. počet symbolov, koľko má databáza obsahovať. Užívateľ môže zvoliť, ktoré symboly sa majú v databáze nachádzať, zaškrtnutím príslušných symbolov v zozname symbolov na ľavej strane hlavného okna aplikácie. Tlačidlá *Select all* a *Deselect all* slúžia na zjednodušenie práce pri vyberaní symbolov. Po zadaní veľkosti databázy a výbere symbolov je možné uložiť novú databázu pomocou *Database* → *Save*.

B.3.2 Vytvorenie databázy typu InftyCDB3 z databázy typu InftyCDB1

Program MathSymbolTrainer umožňuje vytváranie databáz typu InftyCDB3 z databáz typu InftyCDB1. Databázy typu InftyCDB1, detailne popísaná v [3], sa skladá z dvoch častí:

- popisnej časti - túto časť tvorí súbor, ktorý obsahuje informácie o polohách a druhoch symbolov v obsahovej časti.
- obsahovaj časti - táto časť je tvorená naskenovanými dokumentami, ktoré obsahujú matematické symboly.

Zvolením *InftyCDB3* → *Convert from InftyCDB1* sa zobrazí okno na konverziu databáz, viď obrázok B.5. Užívateľ inicializuje konvertor zvolením *Converter* → *Initialize*. Zobrazí sa okno na inicializáciu konvertoru. V ňom užívateľ zadá cestu

k súboru so všetkými dostupnými znakmi, cestu k popisnej časti databázy InftyCDB1 a cestu ku koreňovému adresáru obsahovej časti databázy InftyCDB1. Po inicializácii užívateľ vyberie zo zoznamu symboly, ktoré z ktorých chce vytvoriť databázu InftyCDB3. Pomocou *Converter* → *Save as CDB3 database* spustí proces konverzie databáz.

Obr. B.5: Okno konvertoru databázy typu InftyCDB1 na typ InftyCDB3

B.3.3 Vytvorenie databázy typu InftyCDB3 z adresárovej štruktúry

Aplikácia MathSymbolTrainer umožňuje vytvorenie databázy typu InftyCDB3 z presne definovanej adresárovej štruktúry, ktorá obsahuje obrázky s jednotlivými symbolmi. Každý obrázok obsahujúci symbol musí byť čiernobiely, formátu BMP s indexovanými farbami a formátom dát 1 bit na 1 pixel.
Koreňový adresár štruktúry musí obsahovať podadresáre, ktorých názvy sú hexadecimálne kódy symbolov. V každom takom podadresári sa musia nachádzať obrázky obsahujúce daný symbol, s formátom popísaným vyššie.

Zvolením *InftyCDB3* → *Create from image directory* v aplikácii sa zobrazí formulár na zadanie cesty ku koreňovému adresáru štruktúry, viď obrázok B.6. Po tom ako užívateľ zadá cestu a zvolí *Create* sa zobrazí formulár na uloženie vytváranej databázy. Po zadaní mena a cesty sa začne spracovávať štruktúra.

Obr. B.6: Formulár na vytvorenie databázy typu InftyCDB3 z adresárovej štruktúry

B.3.4 Vytvorenie klasifikátora

Užívateľ začne proces vytvárania klasifkikátora zvolením *Classifier* → *New*. Užívateľ následne vyberie trénovaciu databázu. Po nej vyberie dynamickú knižnicu so zvoleným typom klasifikátora. Následne sa zobrazí okno s vlastnosťami klasifikátora, ktoré môžu byť vopred inicializované. Ktoré vlastnosti musia byť týmto spôsobom inicializované závisí od typu klasifikátora takto:

- pre klasifikátor typu TianLiLiZhang, t.j. základného klasifikátora:

  1. InternalFeatureExtractorName - meno extraktoru vnútorných vlastností symbolu.
  2. ExternalFeatureExtractorName - meno extraktoru vonkajších vlastností symbolu.

  Tieto extraktory budú inicializované a ich vlastnosti je možné nastaviť v hlavnom okne aplikácie.

- pre klasifikátor typu ExtendedSuzuki, t.j. rozšíreného klasifikátora:

  1. SymbolInGroupClassifierName - meno klasifikátora slúžiaceho na klasifikáciu symbolov v rámci chybových skupín.
  2. SymbolToMatrixClassifierName - meno prvotného klasifikátora slúžiaceho na klasifikáciu symbolov a vytvorenie chybovej matice.
  3. SymbolToMatrixClassiferDataFile - cesta k súboru s uloženým natrénovaným prvotným klasifikátorom.

Po inicializácii tohto klasifikátora bude v niektorých prípadoch nutné nastaviť parametre pomocného klasifikátora používaného v rámci chybových skupín, konkrétne ak ide o klasifikátor typu MathSymbol. V tom prípade po zadaní mena extraktoru vlastností symbolu sa vytvorí inštancia extraktoru s daným menom. Vlastnosti vytvoreného extraktoru je možné ďalej upravovať.

- pre klasifikátor typu MathSymbol (klasifikácia pomocou Support Vector Machines):

  1. FeatureExtractorName - meno extraktoru vlastností symbolu

Po inicializácii klasifikátora môže užívateľ dodatočne upraviť jeho vlastnosti, prípadne parametre používaných extraktorov. Dodatočná úprava pred trénovaním sa týka tých vlastností, ktoré sa nenachádzajú vo vyššie zmienených zoznamoch. Po trénovaní je zmena akýchkoľvek vlastností nežiadúca a môže viesť k chybe programu.

B.3.5 Úprava vlastností klasifikátorov

V tejto časti popíšeme vlastnosti klasifikátorv, ktoré je možné upraviť po inicializácii a pred začatím trénovania.

Základný klasifikátor

- EDDConstant - konštanta používaná pri klasifikácii v rámci jednej skupiny vybranej pomocou Kohonenovej mapy.
- EpochCount - počet epôch vykonaných pri učení Kohonenovej mapy
- paremetre extraktoru vnútorných vlastností
- parametre extraktoru vonkajších vlastností
- LearningRate - rýchlosť učenia sa Kohonenovej mapy
- Topology - topológia Kohonenovej mapy - pravouhlá alebo šesťuholníková
- OutputLayerWidth - šírka výstupnej vrstvy Kohonenovej mapy
- OutputLayerHeight - výška výstupnej vrstvy Kohonenovej mapy

Rozšírený klasifikátor

- DestinationDirectory - cieľový adresár kam majú byť vytvorené súbory pre program DistrubutedPartManager

- Distribute - indikátor použitia distribuovaného trénovania. Ak je nastavený cieľový adresár a hodnota tohto parametru je true, tak po začatí trénovania budú vytvorené pomocné súbory do cieľového adresára.
- DistributeTo - približný počet súborov, ktoré budú vytvorené za účelom distribúcie trénovania. Každý súbor bude obsahovať približne rovnaký počet nenatrénovaných klasifikátorov chybových skupín.
- vlastnosti klasifikátora používaného v rámci chybových skupín - klasifikátory chybových skupín budú klonované a budú mať rovnaké vlastnosti ako upravovaný klasifikátor

SVM klasifikátor typu MathSymbol

- parametre extraktoru vlastností

B.3.6 Parametre extraktorov vlastností

Niektoré extraktory vlastností vyžadujú po svojej inicializácii aj dodatočné upravenie parametrov extrakcie. V tejto časti popíšeme parametre jednotlivých extraktorov.

TianLiLiZhangInternalExtractor

Tento extraktor slúži na extrakciu vnútorných vlastností symbolu a štandardne je používaný klasifikátorom typu *TianLiLiZhang* (t.j. základným klasifikátorom, viď. 3.3). Užívateľ môže nastaviť nasledovné parametre:

- ApplyEdgeDetector - indikátor použitia hrán symbolu namiesto samotného symbolu
- SubblockFeatureType - metóda extrakcie vlastností:
  - TianLiLiZhang - viacnásobná extrakcia vlastností v jednom bode
  - TibetanStyle3by3 - extrakcia vlastností v deviatich bodoch
  - TibetanStyle3by2 - extrakcia vlastností v šiestich bodoch
  - TibetanStyle2by2 - extrakcia vlastností v štyroch bodoch
  - TibetanStyle5 - extrakcia vlastností v piatich bodoch
  - TibetanStyle3by3summed - alternatívna extrakcia vlastností v deviatich bodoch

B.3.7 Prehliadanie načítanej databázy

Užívateľ môže prezerať symboly nachádzajúce sa v databáze. Aktuálny symbol je zobrazený v ľavom dolnom rohu hlavného okna. Rôzne symboly s rovnakým reprezentantom je možné prezerať pomocou tlačidiel < a >. Pomocou tlačidiel *Previous* a *Next* môže užívateľ prechádzať zoznam načítaných reprezentantov.

B.3.8 Trénovanie klasifikátora

Po inicializácii a dodatočnej úprave klasifikátora môže užívateľ tento klasifikátor trénovať. Proces trénovania na zvolenej trénovacej databáze je zahájený pomocou tlačidla *Train*. Trénovanie je možné zrušiť tým istým tlačidlom.

B.3.9 Testovanie klasifikátora

Po ukončení procesu trénovania nového klasifikátora, alebo po načítaní uloženého klasifikátora, je možné zisťiť jeho úspešnosť rozpoznania na načítanej databáze symbolov. Výsledná úspešnosť sa zobrazuje v konzole. Užívateľ môže vidieť výsledky rozpoznania jednotlivých znakov databázy zaškrtnutím možnosti *Show details*.

B.4 Program RuleCreator

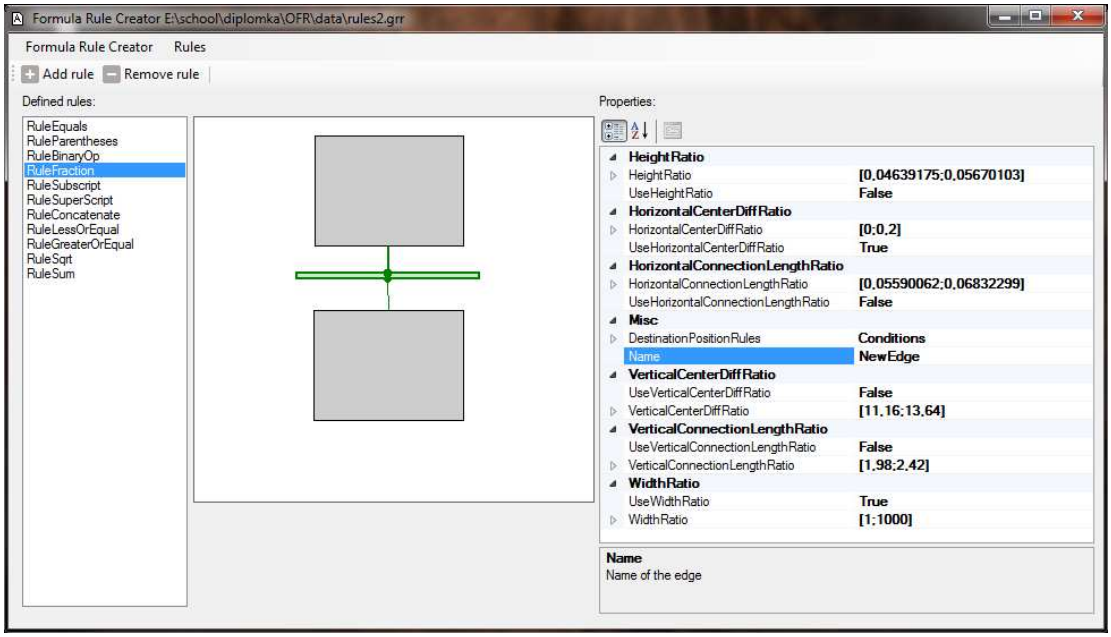
Program RuleCreator slúži na vytváranie a úpravu pravidiel prepisu grafu, ktoré sa využívajú pri rozpoznávaní štruktúry vzorcov. Hlavné okno aplikácie (viď obrázok B.7) je rozdelené do troch častí. V ľavej časti sa zobrazuje zoznam aktuálnych pravidiel. Strednú časť tvorí plátno, kde sa zobrazujú a kreslia grafy pravidiel. V pravej časti sa zobrazujú vlastnosti prvkov grafu pravidla.

B.4.1 Správa zoznamu pravidiel

Užívateľ môže vytvoriť nový zoznam pravidiel pomocou *Formula rule creator* → *New ruleset*. Vybraním *Formula rule creator* → *Load ruleset* načíta uložený zoznam pravidiel. Načítané pravidlá sa zobrazia v ľavej časti hlavného okna aplikácie. Upravený zoznam pravidiel môže užívateľ uložiť do súboru pomocou *Formula rule creator* → *Save ruleset*.

B.4.2 Vytvorenie a úprava grafu pravidla

Užívateľ môže vytvoriť nové pravidlo pomocou *Add rule* v hlavnom paneli aplikácie. V strednej časti hlavného okna sa bude nachádzať zatiaľ prázdne plátno. Užívateľ doň môže kresliť vrcholy grafu pravidla. Každý vrchol grafu je obdĺžnik. Ten je možné vytvoriť kliknutím a ťahom myši na plátne. Prvý vytvorený vrchol má zelenú farbu a je to hlavný vrchol grafu. Po zakreslení vrcholov grafu je možné vytvárať hrany medzi vrcholmi kliknutím na pravé tlačidlo myši a



Obr. B.7: Hlavné okno programu Rule Creator

ťahom od jedného vrcholu k druhému. Kliknutím na vrchol grafu a ťahom myši je možné posúvať jednotlivé vrcholy po plátne. Každý obdĺžnik je možné zväčšovať a zmenšovať kliknutím na jeho hranu alebo vrchol a ťahom myši. Pravým tlačidlom myši na nejakom elemente grafu vyvoláme kontextové menu, ktoré umožňuje zmazať daný element z grafu. Pri zmazaní vrcholu budú zmazané aj všetky hrany grafu spojené s týmto vrcholom. Pravidlo, vrchol aj hrana majú rôzne vlastnosti. Kliknutím na príslušný element (v prípade pravidla na položku zoznamu) sa zobrazia vlastnosti daného elementu v pravej časti hlavného okna. Pri každej vlastnosti sa nachádza jej popis a preto nebudeme v tejto príručke popisovať jednotlivé vlastnosti.

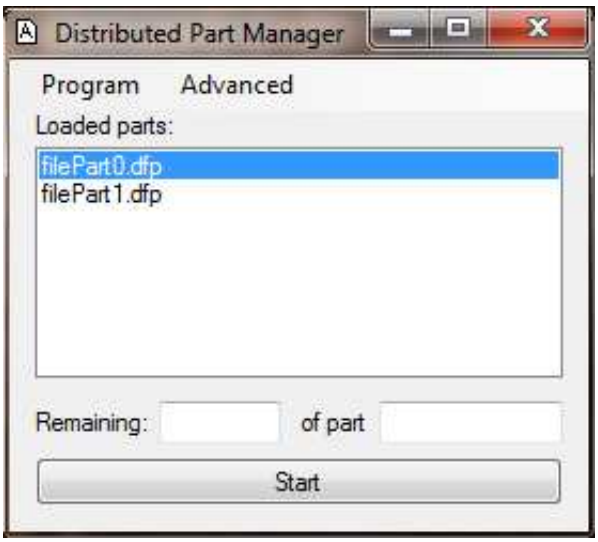
B.4.3 Odstránenie pravidla

Pravidlo je možné odstrániť zo zoznamu tak, že ho označíme v zozname pravidiel a klikneme na *Remove rule* v hlavnom paneli aplikácie.

B.5 Program Distributed Part Manager

Program DistributedPartManager slúži na trénovanie menších častí rozšíreného klasifikátora (ExtendedSuzuki) a na ich zliatie do jedného súboru dát pre rozšírený klasifikátor (LargeExtendedSuzuki). Užívateľ vyberie jednu alebo viac častí (súbory .dfp) pomocou *Program* → *Load part*. Načítané časti sa objavia v zozname častí v hlavnom okne aplikácie. Užívateľ môže tieto časti sériovo trénovať zvolením *Start*. Pod zoznamom častí sa po začatí trénovania zobrazujú informácie

o aktuálnom priebehu trénovania v rámci aktuálnej časti a informácie o poradí aktuálnej časti v rámci zoznamu častí. Výsledky trénovania každej časti sú uložené do adresára s aplikáciou. Výsledné súbory majú príponu .dfp.out. Po tom, ako sú natrénované všetky časti a uložené v jednom adresári spolu so súborom *mainPart.dfp*, ktorý bol vytvorený programom MathSymbolTrainer, je možné tieto časti spojiť do výsledného súboru pomocou *Advanced* → *Merge parts*. Po zadaní zdrojového adresára dôjde k spojeniu jednotlivých častí a vytvoreniu pomocných súborov do adresára *GroupClassifiers*, v adresári so spúšťacím súborom. V prípade použitia spojeného klasifikátora v hlavnej aplikácii, je nutné adresár GroupClassifiers skopírovať do adresára so spúšťacím súborom hlavného programu.



Obr. B.8: Hlavné okno programu Distributed Part Manager

B.6 Program RecognitionTest

Program RecognitionTest slúži na jednoduché otestovanie úspešnosti zadaného klasifikátora na zadanej množine symbolov. Každý symbol množiny sa nachádza v samostatnom súbore. Názvy súborov dodržiavajú konvenciu *test_číselný identifikátor_celý názov symbolu*. Predpokladáme formát súboru PNG s ukladaním informácii o farbe do 24 bitov na pixel.
Program sa spúšťa s nasledovnými povinnými parametrami:
Meno klasifikátora medzera *Súbor s uloženým klasifikátorom* medzera *Cesta k adresáru so symbolmi*.
Hlavný adresár so symbolmi musí obsahovať len súbory so symbolmi, žiadne iné súbory sa v adresári nesmú nachádzať. Adresár môže obsahovať podadresáre pre ktoré platia rovnaké pravidlá ako pre hlavný adresár.

Dodatok C

Podrobnosti implementácie

Podobne ako OCR systém, všetky implementované moduly a knižnice sú napísané v jazyku C# za podpory .NET Framework-u verzie 4.0. Na preloženie zdrojových kódov je potrebný program Microsoft Visual Studio 2010.

C.1 Použité knižnice

V práci používame knižnicu SVM.NET [21], napísanú v jazyku C#, ktorá obsahuje implementáciu Support Vector Machines.

C.2 Aplikácie

Podpora rozpoznávania matematických vzorcov zahŕňa okrem zásahu do OCR systému aj vytvorenie sady aplikácií, ktoré umožnia trénovať a testovať klasifikátory symbolov a vytvárať a upravovať pravidlá prepisu štruktúry. Pre tieto účely boli vyvinuté nasledovné aplikácie:

- *MathSymbolTrainer* - slúži na vytváranie a trénovanie klasifikátorov a na vytváranie databáz symbolov.
- *DistributedPartManager* - slúži na trénovanie častí ExtendedSuzuki klasifikátora a vytvorenie LargeExtendedSuzuki klasifikátora z natrénovaných častí.
- *RuleCreator* - aplikácia na vytváranie a úpravu pravidiel prepisu.
- *RecognitionTest* - pomocná konzolová aplikácia, ktorá vypisuje úspešnosť zadaného klasifikátora na zadanej množine symbolov.

C.3 Reprezentácia dát

V tejto časti popíšeme najdôležitejšie triedy používané pri rozpoznávaní symbolov a štruktúry matematických vzorcov.

- SymbolInfo - trieda slúžiaca na identifikáciu symbolu
- Symbol - trieda spravujúca symbol ako obrázok. Umožňuje načítavať obrázok so symbolom v rôznych formátoch, ďalej umožňuje meniť veľkosť obrázku, vracať obrázok ako bitmapu.
- Graph - trieda reprezentujúca graf. Umožňuje upravovať a prepisovať graf pomocou pravidiel prepisu.
- GraphRule - trieda odvodená od triedy Graph, ktorá reprezentuje pravidlo prepisu.
- GraphMatch - trieda slúžiaca na zachytenie vzťahu medzi vrcholmi pravidla a vrcholmi grafu vzorcu.
- GraphNodeMatch - trieda zachytávajúca vzťah medzi jedným vrcholom pravidla a jedným vrcholom grafu vzorcu.
- GraphNode - trieda reprezentuje uzol grafu a umožňuje zistiť, či je možné aplikovať na tomto vrchole dané pravidlo prepisu a ak áno, tak zisťuje všetky možnosti prepisu pomocou daného pravidla.
- GraphRuleNode - trieda reprezentujúca vrchol grafu, ktorý vznikol pravidlom prepisu.

C.4 Rozhrania a zásuvné moduly

Vzhľadom k použitiu systému zásuvných modulov je nutné definovať potrebné rozhrania. Väčšina modulov implementuje rozhranie IPlugin, ktoré slúži na pomenovanie a popis modulov. Toto rozhranie je definované nasledovne:

```
public interface IPlugin
{
    string GetName();
    string GetDescription();
}
```

Ďalej je nutné definovať rozhranie pre moduly slúžiace na extrakciu vlastností obrázku. Rozhranie vyžaduje definíciu veľkosti štandardného normalizovaného vstupu a definíciu dĺžky výstupného vektoru vyextrahovaných vlastností. Rozhranie ďalej vyžaduje implementáciu extrakciu vlastností a implementáciu metód na načitanie a uloženie extraktoru. Rozhranie je definované nasledovne:

```
public interface IFeatureExtractor : IPlugin
{
    int NormalizedInputWidth { get; set; }

    int NormalizedInputHeight { get; set; }

    int OutputLength { get; set; }
```

```
    double[] ExtractFeatures(ushort[,] imageData, int width, int height);

    void LoadExtractor(Stream stream);

    void SaveExtractor(Stream fs);
}
```

Aby sme docielili maximálnu variabilitu rozpoznávania, definujeme klasifikátor symbolov ako zásuvný modul. K tomuto zásuvnému modulu sa viaže rozhranie ISymbolClassifier, ktoré umožňuje serializovať a deserializovať klasifikátor, trénovať klasifikátor na zadanej tréningovej množine, rozpoznávať neznámy symbol, prípadne upraviť parametre klasifikátoru vzhľadom k zadanej množine symbolov. Rozhranie ďalej umožňuje vytvárať klon daného klasifikátora a trénovať klasifikátor tak, aby bolo možné proces okamžite zrušiť. Toto rozhranie je definované takto:

```
public interface ISymbolClassifier
{
    void Load(string file);

    void Load(Stream stream);

    void Load(Stream stream, BinaryFormatter bf);

    void Save(string file);

    void Save(Stream stream);

    void Save(Stream stream, BinaryFormatter bf);

    void Train(ITrainingSource dataSource);

    SymbolInfo Classify(Symbol symbolToClassify);

    void Initialize();

    ISymbolClassifier Clone();

    event IterationChangedEventHandler IterationChangedEvent;

    void Cancel();

    void AdjustParametersFor(IEnumerable<SymbolInfo> symbols);
}
```

Metódy Save a Load rozhrania, ktoré majú dva parametre, slúžia na serializáciu a deserializáciu zo súboru, ktorý obsahuje aj iné serializované objekty pomocou objektu typu BinaryFormatter.

Posledné rozhranie umožňuje vysokú variabilitu na úrovni systému OCR. Systém je možné dopĺňať o zásuvné moduly slúžiace na rozpoznávanie matematických oblastí. Tieto moduly implementujú nasledovné rozhranie, ktoré umožňuje načítať súčasti modulu, inicializovať modul vstupným dátami, konkrétne vstupnou matematickou oblasťou a zoznamom zatiaľ neznámych symbolov nájdených v matematickej oblasti. Rozhranie ďalej poskytuje metódu na rozpoznanie štruktúry a symbolov v oblasti a metódu na vrátenie výsledku. Metóda *Load* slúžiaca na spomínané načítavanie súčastí môže byť zavolaná viackrát.

```
public interface IMathRecognizer : IPlugin
{
    void Load();

    void Initialize(ushort[,] image, List<Symbol> symbols);

    void Recognize();

    FinalMathText GetResult();
}
```

Podrobnejší popis všetkých významných používaných tried a metód je možné nájsť v CD prílohe v adresári *dokumentacia*.