VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

ROZPOZNÁVÁNÍ OBJEKTŮ POMOCÍ EVOLUČNÍCH METOD

OBJECT RECOGNITION BY MEANS OF EVOLUTIONARY METHODS

DIZERTAČNÍ PRÁCE
DOCTORAL THESIS

AUTOR PRÁCE
AUTHOR

Ing. JIŘÍ LÝSEK

VEDOUCÍ PRÁCE
SUPERVISOR

prof. RNDr. Ing. JIŘÍ ŠŤASTNÝ, CSc.

BRNO 2013

Abstrakt
Tato práce pojednává o použití evolučních metod, konkrétně gramatické evoluce k rozpoznávání objektů v obraze. V práci jsou popsány principy rozpoznávání objektů a evoluční metody se zaměřením na gramatickou evoluci. Ve vlastní práci jsou navrženy metody pro tvorbu klasifikátorů pomocí gramatické evoluce a je navržen vhodný postup včetně návrhu výpočtu fitness funkce. Nakonec je představeno vytvořené vývojové a testovací prostředí v jazyce Java.

Summary
This thesis deals with usage of evolutionary methods, grammatical evolution particularly in application for object recognition in an image. Basic principles of object recognition and evolutionary methods with focus on grammatical evolution are described. The core of the thesis lies in design of techniques and methods for classifier programs creation using grammatical evolution. Also the designed fitness formula is presented. In the end, created testing and development environment in Java programming language is described.

Klíčová slova
evoluční metody, počítačové vidění, rozpoznávání, učení

Keywords
evolutionary methods, computer vision, recognition, learning

LÝSEK, J.*Rozpoznávání objektů pomocí evolučních metod*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2013. 82 s. Vedoucí prof. RNDr. Ing. Jiří Šťastný, CSc.

Prohlašuji, že jsem tuto práci vypracoval samostatně s pomocí informačních zdrojů uvedených v seznamu literatury.

Ing. Jiří Lýsek

Děkuji na tomto místě mému vedoucímu prof. RNDr. Ing. Jiřímu Šťastnému, CSc. za cenné odborné rady a všem ostatním, kteří mne podporovali během celého studia.

Ing. Jiří Lýsek

Obsah

1. Úvod

Evoluční metody jsou počítačové algoritmy inspirované procesem evoluce, který probíhá všude kolem nás již miliardy let. Tyto metody nehledají inspiraci v nějakém hotovém řešení, které se v přírodě nachází, ale napodobují samotnou podstatu evolučního procesu. Tedy přírodou navrženého hledání optimálních řešení.

Jde tedy o přechod genetických informací uložených v rodičích na jejich potomky pomocí procesu křížení. Dále o náhodnou mutaci, která modifikuje předané genetické informace a vliv prostředí a schopnost nového jedince v daném prostředí existovat a přežít. Tato schopnost každého organismu musí být nějakým způsobem ohodnocena - v přírodě je to délka a kvalita života takového jedince, v počítačové simulaci můžeme vhodnost jedince ohodnotit pouhým číslem.

V dnešní době roste potřeba algoritmů, které by uměly řešit mnoho typů úloh a nebylo by potřeba je příliš složitě upravovat a ladit pro nové aplikace. Takové algoritmy by mohly být nasazeny k řešení nejrůznějších typů problémů, kde zatím neznáme optimální přístupy, nebo nemáme dostatek prostředků k použití lidských expertů. Evoluční metody mohou takový rámcový systém pro řešení problémů v mnoha doménách poskytnout.

Univerzálnější přístup nabízí gramatická evoluce, která dokáže využít standardní princip optimalizace založené na evolučním principu a spojit jej s doménou řešeného problému prostřednictvím definované gramatiky a pomocí hodnotící funkce, která slouží k výpočtu vhodnosti jedince pro řešení daného problému. Gramatická evoluce je dnes úspěšně nasazována na problémy generování různých matematických modelů, kde může dosahovat lepších výsledků, než klasické metody.

U evolučních metod je ale nutné si uvědomit, že jde o úplně jiný přístup k řešení problému. Výsledek evoluční metody vždy silně závisí na parametrech a možnostech algoritmu. Někdy evoluční metoda není schopna přinést lepší řešení problému po mnoho iterací výpočtu. Potom najednou přijde díky náhodě nové řešení a tím se celý proces zlepšování řešení problému zase rozhýbe. Evoluční metody pracují velmi často s nepředstavitelně velkým prostorem řešení, který má mnoho lokálních extrémů. Díky tomuto je někdy nutné celý proces spouštět opakovaně, aby bylo nalezeno to nejlepší řešení.

Dalším intenzivně zkoumaným tématem je využití metod umělé inteligence k zastoupení lidí. Konkrétně snaha naučit počítače a roboty vnímat – rozpoznávat. Důvody k nasazení umělé inteligence jsou různé, ale výsledkem by mělo být vždy zlepšení proti předchozímu stavu. Proto je nutné vylepšovat a hledat metody, jak stroje naučit efektivně pomáhat.

V této práci bude gramatická evoluce použita jako nástroj pro hledání spustitelných programových struktur, které na základě vstupních dat budou schopné identifikovat předložené objekty.

1.1. Cíl práce

Cílem práce je navrhnout variantu genetických algoritmů pro aplikaci v oblasti rozpoznávání objektů. Jde tedy o aplikaci evolučních metod v oblasti, kde tradičně používáme buď klasické statistické metody, nebo zaběhnuté metody umělé inteligence, jako jsou na-

příklad neuronové sítě. Práce je zaměřena hlavně na aplikaci gramatické evoluce, která je vhodná k vytváření struktur, které lze spouštět jako počítačové programy. Abychom takovéto struktury získali a byly užitečné, je nutné pomocí evolučního procesu tyto programy nejprve nechat vyvinout. Tento proces se dá přirovnat k učení neuronové sítě před jejím nasazením k řešení úkolu. I pomocí evoluční metody vlastně do programové struktury ukládáme informace, které pak slouží k vyřešení daného problému. Na rozdíl od neuronové sítě však tato struktura není nikdy pevně daná a její parametry jsou dopředu neznámé.

Ke splnění tohoto cíle je nutné vytvořit testovací prostředí, kde bude možné spouštět různé varianty evolučních algoritmů. V tomto prostředí potom budou testovány různé možnosti jak využít evoluční metody k rozpoznávání objektů v obraze.

1.2. Členění práce

Práce je rozdělena na teoretickou část, kde je přehled metod pro rozpoznávání objektů a popis evolučních metod a jejich modifikací se zaměřením hlavně na gramatickou evoluci. V další části práce je pak popis využití genetických algoritmů pro vlastní řešení problému rozpoznávání objektů. Nakonec je popsáno vytvořené testovací prostředí a jsou prezentovány výsledky algoritmů na testovacích objektech.

2. Rozpoznávání objektů v obraze

Rozpoznávání objektů je široký obor s mnoha úspěšnými aplikacemi. V dnešní době hojného rozšíření levných digitálních kamer, integrovaných do všemožných zařízení, má nějakou aplikaci počítačového vidění k dispozici každý majitel takovéhoto zařízení (například detekce obličejů ve fotoaparátu, rozpoznávání a autorizace uživatele PC pomocí kamery, ...). Tyto aplikace sloužící převážně k zábavě jsou spíše vedlejším produktem aplikací s důležitou rolí v průmyslu, bezpečnosti, vojenství, zdravotnictví a jiných oborech, na kterých můžou záviset ekonomiky států nebo bezpečnost obyvatel. Pro aplikace metod počítačového vidění jsou tudíž hlavní kritéria rychlost a spolehlivost.

S rozvíjejícím se výpočetním výkonem počítačů je možné aplikovat stále sofistikovanější metody pro rozpoznávání objektů. Někdy však požadavek na rychlost rozpoznání stojí proti požadavku na spolehlivost a je nutné najít vhodný kompromis.

2.1. Důležité parametry procesu

Proces rozpoznávání má podle své aplikace a použité metody následující důležité vlastnosti. Podle aplikace samozřejmě vyžadujeme různé kombinace nebo míru těchto parametrů. Pokud se uvažuje například průmyslová aplikace pro rozpoznávání víceméně plošných objektů, bude důležitým parametrem rychlost a spolehlivost. Podle typu objektů budeme vyžadovat ještě invarianci vůči rotaci, či stranovému převrácení.

Často je při nasazování metod umělé inteligence zároveň nutné upravit i výrobní proces. Například zavedením určitých značek na objektech ve výrobních linkách.

2.1.1. Spolehlivost

Hlavním úkolem algoritmů pro rozpoznávání objektů je spolehlivost. Proto se u všech metod nejčastěji sleduje právě tento parametr.

2.1.2. Rychlost

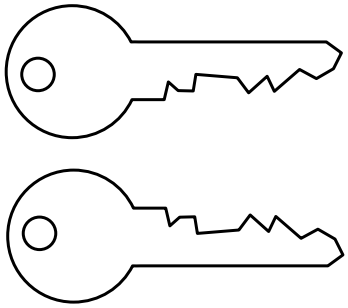
V některých aplikacích je důležité, aby proces rozpoznání proběhl co nejrychleji. Rychlost ale může být vykoupena nesplněním některých dalších parametrů procesu. Rychlost procesu je také často faktorem, který ovlivní cenu celého řešení. Ať už na nakoupeném hardware nebo na nákladech na vývoj a optimalizaci metody.

2.1.3. Invariance vůči rotaci

Často kladeným požadavkem je, aby metoda nebyla citlivá na orientaci objektů, které mají být rozpoznány. Je to logický požadavek, neboť například na dopravníkovém pásu nikdo nezaručí, že objekty budou přicházet vždy srovnané.

2.1.4. Invariance vůči stranovému převrácení

Tato vlastnost je dobře demonstrována na obrázku klíče 2.1. Pokud naše metoda umožní rozlišení horního a dolního klíče, je invariantní vůči stranovému převrácení.



Obrázek 2.1: Ukázka stranově převrácených objektů

2.1.5. Invariance vůči změně měřítka

Tato vlastnost může být někdy nežádoucí. Jde o to, že by metoda měla být schopna rozpoznat stejné objekty i v různých vzdálenostech od snímače.

2.2. Získání obrazu

Obraz pro zpracování do počítače obvykle dostaneme pomocí digitálního fotoaparátu nebo jako sekvenci z digitální videokamery. Bylo vyvinuto mnoho typů digitálních snímačů postavených na různých technologiích (*CCD*, *CMOS*, ...). Od technologie snímače závisí i počet barev, které je schopen zachytit – některé snímače dokážou snímat jen intenzitu světla. Avšak existují i zařízení, která dokážou zaznamenávat obraz mimo viditelné spektrum nebo v noci.

Důležitým parametrem po technologii snímače, je počet bodů na snímači – rozlišení, které dokážou zachytávat světlo a měnit jeho intensitu na elektrický signál.

Další možnost je obraz uměle vytvořit na počítači. Existuje mnoho softwarových nástrojů, které dokážou generovat realistický obraz například definovaných 3D scén nebo jinak simulovat vstup z fotoaparátu nebo kamery. Pro testovací účely se mnohdy volí právě uměle vytvořené obrazy.

Někdy může být obrazová informace doplněna ještě o další informace jako je například záznam vzdáleností objektů od kamery, teplot nebo zvuková stopa.

2.3. Reprezentace obrazu

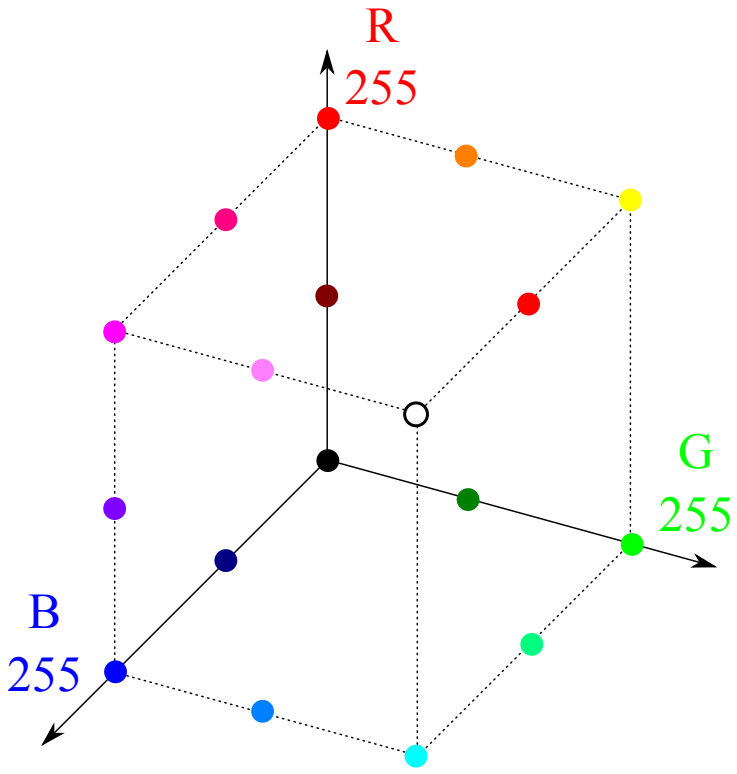
Obrazová informace je v počítači uchována jako číselná matice. Každá hodnota v této matici odpovídá jednomu bodu, který může být zobrazen na obrazovce počítače – tento počet bodů nemusí být nutně stejný, jako počet bodů snímače použitého pro pořízení obrazu. První důležitou informací je tedy velikost této matice.

Dalším kritériem je počet těchto matic. Pro uložení obrazu se používají obvykle takovéto systémy:

- 1 matice – stupně šedi, binární obraz
- 3 matice – systémy *RGB*, *HSV*, ...
- 4 matice – systémy *RGBA*, *CMYK*, ...

Každá matice reprezentuje jednu složku obrazu.

Nejčastěji se používá pro ukládání obrazu *RGB* barevný model s 8 bity na barvu (obrázek 2.2), který je aditivní a je složený ze tří složek R – červená (Red), G – zelená (Green) a B – modrá (Blue). Mícháním třech základních složek barvy získáme 256^3 barev.



Obrázek 2.2: RGB barevný model se znázorněnými body

Pokud volíme reprezentaci obrazu ve stupních šedi, pak se obvykle používá rozdělení na 256 stupňů – tedy 8 bitů na každou hodnotu matice. V počítačovém zpracování obrazu velmi často pracujeme právě s obrazem ve stupních šedi, jelikož nám dovoluje pracovat s menším objemem dat, než při práci s barevným obrazem.

Pokud chceme převést barevný obraz na obraz ve stupních šedi, existuje snadný přepočet. Někdy je tento přepočet nutný i z důvodů, že výstupní zařízení (například tiskárna) nepodporuje barevný výstup.

2.3.1. Histogram

Histogram je často používaná charakteristika obrazu. Je to přehled počtu obrazových bodů v určité barvě. Dá se vytvořit pro každou složku obrazu. Příklad histogramu je na obrázku 2.3.

Obrázek 2.3: Zdrojový obraz (vlevo) a histogram pro jednotlivé barevné složky i stupně šedi (vpravo)

2.4. Předzpracování

Většina aplikací pro rozpoznávání objektů v obraze používá nějaké předzpracování obrazu pro zlepšení výsledků dané metody. Všechny kroky předzpracování je nutné provádět s rozvahou, abychom použité metodě pro popis objektů a rozpoznání nezabránili správně fungovat.

2.4.1. Odbarvení

Jelikož obraz ve stupních šedi obsahuje třikrát méně informací než běžný barevný obraz, je často prvním krokem převod barevného obrazu do obrazu v odstínech šedi. Tento krok obvykle výrazně urychlí práci s obrazem. Pokud je potřeba pracovat s barvou objektů, pak se často používají histogramy celého obrazu nebo jeho částí.

$$Y = 0,299 \cdot R + 0,587 \cdot G + 0,114 \cdot B \tag{2.1}$$

Existuje více variant přepočtů, které je snadné najít na internetu. Pro výpočet odstínu šedé z *RGB* systému je možné použít například vzorec 2.1. Vzorce obvykle sdílí následující vlastnost: největší hodnota je u zelené složky, na kterou je lidské oko nejcitlivější. Naopak nejméně je lidské oko citlivé na modrou složku, proto je koeficient ve vzorci u modré složky nejmenší.

2.4.2. Změna velikosti

Další zmenšení objemu zpracovávaných informací je možné dosáhnout změnou velikosti obrazu. Je možné aplikovat různé algoritmy pro získání zmenšeného obrazu.

2.5. Detekce míst zájmu, segmentace

Následujícím krokem může být vyhledání oblastí obrazu, na které následovně budeme rozpoznávat objekty. Tento krok často klade důraz na získání částí obrazu – segmenty,

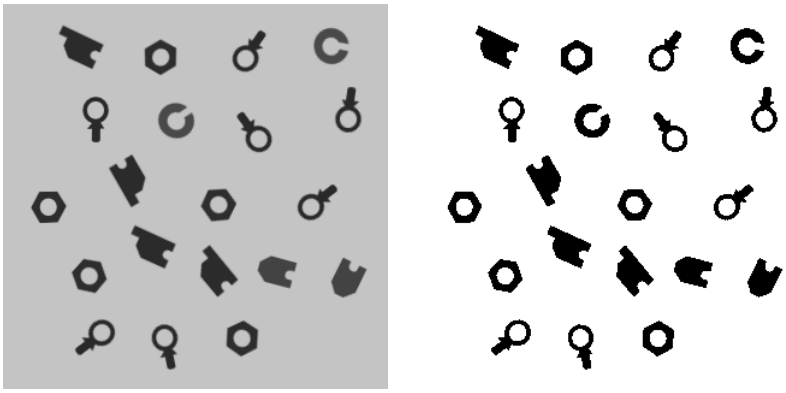
které potenciálně obsahují nějaké objekty a odstranění oblastí obrazu, kde je pozadí. V podstatě se jedná o binární klasifikaci.

Segmentaci lze přeskočit, pokud použijeme posuvné okno, které si prohlédne celý obraz krok po kroku.

2.5.1. Prahování

Pomocí prahování můžeme oddělit světlé a tmavé oblasti obrazu. Obvykle se pracuje s předem nastavenou hodnotou pro oříznutí. Sofistikovanější aplikace pracují s histogramem obrazu. Pomocí prahování lze obraz segmentovat. Ukázka prahování je na obrázku 2.4.

Prahování pomocí histogramu lze provádět i dynamicky na menších výřezech z obrazu.



Obrázek 2.4: Ukázka prahování – zdrojový obraz (vlevo) a jeho varianta po operaci prahování (vpravo)

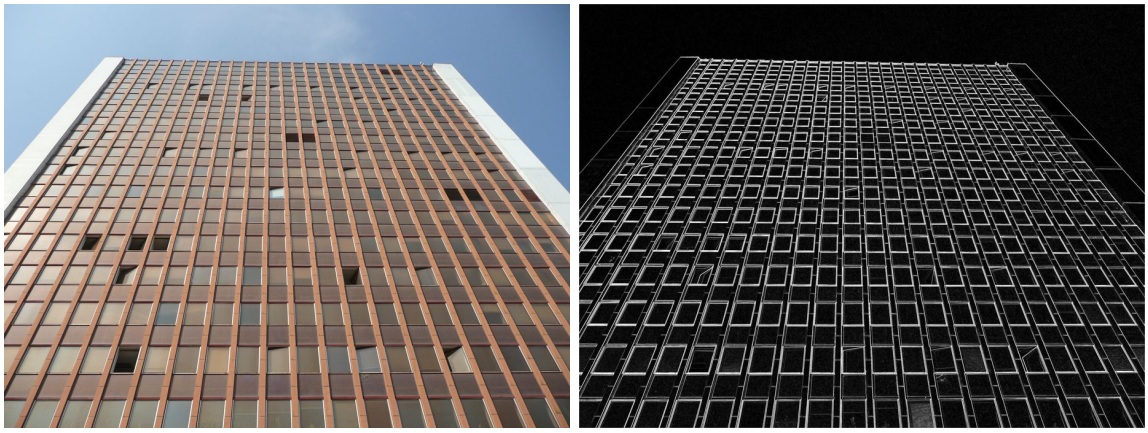
2.5.2. Hrany a rohy

Existují postupy, pomocí kterých lze z obrazu získat místa, kde jsou hrany (prudké změny gradientu). Jsou to například Sobelův operátor, Canny edge detector a další. Pracují obvykle na principu konvoluce obrazu a specifického konvolučního jádra. Další obvyklé detektory jsou určeny k detekci rohů (křížící se hrany). Pomocí nalezených hran lze obraz opět segmentovat. Příklad detekce hran je na obrázku 2.5.

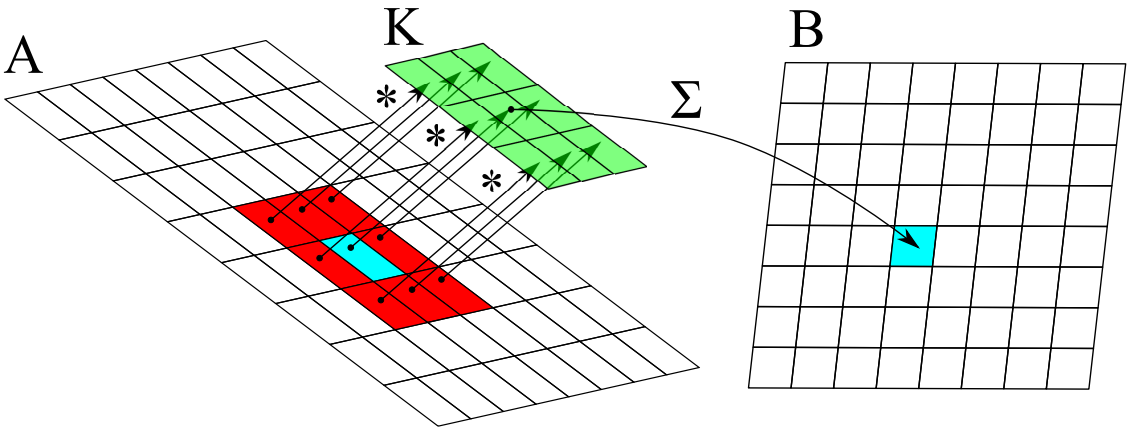
Konvoluce

Pro detekci hran a mnoho dalších operací při předzpracování obrazu se používá operace konvoluce. Vzorec 2.2 ukazuje její diskrétní variantu pro výpočet bodu v matici B daného souřadnicí x a y. Graficky je znázorněna na obrázku 2.6. Je to operace definovaná pro dvě matice A a K. Kde K se nazývá konvoluční jádro, tato matice je malá a obvykle čtvercová o velikosti $2 \cdot k + 1$ řádků i sloupců. Existuje mnoho známých konvolučních jader, které dokážou například zaostřit, rozostřit, posunout, odbarvit vstupní obraz.

$$B(x, y) = (A * K)(x, y) = \sum_{i=1}^{2 \cdot k+1} \sum_{j=1}^{2 \cdot k+1} A(x + i - (k + 1), y + j - (k + 1)) \cdot K(i, j) \quad (2.2)$$



Obrázek 2.5: Zdrojový obraz (vlevo) a hrany zjištěné pomocí Sobelova operátoru (vpravo)



Obrázek 2.6: Princip konvoluce

2.5.3. Ostatní

Podle aplikace je možné najít libovolné jiné místa zájmu. Například v průmyslových aplikacích se používají tzv. *markery*, které jsou nalepené na různé výrobky či stroje. Tyto místa je potom nutné vyhledat.

2.6. Získání popisu objektu

Pokud vlastní rozpoznání objektů nepracuje přímo na obrazové matici, je nutné z obrazových segmentů získat popis objektů. Například získat statistické charakteristiky o barvě nebo pomocí frekvenční analýzy informace o frekvencích v obraze. Další možností je využít hrany objektů pro získání popisu kontury objektu. Popis objektů je nutné navrhnout specificky podle řešené úlohy.

2.7. Vlastní rozpoznání – přehled metod

Vlastní rozpoznávání objektů můžeme realizovat pomocí metod umělé inteligence nebo statistickou analýzou. Obvykle dojde k předložení získaného popisu objektu tzv. klasifi-

kátoru, který s určitou mírou přesnosti dokáže určit, o který objekt se jedná. Klasifikátor je dopředu seznámen s objekty, které mu mohou být předkládány. Tento proces se nazývá učení. Pokud je mu předložen neznámý objekt, obvykle nedokáže upravit naučené znalosti, aby jej dokázal rozpoznat. Existují samozřejmě systémy, které se dokážou adaptovat.

2.7.1. Běžné metody umělé inteligence

Pro rozpoznávání libovolných objektů se nejčastěji používá neuronová síť. Je to struktura tvořená umělými neurony [24], které představují model lidského neuronu a tyto jsou navzájem různě propojeny. Podle systému propojení se neuronové sítě dělí na dopředné a se zpětnou vazbou. Rozložení neuronů a jejich propojení se souhrnně nazývá topologie.

Pro rozpoznávání se často používají dopředné vícevrstvé sítě (*MLP*). Neurony jsou v nich uspořádány po vrstvách a každá vrstva má přístup k výstupním hodnotám předchozí vrstvy. Pro učení takovýchto sítí se používá algoritmus backpropagation [36].

Dalším klasifikačním nástrojem může být Kohonenova samoorganizační mapa [17]. Tato síť pracuje na principu shlukování podobných vzorů. Dá se tedy použít k automatické klasifikaci bez učitele.

Dnes již zastaralou, sítí je Hopfieldova síť [16], která dokáže rozpoznávat pouze binární vstupní vzory.

Dalším možným nástrojem pro klasifikaci jsou tzv. *Support Vector Machines* (*SVM*) [9]. A mnoho jiných.

V neposlední řadě je možné vytvářet různé rozhodovací stromy nebo jiné modely založené i na obecné popisné statistice.

2.7.2. Přístupy založené na evolučních metodách

V současnosti již byly evoluční algoritmy více či méně úspěšně použity k řešení různých úkolů k týkajících se rozpoznávání objektů v obraze. Některé aplikace používaly evoluční metody jen jako podporu k řešení vlastního problému například pomocí neuronových sítí. Například v [6] jde o optimalizaci dat použitých k učení klasifikátoru.

Jiné aplikace využívají genetické algoritmy přímo k operacím vedoucím ke klasifikaci objektů. Tyto lze rozdělit podle použité varianty evoluční metody na implementace se specifickým chromozomem a ty, které používají gramatickou evoluci.

2.7.2.1. Metody využívající specifickou interpretaci chromozomu

Tyto metody obvykle vyžadují více informací o řešené úloze. Programátor musí vymyslet, jak bude uložen chromozom, jak bude probíhat křížení a mutace a jak budou jedinci interpretování a hodnoceni. Výsledkem je tedy systém, který není použitelný univerzálně, ale pouze k řešení jednoho typu úlohy. Výhodou může být větší rychlost navrženého systému pro řešení požadované úlohy. U těchto přístupů jde většinou o přiřazování vybraných charakteristik z předloženého obrazu k charakteristikám uloženým v databázi modelů.

Přiřazování kontury objektu k modelu
Dobré výsledky jsou dosaženy pro přiřazování kontury objektu z databanky ke kontuře získané například detekcí hran. Chromozom jedince může kódovat třídu objektu z databanky,

jeho natočení, pozici jeho těžiště apod. Další možná informace může být zkosení, zúžení čí zploštění objektu. Pro vyhodnocení fitness se obvykle používá výpočet vzdáleností bodů scény modelované genetickým algoritmem od bodů získaných hranovým detektorem. Čím je větší shoda modelu s předloženou scénou, tím je vzdálenost všech bodů menší.

Pokud ve scéně předpokládáme více než jeden objekt, je možné genetický algoritmus spustit vícekrát a postupně odmazávat nalezené objekty.

Výhoda tohoto přístupu je hlavně ve schopnosti rychlé adaptace genetických algoritmů na proměnlivé prostředí, takže není problém aplikovat takovouto metodu i pro pohyblivou scénu.

Někdy jsou pomocí genetického algoritmu přiřazována jen geometricky primitivní tvary (kruhy, čtverce apod.) jako v [34]. Detekce jednoduchých tvarů, lze použít například pro detekci dopravních značek, jako v [14]. Někdy je v databance objektů uložena informace o kompletní kontuře objektu dokonce i ve více možných pohledech. Takový postup byl použit např v [4], [43], [12].

Přiřazování jiných charakteristik k modelu

Podobně jako přiřazování kontury lze pomocí evolučních metod přiřazovat jiné charakteristiky získané z obrazu k charakteristikám uloženým v databázi modelů. Např . v [40] je genetický algoritmus použit k přiřazování uzlů grafu obsahujícího atributy obrazu.

2.7.2.2. Metody využívající gramatickou evoluci

Výhodou použití gramatické evoluce v jakékoliv aplikaci evolučních metod je oddělení procesu evoluce od interpretace jedinců pro konkrétní řešenou úlohu. Gramatická evoluce vidí jedince jako programy generované gramatikou (což je samozřejmě interpretace číselného chromozomu). Tato interpretace dovoluje provádět standardní operace mutace a křížení, což algoritmu stačí k tvorbě nových jedinců a nepotřebuje k tomu znalosti o řešeném problému.

Gramatickou evoluci nezajímá, k čemu je program nakonec použit. Pro problém řešený pomocí gramatické evoluce tedy potřebujeme nadefinovat pouze gramatiku a hodnotící funkci pro získání hodnoty fitness každého jedince.

Do jisté míry se tedy dá říct, že gramatická evoluce je obecný nástroj pro tvorbu jakýchkoliv počítačových programů či jiných struktur, které jsou generovány gramatikou.

Použití těchto metod je lákavé hlavně proto, že slibuje aplikaci v problémech, kde dopředu neznáme informace o předkládaných objektech. Vhodné by mohlo být například využití k automatické klasifikaci velkého množství objektů na obrazech (například v prostředí internetu), kde by evoluční metoda mohla nalézt společné charakteristiky velkého množství označených objektů na poměrně malém vzorku obrazů. Následně by mohlo probíhat vyhledávání obrazů s podobnými charakteristikami v určitých lokacích a tyto výsledky by se použily jako odpověď na vyhledávací dotaz.

Metody založené na posuvném okně

Nejobecnější přístup k rozpoznávání objektů v obraze mají metody založené na přístupu posuvného okna. U těchto metod nedochází v podstatě k žádné extrakci dat z obrazu. Matice obrazových bodů je přímo předložena nástroji pro rozpoznání třídy objektu.

U přístupu využívající posuvné okno je nutné postupně předložit klasifikátoru celý obraz a to může v závislosti na složitosti klasifikátoru trvat poměrně dlouho. Jediný parametr, kterým lze u této metody ovlivnit rychlost procházení obrazu je krok okna. Pokud se posuvné okno nepohybuje bod po bodu, ale vynechává každý druhý bod, sníží se doba nutná k průchodu celého obrazu 4×. Čím větší krok, tím méně času je nutné pro zpracování celého vstupu.

Někdy jsou tyto metody používány jen k detekci objektů (klasifikace do 2 tříd). Tento přístup byl využit například k detekci obličejů v [44].

Jindy jsou evoluční metody použity k tvorbě programů, které dokáží klasifikovat objekty do více tříd [46, 45]. Zde je potom velký problém s nastavením výstupních hodnot jedinců (programů), které obvykle vrací jen desetinné číslo. Naproti tomu neuronové sítě dokážou na svém výstupu vrátit celý vektor, který se snadněji interpretuje.

Použití gramatické evoluce k získání lokálních deskriptorů
Gramatickou evoluci je možné použít k získání lokálních deskriptorů podobných těm, které dostáváme při použití metody SIFT [20] nebo SURF [2]. V [28] je popsán postup získání krátkých programů, které označují místa k vypočtení lokálních deskriptorů. Tyto deskriptory, ať už získané pomocí kterékoliv metody, mohou být použity k přiřazování odpovídajících míst na předloženém obraze k místům uloženým v nějaké databázi a tím můžeme dosáhnout rozpoznání objektů.

Generování L-systémů pomocí gramatik
L-systém [33] (Lindenmayerův systém) je paralelní přepisovací gramatika, která je schopná modelovat fraktály. Pokud tuto gramatiku poskytneme gramatické evoluci, budeme dostávat různé řetězce. Tyto jsou obvykle znázorněny graficky pomocí tzv. želví grafiky a porovnány s předloženým obrazem.

Pro výpočet fitness hodnoty se použije podobný postup jako při postupu s přikládáním kontur.

Příklad použití tohoto postupu je v [3] a [27], kde jsou pomocí L-systémů hledány vhodné aproximace předložených tvarů.

Někdy je tento postup použit k vytváření obrazů v počítačové grafice. Není zde pak automatická hodnotící funkce, ale člověk, který řadí vygenerované obrazce podle toho, jak se mu jednotliví jedincí vizuálně líbí.

2.7.3. Další příklady použití evolučních metod v oblasti počítačového vidění

Evoluční metody jsou často používány k řešení podpůrných úloh, které přímo nesouvisí s klasifikací objektů. Velmi dobré výsledky jsou dosahovány u problémů segmentace obrazu. Jedinci obvykle kódují rozložení regionů nad obrazem (určují příslušnost obrazových bodů do regionů).

Optimalizovaná hodnota pak může být například co nejmenší rozdíl v nějaké sledované charakteristice (barva, textura, ...) v oblastech spadajících do jednoho regionu proti co největší rozdílnosti v odlišných regionech. Další optimalizované kritérium může být délka hranic regionů. Příklady se dají nalézt v [13, 10, 39, 5, 38].

Další zajímavá aplikace je k nalezení v [19], kde je genetický algoritmus použit k získání informací o rozdílech ve stereo obrazech. Toto se dá využít například k sestrojení 3D mapy terénu z leteckých fotografií.

3. Evoluční metody

V této kapitole budou popsány základní vlastnosti algoritmu, bude zavedeno základní názvosloví a bude vysvětlen průběh řešení úlohy pomocí evolučního algoritmu. Dále bude popsána gramatická evoluce. Nakonec budou vyhodnoceny výhody a nevýhody evolučních metod.

3.1. Názvosloví

Evoluční metody jsou inspirované procesem evoluce v přírodě. Proto i při řešení úlohy pomocí algoritmu založeného na těchto metodách používáme názvosloví převzaté z genetiky.

3.1.1. Gen

Konkrétní část chromozomu. Pro kódování genu se obvykle používají čísla. Je možné použít buď binární kód nebo jiné kódování.

3.1.2. Alela

Konkrétní hodnota daného genu.

3.1.3. Chromozom

Je to část genetické informace jedince. Pro kódování chromozomu obvykle používáme řetězec nebo pole. Je to tedy vektor čísel nebo znaků definovaný vzorcem 3.1.

$$\vec{ch} = [g_1, g_2, \ldots, g_n] \tag{3.1}$$

3.1.4. Fenotyp

Fenotyp je konkrétní forma jedince, který vznikne dekódováním chromozomů.

3.1.5. Genotyp

Celkový soubor informací o jedinci je uložen v genotypu. Tyto informace jsou použity při vytváření konkrétní formy jedince (překladu). Genotyp je předpis, jak vytvořit konkrétní fenotyp. Genotyp obsahuje jednotlivé chromozomy – je to tedy matice definovaná podle vzorce 3.2. Pokud genotyp obsahuje právě jeden chromozom, jsou pak tyto totožné.

$$G = \{\vec{ch}_1, \vec{ch}_2, \ldots, \vec{ch}_n\} \tag{3.2}$$

3.1.6. Jedinec

U evolučních metod se takto označuje jeden kandidát řešení problému – množina 3.3. Tento jedinec musí držet svůj genotyp G a případně na něj můžou být navázány další konkrétní informace jako například věk (v – reprezentovaný počtem iterací algoritmu, kterými jedinec prošel) nebo jiné informace. Dodatečné informace nejsou použity k překladu genotypu na fenotyp.

$$M = \{G, v, \ldots\} \tag{3.3}$$

3.1.7. Populace

Populace je množina jedinců v daný časový okamžik – definovaná vzorcem 3.4. V průběhu algoritmu máme pro každou iteraci (časový okamžik) jinou populaci.

$$P(t) = \{M_1, M_2, M_3, \ldots\} \tag{3.4}$$

3.1.8. Selekce

Proces výběru jedinců z populace pro další zpracování.

3.1.9. Křížení – crossover

Proces výměny informací mezi dvěma nebo více jedinci vybranými pomocí selekce. Výsledkem křížení je nový jedinec, nebo více nových jedinců. Tito noví jedinci jsou kombinací jedinců, kteří do operace vstoupili.

$$\{M'_1, \ldots, M'_m\} = krizeni(M_1, \ldots, M_n) \tag{3.5}$$

3.1.10. Mutace

Nový jedinec může za určitých podmínek náhodně změnit část svého genotypu. Takto vzniklý jedinec má v sobě uloženu novou informaci, která s předchozí formou jedince nesouvisí.

$$M' = mutace(M) \tag{3.6}$$

3.1.11. Fitness

Pod tímto pojmem si obvykle představujeme u každého jedince jakousi hodnotu, která říká, jak moc je daný jedinec vhodný a schopný. V našem případě tím hodnotíme kvalitu jedince z pohledu schopnosti vyřešit problém. Tato hodnota se obvykle počítá nějakým vzorcem, nebo může jít o jakési skóre, které jedinec získá existencí v nějakém prostředí.

$$hodnota = Fit(M) \tag{3.7}$$

3.2. Vlastnosti evolučních algoritmů

V této části budou popsány vlastnosti evolučních metod z pohledu zařazení do určitých rodin algoritmů.

3.2.1. Iterativní proces

Jedná se o iterativní proces. Algoritmy pracují vždy s jednou populací jedinců a tu pomocí operátorů transformují do nového stavu. Iterativní algoritmy mají obvykle tu výhodu, že dokáží řešení problému postupně vylepšovat a případně se mohou i vracet k předchozím kvalitním řešením a pracovat s nimi.

3.2.2. Přímé prohledávání

Optimalizační metody používající přímé prohledávání prostoru řešení a nevyužívají derivaci funkce, kterou optimalizují k získání informací o gradientu nebo lokálních extrémech. Jejich výhoda spočívá v tom, že optimalizovaná funkce nemusí mít definovanou derivaci a nemusí být spojitá.

Genetický algoritmus vyhodnocuje řešení pouze podle aktuální kvality jedinců, které porovnává mezi sebou.

Nevýhodou takovýchto algoritmů je samozřejmě nejistota, zda nové řešení problému, vzniklé kombinací dvou existujících bude lepší.

3.2.3. Stochastický proces

Evoluční algoritmy jsou stochastické, jelikož často pracují s náhodným výběrem a už samotná inicializace algoritmu je náhodná. Nelze u nich zaručit stejný výsledek ani při zachování stejných vstupních podmínek a dat.

Proto je vhodné metodu spustit několikrát a vhodné řešení problému vybrat až z množiny více řešení.

3.2.4. Heuristický algoritmus

Evoluční algoritmy obvykle nejsou schopné najít úplně přesné a nejlepší řešení. Jejich síla spočívá ve schopnosti velmi rychle nalézt vhodné řešení. O optimálnosti nalezeného řešení ovšem nelze rozhodnout bez prohledání celého prostoru možných řešení.

3.3. Obecné evoluční metody

V této kapitole budou uvedeny základní principy evolučních metod. Evoluční metody jsou široká vědní disciplína, její vznik se datuje do sedmdesátých let a za autora je považován John Holland. Existuje mnoho variant, v této části práce jsou uvedeny obecně platné informace.

3.3.1. Použití

Studované algoritmy jsou obvykle používány k řešení úloh optimalizačního charakteru. Evoluční metody pracují s populací kandidátních řešení (populace jedinců), která jsou modifikována (mutace) a kombinována (křížení), aby byl prohledán co největší prostor možných řešení. Každý kandidát má pomocí fitness funkce vypočítánu hodnotu, která říká, jak moc je vhodný pro řešení daného problému.

3.3.2. Kódování jedinců

Podobně jako v přírodě, je i u evolučních metod chromozom pouze předpisem, jak získat funkčního jedince (fenotyp).

3.3.2.1. Binární kódování

Jednotlivé geny jedinců mohou nabývat pouze hodnot 1 nebo 0. Takto kódovaní jedinci jsou často interpretováni jako číslice. Aby nedocházelo k příliš velkým změnám fenotypu při změně genotypu, používá se Grayův kód.

3.3.2.2. Celočíselné kódování

Pokud není nutné mít kontrolu nad jednotlivými bity, používá se celočíselné kódování. Toto kódování je použito například u gramatické evoluce.

3.3.2.3. Jiné

V závislosti na aplikaci je možné zvolit jakékoliv jiné kódování. Například diferenciální evoluce, která se užívá k optimalizaci pevného počtu parametrů, umožňuje pracovat s chromozomem složeným z reálných čísel.

3.3.2.4. Proměnlivá délka chromozomu

Většinou se používají chromozomy, které mají pevně daný počet hodnot (obvykle číslic). Ve speciálních případech je možné použít i chromozom s proměnlivou délkou. Použití se nabízí u gramatické evoluce. Nicméně i tam je vhodné nastavit jakousi horní hranici.

3.3.3. Operátory

V každé iteraci algoritmu jsou na celou populaci aplikovány operátory, které zajišťují vznik nové populace jedinců pomocí kombinace a modifikace jedinců z aktuální populace. Ještě než jsou tyto operátory aplikovány, je potřeba vybrat pomocí nějaké strategie jedince, na které budou tyto operátory aplikovány. Tento proces se nazývá selekce a úzce souvisí s operátory. Pomocí operátorů je transformována populace $P(t)$ na populaci $P(t+1)$.

3.3.3.1. Výpočet fitness

Předtím, než je možné provádět výběr jedinců, je nutné jedince ohodnotit hodnotou fitness.

Pro ohodnocení kvality jedince se používá tzv. fitness funkce. Její charakter je důležitý pro výsledek celého procesu. Měla by vést algoritmus tak, aby postupně byl schopen objevovat nová a lepší řešení – to je možné pouze tehdy, pokud fitness funkce dokáže dostatečně silně odlišit slabší a silnější jedince. Na druhou stranu, velké skoky v hodnotách fitness také nejsou žádoucí.

Hodnota fitness funkce se obvykle maximalizuje, ale lze samozřejmě fitness i minimalizovat. Pokud nemáme možnost zasáhnout do algoritmu, je nutné fitness funkci transformovat například následujícím vzorcem 3.8.

$$Fit'(Mjedinec) = \frac{1}{1 + Fit(M)} \tag{3.8}$$

Vlastní výpočet fitness
Často je problém jasně definován a výpočet hodnoty fitness je dán vlastní navrženou funkcí.

Přiřazení fitness na základě kvality – rankování
Jedinci jsou porovnáni navzájem a seřazeni podle úspěšnosti k řešení problému. Hodnota fitness je přiřazena na základě pořadí (kvality) jedince v populaci.

3.3.3.2. Selekce

Selekce slouží k výběru jedinců z populace podle nějakého kritéria. Tito vybraní jedinci jsou potom použiti v dalších operátorech.

Truncation selection – oříznutí
Při tomto typu selekce jsou odstraněni ti nejhorší jedinci populace. Je tedy nutné populaci nejprve setřídit. Ostatní jedinci jsou použiti k vytvoření potomků, aby naplnili populaci.

Turnajová selekce
Turnajový výběr je často používaný a jednoduchý způsob výběru jedinců. Spočívá v náhodném výběru zvoleného počtu jedinců. Z těchto vybraných jedinců je potom vybrán ten nejlepší. Důležitým parametrem je tedy počet jedinců vybíraných do turnaje. Pokud je vybrán právě jeden jedinec, jedná se o náhodný výběr.

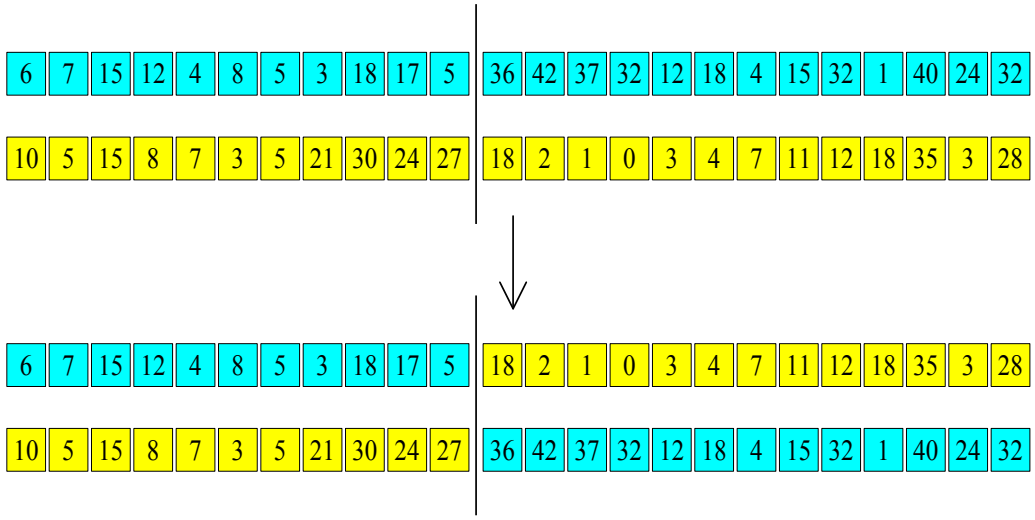
Ruletová selekce
Ruletový výběr je založen na výběru jedinců v závislosti na jejich kvalitě. Větší šanci na výběr má jedinec, který má vyšší hodnotu *fitness*. Nejprve je proveden součet všech hodnot *fitness* u všech jedinců populace, tu můžeme označit *FitnessSum*. Poté se vygeneruje náhodné číslo v intervalu <0, *FitnessSum*>.

3.3.3.3. Operátor křížení

Křížení je výměna genetických informací mezi dvěma nebo více jedinci. Křížení samo o sobě do populace žádné nové fenotypy nepřináší. Slouží pouze k optimalizaci rozložení již existujících fenotypů.

Uniformní křížení
Je nejjednodušším typem křížení. Je možné jej implementovat jen na chromozomy se stejnou délkou. Nastaví se určitá pravděpodobnost, podle které budou vyměňovány jednotlivé geny v chromozomu s geny na stejné pozici v chromozomu druhém.

Bodové křížení
Pro binární nebo celočíselné chromozomy se nejčastěji používá jednobodové (obrázek 3.1) nebo dvoubodové (obrázek 3.2) křížení. Princip spočívá v náhodné volbě místa nebo míst křížení. Číselné sekvence jsou následovně upraveny tak, že na každé hranici je použit sled čísel z jiného chromozomu, než před hranicí.



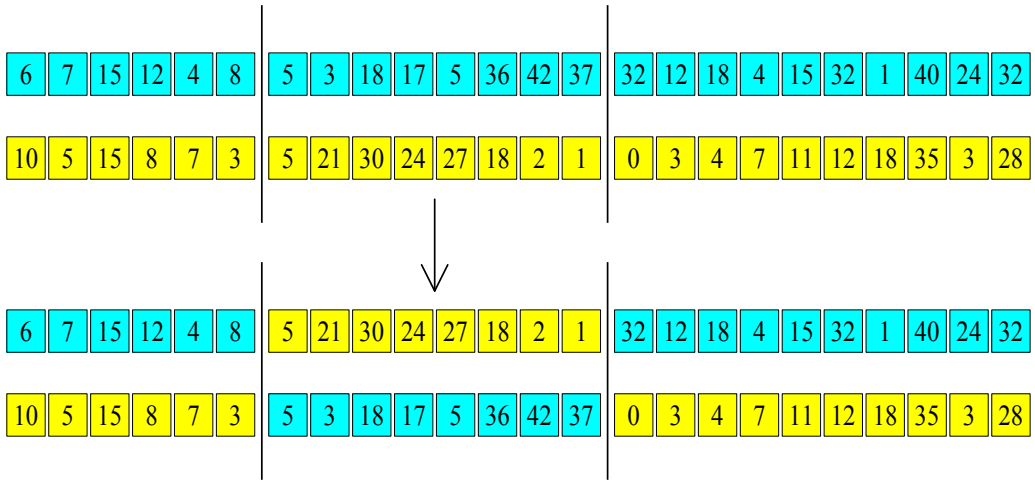
Obrázek 3.1: Jednobodové křížení

V závislosti na řešené úloze je možné implementovat jakékoliv jiné principy křížení. Ukázky jsou demonstrovány na dvou jedincích (chromozomech), nicméně je možné použít i více než dva jedince (chromozomy).

3.3.3.4. Operátor mutace

Mutace je náhodná změna jednoho nebo několika genů v chromozomu. Jejím úkolem je dodat do populace genetickou informaci, která v ní chybí. To může být způsobeno vyhynutím určitého druhu jedinců, nebo tím, že určitý fenotyp v populaci nebyl už při inicializaci populace.

Právě pomocí mutace má genetický algoritmus schopnost prohledávat prostor možných řešení.



Obrázek 3.2: Dvoubodové křížení

3.3.3.5. Elitismus

Elitismus úzce souvisí se selekcí a operátory. Během náhodného výběru jedinců pomocí selekce se může stát, že nejlepší jedinec v populaci nebude vybrán. Případně jedinec vybrán být může, ale pomocí operátoru křížení nebo mutace je jeho kvalita snížena.

Pokud chceme, aby nejlepší jedinec nebo skupina jedinců přešla vždy do další iterace algoritmu, zapínáme tzv. elitismus, který toto zajistí.

Elitismus je vhodné použít v případě, když pracujeme s menší populaci. Tedy tehdy, pokud je menší pravděpodobnost průchodu nejlepšího jedince bez modifikací. Nevýhoda elitismu je, že díky propagaci určitého řešení problému, může podporovat uvíznutí algoritmu v lokálním extrému.

Náhrada elitismu je paměť, do které se ukládají nejlepší jedinci ze všech epoch a i při horším celkovém výsledku algoritmu je pak možné na konci získat nejlepšího jedince.

3.3.4. Parametry algoritmu

Při spuštění procesu jsou nastaveny určité hodnoty parametrů, které lze nalézt u všech variací evolučních metod. Tyto parametry jsou:

3.3.4.1. Velikost populace

Určuje počet jedinců, kteří budou dostupní v každé populaci. Čím větší populace, tím větší prostor možných řešení je pokryt a o to je rozmanitější výběr jedinců pro operaci křížení. Nevýhodou je nutnost ohodnotit v každé iteraci více jedinců, což prodlužuje značně celý proces.

Velikost populace je dobré nastavit podle odhadu velikosti prohledávaného prostoru. Pokud máme například binární chromozom s deseti geny; máme $2^{10} = 1024$ kombinací čísel 1 a 0. Nastavením velikosti populace na hodnotu 1000 bychom tedy s velkou pravděpodobností získali i jedince, který představuje globální extrém.

Proces optimalizace by tak ztratil smysl, jelikož by se jednalo o hledání optima hrubou silou.

3.3.4.2. Pravděpodobnost křížení

Určuje, s jakou pravděpodobností budou jedinci vybraní ke křížení produkovat potomka nebo potomky. Pokud ke křížení nedojde, vybraní jedinci obvykle vstoupí do nové populace beze změny. Pravděpodobnost křížení se obvykle nastavuje poměrně vysoká – okolo 80 %.

Pokud bychom nastavili pravděpodobnost křížení příliš malou, nedocházelo by v populaci ke změnám a mnoho jedinců by jen migrovalo z populace do populace.

3.3.4.3. Pravděpodobnost mutace

Pokud jedinci vybraní ke křížení vytvoří potomka nebo potomky, bude tento potomek náhodně pozměněn. Pravděpodobnost křížení se nastavuje obvykle nízká – do 10 %.

Pokud bychom nastavili pravděpodobnost mutace příliš velkou, mohlo by dojít k destrukci mnoha kvalitních jedinců náhodnou mutací. Úloha mutace není hledat kvalitnější jedince, ale zanášet do populace nové části genů, které v ní nejsou přítomny a ty pak použít při křížení.

Pravděpodobnost křížení a mutace můžeme během celého procesu křížení měnit. Míru mutace lze zvyšovat, pokud v populaci nedochází ke zlepšování hodnoty fitness (algoritmus mohl uvíznout v lokálním extrému). Tento postup se ovšem nedoporučuje.

3.3.4.4. Ukončovací podmínka

Důležitým parametrem je i ukončovací podmínka. Obecně lze algoritmus ukončit, pokud je dosažena určitá kvalita jedinců, nebo pokud uběhl určitý počet iterací algoritmu. Po ukončení běhu je obvykle vrácen jako řešení úlohy nejlepší jedinec z populace, případně několik dalších kvalitních jedinců.

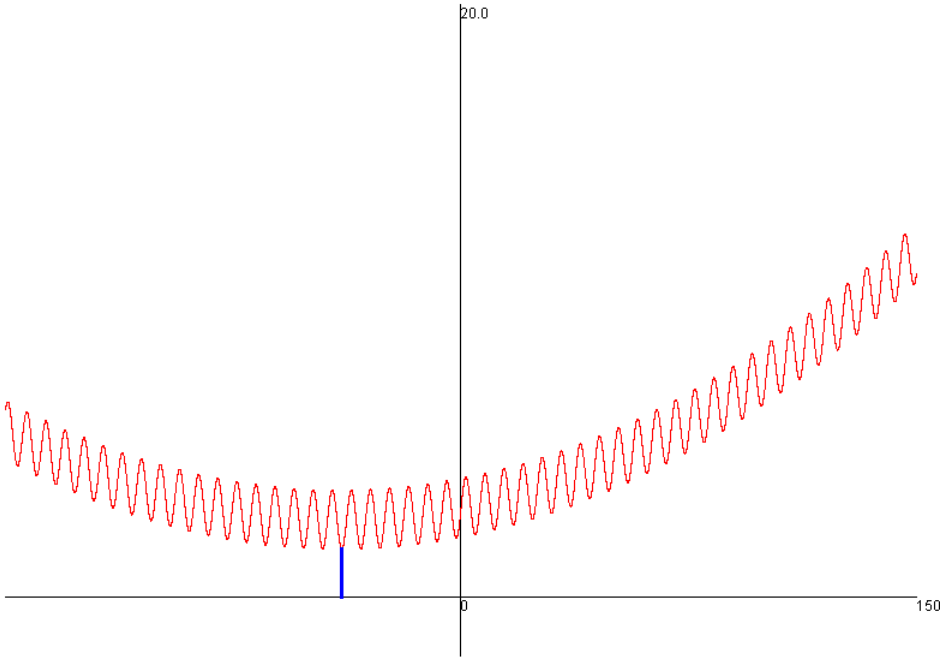
3.4. Běh algoritmu – ukázka

V této kapitole bude ukázán průběh řešení úlohy pomocí evoluční metody na jednoduchém příkladu hledání minima funkce. Tato úloha se často používá k demonstraci evoluční metody, implementace je vlastní.

3.4.1. Zadání úlohy

Pomocí evolučního algoritmu máme za úkol nalézt minimum funkce dané vzorcem 3.9 v intervalu hodnot $x \in$ <-150, 150>. Funkce je zobrazena na obrázku 3.3 modře je vyznačeno minimum nalezené pomocí evoluční metody.

$$f(x) = \frac{x^2}{4000} + sin(x) + \frac{x}{50} + 3 \tag{3.9}$$



Obrázek 3.3: Graf funkce a minimum nalezené evolučním algoritmem. Obrázek byl pořízen přímo v testovacím prostředí.

3.4.2. Analýza problému

Použijeme binárně kódovaný chromozom a Grayův kód. Důležitá je volba počtu genů (bitů). Prostor od spodní do horní hranice rozdělíme na dílky, které budou kódovat hodnotu x. Levý dílek, bude kódovat hodnotu -150 a pravý hodnotu 150. Po dekódování binárního chromozomu získáme pozici dílku, který představuje odpovídající jedinec.

Zadaný interval má 300 dílků, pokud jej rozdělíme po celých číslech. Pokud bychom použili například 8 genů (8 bitů, tedy bajt), dostaneme $2^8 = 256$, což je málo. Jelikož má tvar funkce charakter periodické funkce, bude vhodné nastavit chromozom alespoň tak, aby dokázal nalézt minimum s přesností alespoň na dvě desetinná místa. To znamená, že nutná rozlišovací schopnost je minimálně 30 000 dílků. Můžeme použít tedy například 16 bitů. Chromozom bude mít 16 genů, interval bude tedy rozdělen na $2^16 = 65536$ dílků.

Fitness hodnotu pro každého jedince vypočítáme pomocí vzorce 3.10. Čím menší je hodnota y, tím vyšší je hodnota fitness funkce. Maximální hodnota funkce je 1. Předpokládáme, že funkce má v intervalu, kde hledáme minimum, jen kladné hodnoty.

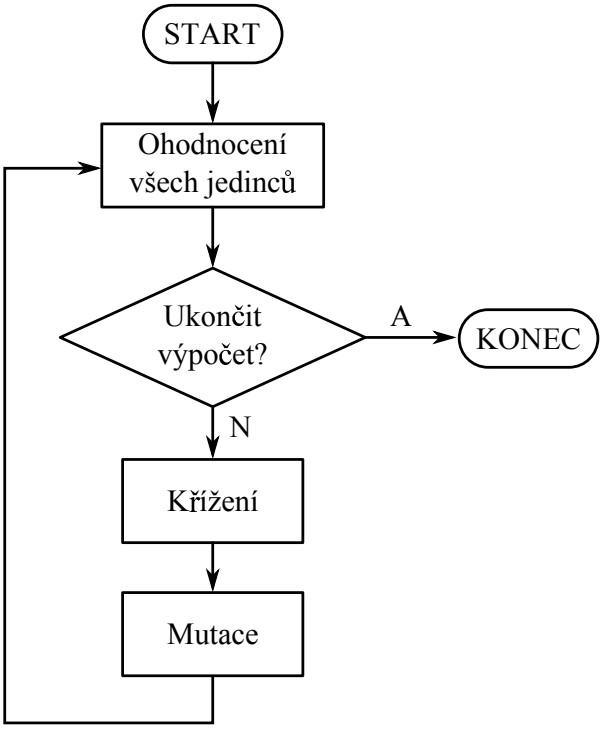
$$Fit(M) = \frac{1}{1+y} \tag{3.10}$$

3.4.3. Inicializace a parametry

Jako velikost populace byla zvolena hodnota 50 jedinců. Inicializace spočívá v prvotním naplnění populace jedinců. Obvykle jsou jedinci inicializováni náhodně, ale můžeme i cíleně generovat jedince, kteří po překladu genotypu na fenotyp mají nějakou konkrétní hodnotu. To může zrychlit hledání řešení. V tomto příkladu byla zvolena náhodná inici-

alizace. Pravděpodobnost křížení byla nastavena na 90 % a pravděpodobnost mutace na 10 %. Elitismus nebyl aktivován.

3.4.4. Iterativní hledání řešení



Obrázek 3.4: Průběh evolučního algoritmu

Genetické algoritmy fungují iterativně. Na obrázku 3.4 je znázorněn průběh evolučního algoritmu, který se opakuje dokud není splněna ukončovací podmínka.

Pro křížení jedinců je použit postup dvoubodového křížení. Na obrázku 3.5 je naznačen princip – jsou vybráni dva rodiče (modrý a zelený), jeden možný výsledek křížení je fialový jedinec.

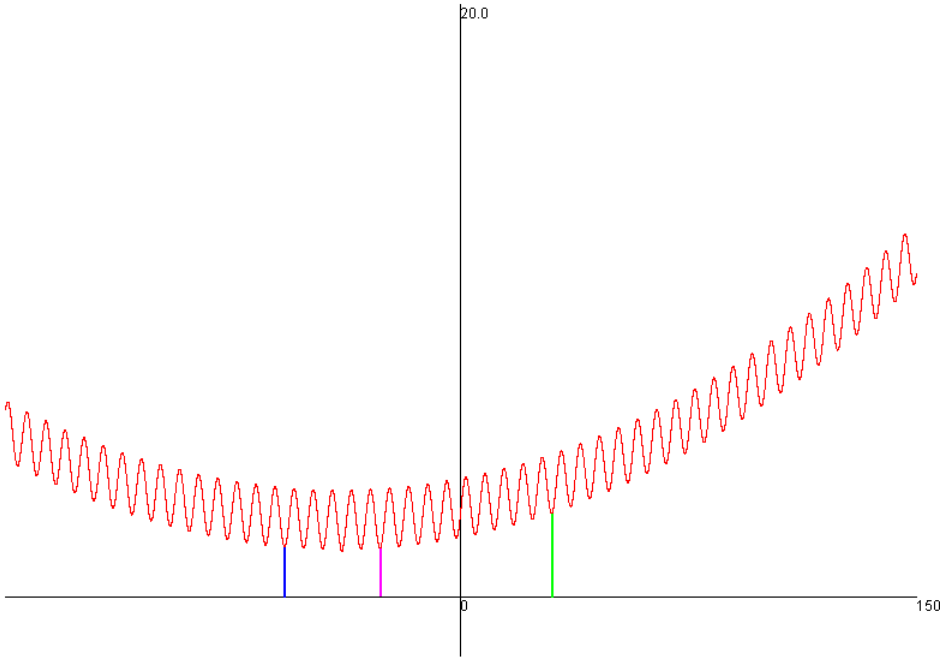
Průběh hodnot fitness funkce nejlepšího jedince (červeně) a průměrné hodnoty fitness funkce v populaci (zeleně) je znázorněn na obrázku 3.6.

3.4.5. Ukončení a výsledek

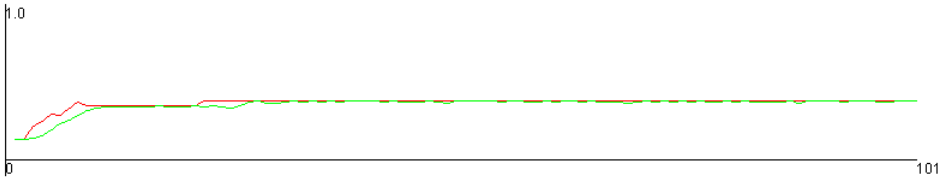
V tomto případě bude evoluční metoda ukončena, pokud proběhne nastavený počet iterací. Byla nastavena hodnota 100 iterací. Z průběhu fitness funkce je vidět, že finální řešení bylo nalezeno asi po 40ti iteracích. Minimum bylo nalezeno na hodnotě $x = -39,27$, funkční hodnota $f(-39,27) = 1.600$. Hodnota fitness funkce byla 0,384.

3.5. Gramatická evoluce

Gramatická evoluce [37] je jedna z variant evolučních metod. Její vznik se datuje k roku 1998. Patří do širší oblasti genetického programování [18]. Používá se tehdy, když je



Obrázek 3.5: Možný průběh křížení – rodiče (zelený a modrý) a potomek (fialový)



Obrázek 3.6: Graf hodnoty fitness. Obrázek byl pořízen přímo v testovacím prostředí.

řešení problému možné reprezentovat například jako krátký počítačový program nebo matematický vzorec.

Velkou výhodou je vložení mezivrstvy mezi genetický algoritmus a doménu řešeného problému, kterou představuje gramatika. V gramatice nacházíme symboly, které se vztahují k řešení daného problému - to mohou být například matematické funkce jako *sin* nebo operátory $+$, -, $\div$, $\times$ apod. Tyto symboly jsou vázány na řešený problém a jejich význam je nutné interpretovat pomocí překladače. Jsou ovšem definovány ve standardní struktuře, kterou používáme pro překlad chromozomu podle pravidel, které z něj získáme.

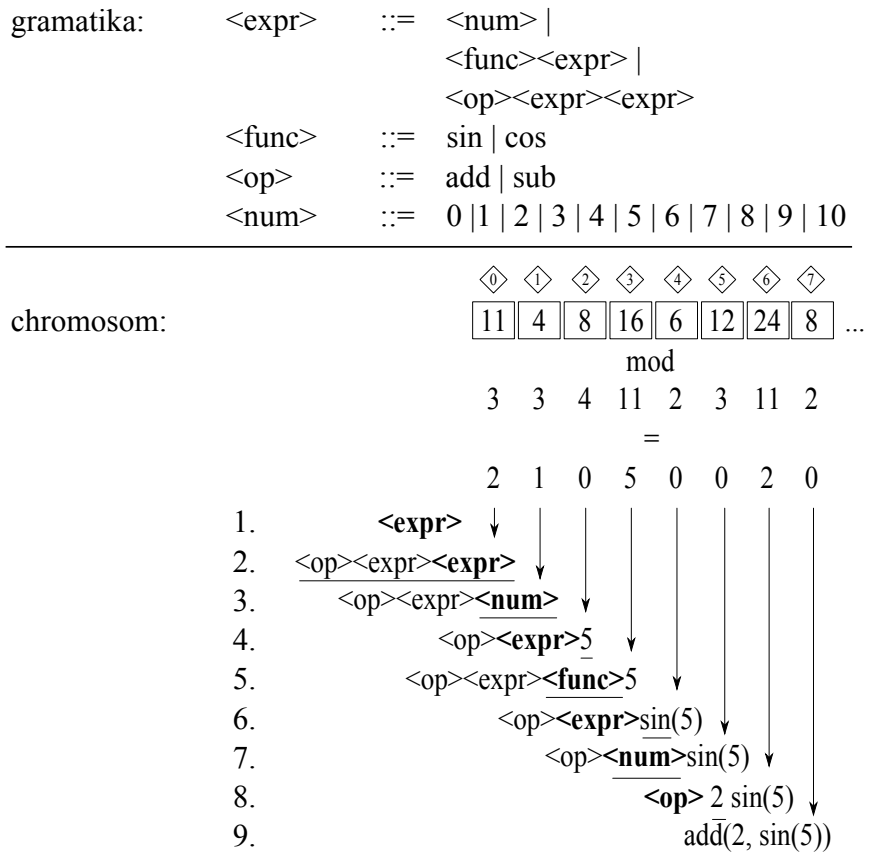
3.5.1. Gramatiky

Gramatická evoluce používá bezkontextovou přepisovací gramatiku, což je množina $G = \{N, T, P, S\}$ [15]. Kde N je množina neterminálních symbolů, T je množina terminálních symbolů, P jsou pravidla pro přepisování symbolů z množiny N na symboly z množiny T. Poslední symbol S je počáteční neterminální symbol.

Přepisovací pravidla P mají tvar $x \to y$. Kde $x \in N$ a $y \in \{N \cup T\}$. Pravidla $x \to y$ a $x \to z$ mohou být zapsána formou $x \to y \mid z$.

3.5.1.1. Překlad chromozomu

Překlad chromozomu spočívá ve volbě pravidel přepisu z množiny P na základě číselné hodnoty uložené v chromozomu.



Obrázek 3.7: Průběh překladu chromozomu pomocí gramatiky

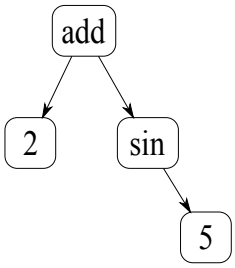
Překlad gramatiky pomocí zpětného překladu je zobrazen na obrázku 3.7. Tučně vyznačené neterminální symboly budou transformovány v dalším kroku překladu (načítají se od konce pracovního řetězce). Podtržené symboly jsou výsledkem transformace v aktuálním kroku.

Zpětný překlad pracuje na principu přidávání nepřeložených symbolů do zásobníku LIFO (*Last In First Out*).

V chromozomu jsou uložena různá čísla, abychom získali informaci o tom, který přepis použít, používáme operaci *modulo* (zbytek po dělení). Každý neterminální symbol má v gramatice určitý počet možností, kterými může být nahrazen. Provedeme tedy výpočet $hodnotaGenu \ mod \ pocetMoznosti = index$. Hodnota *index* je potom použita pro výběr přepisu – počítá se od nuly.

Na obrázku 3.7 je pomocí číslice 11 zvolen přepis neterminálu $<expr>$ na sekvenci $<op><expr><expr>$ jelikož zbytek po dělení 11 počtem voleb, což jsou 3, je 2.

Pokud má neterminál jen jeden možný přepis, není nutné použít číslici z chromozomu. Nicméně je vhodné optimalizovat strukturu gramatiky.



Obrázek 3.8: Výsledek překladu chromozomu – spustitelný program se stromovou strukturou

Na obrázku 3.8 je výsledek překladu chromozomu z obrázku 3.7, je reprezentován jako stromový graf, jehož uzly jsou terminální symboly gramatiky. Listové uzly jsou obvykle konstanty nebo jiné vstupní hodnoty. Ostatní uzly jsou operace.

3.5.2. Operátory

Operátory jsou spouštěny nad jedinci populace. Pomocí nich jsou jedinci cíleně modifikováni, aby algoritmus mohl prohledávat prostor řešení. U gramatické evoluce jsou základní operátory (křížení a mutace) pevně navrženy.

3.5.2.1. Křížení

Křížení u gramatické evoluce je založeno na výměně informací o struktuře programu mezi dvěma jedinci. Obvykle je u obou jedinců, kteří byli vybráni ke křížení, zvolen náhodný podstrom a tyto podstromy jsou potom vyměněny.

Princip křížení je znázorněn na obrázku 3.9. Implementace křížení může být realizována buď překladem jedince, následné úpravy fenotypu obou jedinců a opětovného zakódování do číselného chromozomu. Rychlejší je ovšem použít tzv. značkovací vektor a vyměnit pouze části číselného chromozomu bez překladu na fenotyp [29, 30].

V jakékoliv implementaci je důležité dodržet ještě podmínku, která zajistí, abychom nevyměnili nekompatibilní terminální symboly. Je tedy nutné, aby kořeny obou vyměňovaných podstromů patřily do množiny přepisů jednoho přepisovacího pravidla.

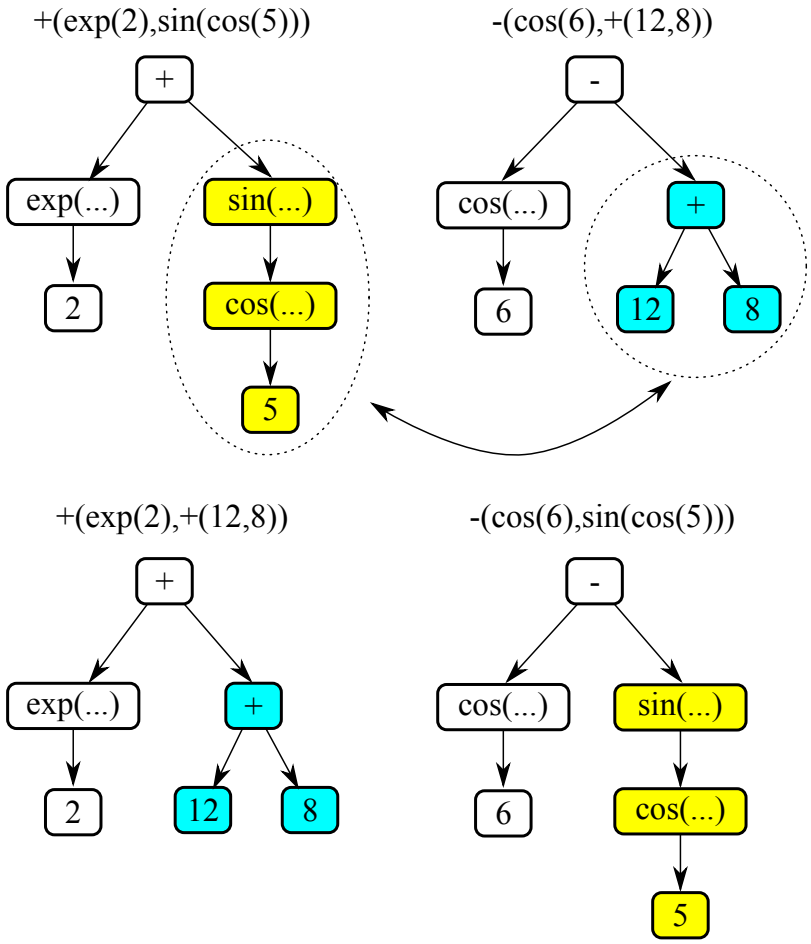
Pokud by totiž jeden uzel podstromu měl jako výstup například tříprvkový vektor, nejde ho vyměnit s uzlem, který má jako výstup skalární hodnotu.

3.5.2.2. Mutace

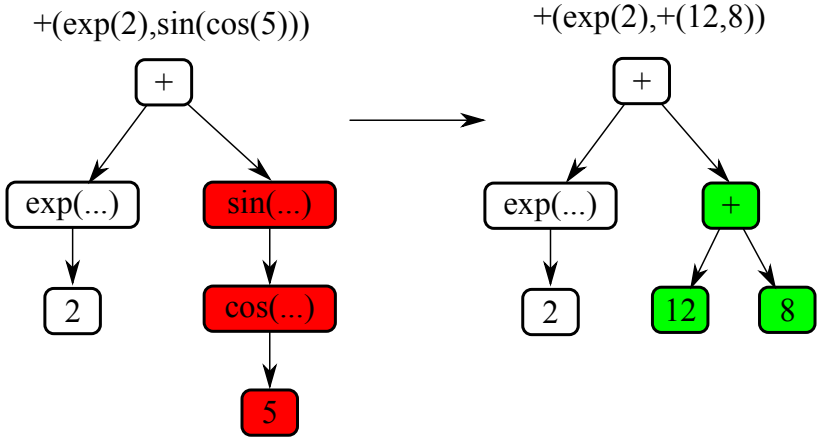
Mutaci lze u gramatické evoluce rozdělit na strukturální a nestrukturální. Rozdíl je v tom, zda měníme strukturu podstromu nebo jen některý terminální symbol. Princip mutace je popsán na obrázku 3.10 a 3.11.

3.5.3. Dvoufázová gramatická evoluce

Pro další zlepšení výsledků evoluční metody byla navržena tzv. dvoufázová gramatická evoluce [30]. Tento princip nechává gramatickou evoluci generovat pouze strukturu řešení



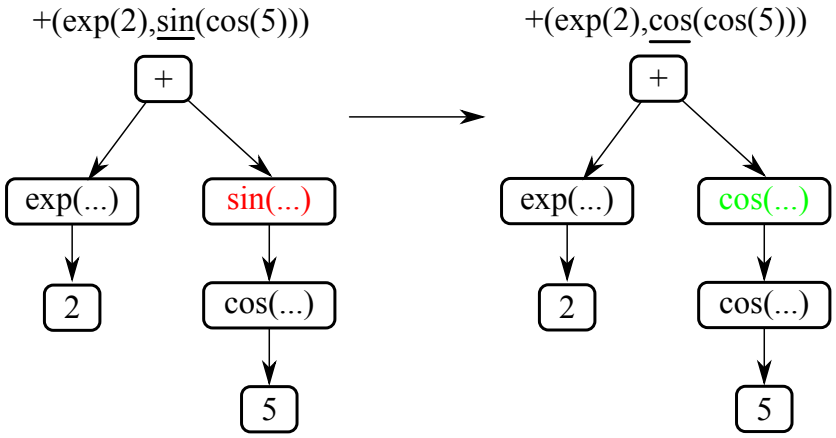
Obrázek 3.9: Princip křížení výměnou podstromu u gramatické evoluce



Obrázek 3.10: Princip strukturální mutace u gramatické evoluce

bez konstant (gramatika není schopná vůbec generovat číslice). V gramatice jsou přítomny speciální terminály, které reprezentují konstantu zatím blíže neurčené hodnoty.

Po sestavení struktury programu je spuštěna druhá fáze, která slouží k nalezení hodnot konstant pomocí procesu diferenciální evoluce.



Obrázek 3.11: Princip nestrukturální mutace u gramatické evoluce

Nevýhodou tohoto přístupu je značné zpomalení procesu, jelikož každý jedinec získá další populaci několika jedinců pro diferenciální evoluci. Každý tento jedinec pak potřebuje znát hodnotu fitness, která musí být vypočítána v každé iteraci diferenciální evoluce pro hlavního jedince. Záleží pak na počtu jedinců a počtu iterací při optimalizaci konstant.

Příklad: mějme populaci se 100 jedinci, kde každému z nich budeme hledat hodnoty jeho konstant pomocí diferenciální evoluce s použitím 30 jedinců a 50 iterací. Pak budeme muset pro každého jedince ve fázi hledání konstant vypočítat fitness hodnotu $30 \cdot 50 = 1500$-krát. Pokud toto číslo vynásobíme počtem jedinců v hlavní populaci, dostaneme hodnotu 150 000. Při jednofázovém postupu bychom počítali fitness hodnotu pouze stokrát za jednu iteraci.

3.5.4. Nastavování vah hranám programového stromu

Další možnost je nastavování vah hran ve stromovém programu. Touho vahou je pak vynásoben výstup odpovídajícího uzlu na konci hrany. Opět může být provedeno pomocí diferenciální evoluce. Tento postup se hodí tehdy, pokud přímo v gramatice není možné generovat konstanty (řešený problém to z libovolných nedovoluje).

Tento postup je možné použít k prořezání stromu programu, jelikož hrany s vahou 0 není nutné zachovávat a jejich podstrom je možné odstranit. Případně je není nutné vyhodnocovat, pokud chceme podstrom zachovat pro potřeby operace křížení, což také zrychlí běh vygenerovaného programu.

3.6. Výhody a nevýhody evolučních metod

At' už je řešen jakýkoliv problém, je vždy nutné zvolit správné nástroje k dosažení požadovaného cíle. Evoluční metody je obecně vhodné nasadit na řešení optimalizačních problémů – tedy takových problémů, kde dokážeme přímo říci, jaký cíl nebo cíle chceme dosáhnout a matematicky formulovat toto kritérium do fitness funkce.

3.6.1. Hlavní výhody genetických algoritmů

Široká oblast využití
Hlavní výhodou genetických algoritmů nebo gramatické evoluce je, že tyto metody mohou být nasazeny na řešení problémů, které mají složité nebo neznámé řešení. To znamená, že například neexistuje konkrétní algoritmus pro řešení dané úlohy. Evoluční metoda poskytuje jakýsi rámcový návod, jak úlohu převést na iterativní proces tvorby nových řešení pomocí operátorů křížení a mutace nad populací jedinců.

Velký prohledaný prostor
Další velká výhoda je schopnost evolučních metod prozkoumat velmi obsáhlý prostor možných řešení a nalézt řešení netradiční, které by člověka řešícího danou úlohu nemuselo napadnout.

Optimalizace struktury
U gramatické evoluce konkrétně, je velkou výhodou možnost optimalizace struktury výsledného počítačového programu. Tuto schopnost nemají neuronové sítě, kde je struktura daná dopředu.

3.6.2. Hlavní nevýhody genetických algoritmů

Genetické algoritmy mají samozřejmě i nevýhody, které hovoří proti jejich použití. Před nasazením evolučních metod je nutné rozhodnout, zda výhody převažují nad nevýhodami.

Časová náročnost
Použití metody nemusí přinést nutně nejrychlejší řešení. Z toho jsou navrhovány speciální algoritmy. Zde velmi záleží na evoluční použití metody. Například v případě gramatické evoluce je proces hledání řešení obvykle velmi pomalý, ale výsledné řešení (krátký počítačový program) může pracovat ve výsledku velmi rychle.

Jiný pohled na časovou náročnost je, že při použití evoluční metody můžeme ušetřit čas nutný pro vývoj specifického algoritmu.

Obecně lze říci, že nejdéle na evolučních algoritmech trvá ohodnocení všech jedinců v populaci. Podle složitosti úkolu trvá ohodnocení jedince až několik sekund. Pokud vynásobíme čas potřebný pro ohodnocení jednoho jedince jejich počtem, dostaneme mnohdy poměrně velkou hodnotu. Jediný způsob jak tuto hodnotu snížit, je použít k ohodnocování jedinců více výpočetních vláken. Tohoto lze dosáhnout použitím více výpočetních jader procesoru nebo grafické karty, případně distribuovat výpočty na další samostatné počítače.

Konkrétně u gramatické evoluce je možné dále optimalizovat nejen strukturu, ale i parametry (konstanty použité v programu, případně lze nastavovat váhy hranám programového stromu). K optimalizaci konstant se opět může použít evoluční metoda – diferenciální evoluce – popsáno v 3.5.3. Každá iterace opět vyžaduje ohodnocení všech jedinců funkcí fitness a tím dále zpomaluje proces.

Lokální extrémy

Jako jiné optimalizační algoritmy, mohou i genetické algoritmy během svého běhu být dovedeny do takzvaného lokálního extrému. Je to stav, kdy se v populaci najde jedinec, který odpovídá fitness funkci lépe, než ostatní jedinci. Díky tomuto se začne genetická informace uložená v něm šířit do populace z důvodu jeho větší úspěšnosti v procesu selekce. Pokud je špatně nastavena míra mutace, nemusí algoritmus tento stav překonat, jelikož při malých změnách v genotypu jedince neopustíme oblast kolem lokálního extrému.

Zabránit tomuto stavu by měla kontrola míry diversity populace, která by měla zajistit dostatečnou variabilitu genotypů jedinců.

Další vliv, který může tento fenomén podporovat, je zapnutí tzv. elitismu (popsáno v 3.3.3.5).

Nutnost navrhnout správně reprezentaci jedince

Velkou výzvou je u obecných evolučních metod návrh genotypu jedince a jeho interpretace. Často se využívá číselný chromozom. Jeho efektivní interpretace je potom velmi důležitá. Je nutné zajistit, aby při malé změně chromozomu, například mutací, byla i změna fenotypu jedince malá. Dobrým příkladem je použití Grayova kódu místo obyčejného binárního kódování.

U gramatické evoluce je tento problém odstraněn použitím gramatiky. Další vylepšení pak přináší značkovací vektor a rozdělení mutace na strukturální a nestrukturální [31].

4. Navržené algoritmy

Hlavním cílem této práce navrhnout aplikaci evolučních metod, konkrétně gramatické evoluce, v oblasti počítačového vidění – rozpoznávání objektů. Jinými slovy to znamená použití gramatické evoluce k tvorbě klasifikátoru. Jde tedy o proces učení a výsledkem by měl být systém, který by mohl pracovat podobně jako neuronová síť.

Samotné evoluční metody nejsou přímo vhodné jako ekvivalent neuronové sítě. Běh evoluční metody je iterativní optimalizační proces, podobající se spíše procesu učení neuronové sítě.

Neznamená to, že by evoluční metoda nebyla použitelná k rozpoznávání objektů. Pomocí evoluční metody lze například generovat polohu objektu a pomocí fitness kontrolovat míru překrytí kontury z databáze s hranou detekovanou hranovým detektorem. Evoluční algoritmus ovšem nezaručuje vždy stejný průběh a konvergence ke globálnímu extrému také není vždy zaručena. Problémem evolučních metod tedy je, že nezaručují opakovaně stejné hodnoty na výstupu při stejných hodnotách na vstupu.

Další nevýhodou je nutnost ohodnocování velkého množství jedinců funkcí pro výpočet fitness hodnoty. Tato akce může hlavně u aplikací počítačového vidění trvat dlouho. Postavení evoluční metody jako alternativy k neuronové síti v produkční fázi se tedy nejeví jako dobrý nápad.

Je proto vhodnější genetický algoritmus použít ke tvorbě klasifikačního nástroje. Zde se nabízí použití právě gramatické evoluce, která je schopná vytvářet stromové programy. Struktura, terminály gramatiky a konstanty použité k sestavení stromu se dají považovat za znalosti. Při použití vhodné funkce pro výpočet fitness hodnoty je pak možné řídit optimalizační proces k tvorbě výkonných klasifikátorů.

Předpokládaná aplikace metod je v průmyslu, například na automatické výrobní lince, kde hlavními požadavky jsou spolehlivost a rychlost rozpoznání.

4.1. Požadavky

Z předchozího textu vyplývá, že hlavním úkolem bude nalezení způsobu, jak pomocí evolučních metod vytvářet klasifikátory schopné rozpoznávat předložené objekty.

Hlavním požadavkem je univerzálnost použití navržené metody. Univerzálností se myslí, že metoda bude použitelná nejen pro rozpoznávání objektů v obraze, ale k rozpoznávání jakýchkoliv objektů, které lze popsat číselně a tyto data zpracovat v počítači. I toto má samozřejmě svá omezení – schopnost úspěšného řešení problému pomocí gramatické evoluce vždy závisí na prostředcích, které dáme optimalizačnímu procesu k dispozici. Je to jednak počet iterací a jedinců v populaci, ale hlavně sada zvolených terminálů.

Dalším vytyčeným požadavkem bude rychlost, která by měla být srovnatelná s neuronovou sítí.

V neposlední řadě pak spolehlivost. Je samozřejmé, že v každé aplikaci počítačového vidění záleží na nastavených podmínkách, nicméně není přípustné, aby měl klasifikátor jinou odezvu pro stejná vstupní data. Proto byl zavržen přístup s aktivní fází založenou na evoluční metodě.

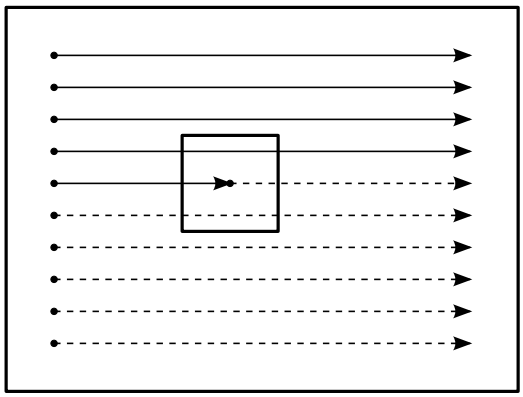
4.2. Zvolená architektura

Jak již bylo řečeno, v práci je použita gramatická evoluce. Klasifikátorem tedy bude program vytvořený tímto postupem. Evoluční metoda bude použita v učící fázi. Dalším krokem je zvolit, jak budou reprezentovány učící a testovací vzory.

Bylo zvoleno, že se bude pracovat s obrazem v odstínech šedi.

4.2.1. Princip posuvného okna

Universálním přístupem je přímé předkládání částí obrazu klasifikátoru. Tzv. posuvné okno je jakýsi výřez z velké obrazové matice. Na obrázku 4.3 je jedna z možností, jak posuvné okno prochází obrazovou matici. Hodnoty pixelů posuvného okna lze snadno dosadit jako vstupní hodnoty do klasifikátoru.



Obrázek 4.1: Vizualizace principu posuvného okna

Hlavními parametry posuvného okna jsou:

- velikost okna – ta by měla být větší než je největší hledaný objekt.
- krok – určuje jakou rychlostí se okno pohybuje po obraze. Krok lze adaptivně měnit, pokud klasifikátor napojený na okno detekuje objekt, je možné krok zmenšit a získat více informací z dané oblasti.

Nevýhodou tohoto přístupu je, že lze poměrně špatně získat invarianci vůči změně měřítka objektů. Subjektivně lze říci, že to je v mnoha aplikacích nepotřebná (kamera je umístěna v konstantní vzdálenosti od linky) nebo dokonce nežádoucí vlastnost (na jedné lince zpracováváme dva výrobky stejného tvaru, ale jiné velikosti).

Jediným způsobem jak posuvné okno dokáže rozpoznat objekty v jiných vzdálenostech je změna velikosti obrazu nebo okna a další průchod celého obrazu. Takový přístup je samozřejmě pomalý.

Spolehlivost, invariance vůči rotaci a stranovému převrácení je již v rukou nasazeného klasifikátoru.

V našem případě, budeme nastavovat velikost posuvného okna vždy jako liché číslo, abychom mohli snadno získat bod ve středu.

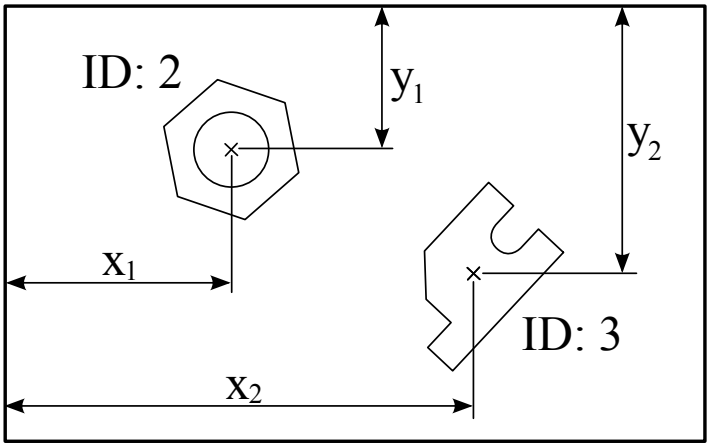
Posuvné okno není nejrychlejším přístupem. Rychlejší by jistě byla klasifikace na základě nějakého popisu objektů. Nicméně pokud bychom na stejném principu nasadili neuronovou síť, byl by výkon celého systému srovnatelný, jelikož provedení výpočtu pomocí neuronové sítě by probíhalo také pro každou pozici posuvného okna.

4.2.2. Vektorový nebo maticový vstup

Jako vstup bude tedy napojeno posuvné okno, pro univerzálnost je toto definováno jako matice. Pro jiné využití lze matici degradovat na vektor a použít úplně stejný postup pro tvorbu klasifikátoru.

4.2.3. Příklady pro učení a testování

Pro učení a testování metod je nutné nachystat i odpovídající data. Protože má jít o rozpoznávání objektů v obraze, je vstup obraz typu PNG nebo JPG s nepřekrývajícími se objekty v různých polohách. K těmto obrázkům je dále připojena informace o pozici objektů a jejich třídě. Vše je uloženo v textovém souboru. V obraze se teoreticky nemusí vyskytovat žádný objekt, je tam tedy pouze pozadí. V takovém případě v popisném souboru nic není – pozadí se nepopisuje.



Obrázek 4.2: Ukázka dat pro učení a testování

4.2.4. Vyhodnocení výsledků klasifikace

Vzhledem ke zvolenému přístupu s posuvným oknem, je pro každou pozici, na které se posuvné okno zastavilo, k dispozici jeden výsledek klasifikace. Výsledek klasifikace lze tedy získat analýzou shluků bodů. Nejprve se odfiltrují výsledky, kde je podle klasifikátoru pozadí. Poté by měly zůstat pouze samostatné skupiny bodů. Pro každou skupinu pak snadno určíme třídu objektu jako nejčastěji vyskytující se hodnotu (modus).

Případně lze nalézt těžiště shluku a jako výsledek klasifikace získat nejčastější hodnoty v jeho okolí. Tato metoda je vhodnější, pokud okolo objektů vzniká mnoho neurčitých výsledků.

4.3. Terminály gramatiky

Konkrétní gramatika, která by měla být předána procesu gramatické evoluce je závislá na doméně klasifikační úlohy. Pokud jde o rozpoznávání objektů v obraze s použitím posuvného okna, jak je popsáno v 4.2.1 a 4.2.2, bude určitě zapotřebí možnost pracovat s jednotlivými body obrazu. Proto je jasné, že v gramatice se musí objevit terminály, které dokážou číst oblasti v posuvném okně.

Dalšími vstupními terminály, pokud jde o obraz můžou být různé transformace, jako například prahovaný obraz nebo výstup některého hranového detektoru.

Abychom umožnili z těchto obrazových matic získat skalární hodnoty, budou nutné terminály pro výpočet základních operací jako suma, nebo průměr. Tyto mohou být kombinovány s terminály, které omezují regiony, nad kterými se odpovídající statistická operace vykoná.

Dále lze do gramatiky vložit terminály reprezentující základní matematické operace a funkce. Konkrétní gramatiky jsou v kapitole 6, kde se nachází testy metod.

4.4. Stromový program se skalárním výstupem

První variantou, která byla navržena, bylo použití stromového programu, který bude mít jako výstup jednu hodnotu. Inspirací byla práce [45]. Princip je takový, že výstup programu by měl mít určitou hodnotu, při předložení objektu určité třídy na vstup. Podle intervalu, do kterého hodnota patří se potom určila třída objektu na vstupu.

4.4.1. Přímá klasifikace

Jako první pokus s touto metodou byl tedy gramatickou evolucí vytvořen program, který přímo na obraze dokázal detekovat objekty. V následujícím odstavci je popsán princip získávání intervalů hodnot pro klasifikaci.

4.4.1.1. Statistické vyhodnocení výstupu programu

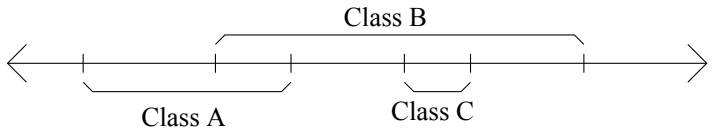
V citované práci [45] jsou intervaly pevně dané, což je poměrně tvrdá podmínka, která může proces gramatické evoluce zpomalit. Inovací v tomto případě byla automatická detekce intervalů.

Každý jedinec byl použit na trénovací množině obrazů. Podle výstupu programu v okolí známých objektů byl vypočten aritmetický průměr a směrodatná odchylka. Z těchto hodnot byly sestaveny hranice intervalů výstupu. Fitness funkce nehodnotila překrytí intervalů, ale pouze úspěšnost detekce. Jedinec, který měl vyhodnocené hranice intervalů tak, že se překrývaly, byl podle fitness funkcí hodnocen jako nekvalitní, jelikož prováděl chybné klasifikace.

Na obrázku 4.3 je naznačen problém s překrývajícími se intervaly.

4.4.2. Dvoufázová klasifikace

Výsledky první metody nebyly příliš dobré, a proto bylo nutné přístup upravit. Výstup programu byl často správný v okolí středu objektu, ale program chyboval v blízkém okolí a



Obrázek 4.3: Program se skalárním výstupem - překrývání tříd

na hranicích objektu. Proto bylo rozhodnuto o použití dvoufázového přístupu, kde prvním krokem bude detekce objektů v obraze a druhým krokem bude klasifikace pouze v místech označených detektorem.

4.4.2.1. Fáze 1: binární klasifikace – detekce objektů

Pro detekci byl použit opět program vytvořený gramatickou evolucí. Byl použit i přístup s automatickou volbou intervalů (jeden interval pro pozadí, druhý pro libovolný objekt).

Pro testovací objekty, které budou představeny v následující kapitole, by šlo použít místo detektoru i jednoduché prahování. Pro složitější objekty ve složitější scéně (například detekce obličejů na různých pozadích) je tento postup opodstatněný.

Detektor označil v obraze místa, kde byl zjištěn objekt. Klasifikátor byl spuštěn pouze v těchto bodech. Toto přineslo i poměrně velké zrychlení procesu klasifikace, neboť klasifikátor byl poměrně složitý program, jehož frekventované spouštění zabíralo mnoho času. Naproti tomu detektor byl velmi jednoduchý a rychlý.

Další urychlení procesu by bylo možné dosáhnout, pokud bychom detektor spouštěli s proměnlivým krokem okna. Tedy v oblasti, kde detektor reaguje pozitivně, bychom mohli krok posuvného okna zmenšit a v oblastech, kde nic není, zase zvětšit.

4.4.2.2. Fáze 2: klasifikace do více tříd

Druhá fáze již sloužila pouze k získání klasifikace objektů. Jak bylo uvedeno, klasifikátor se spouštěl pouze v označených místech. Zbytek metody byl stejný jako 4.4.1.

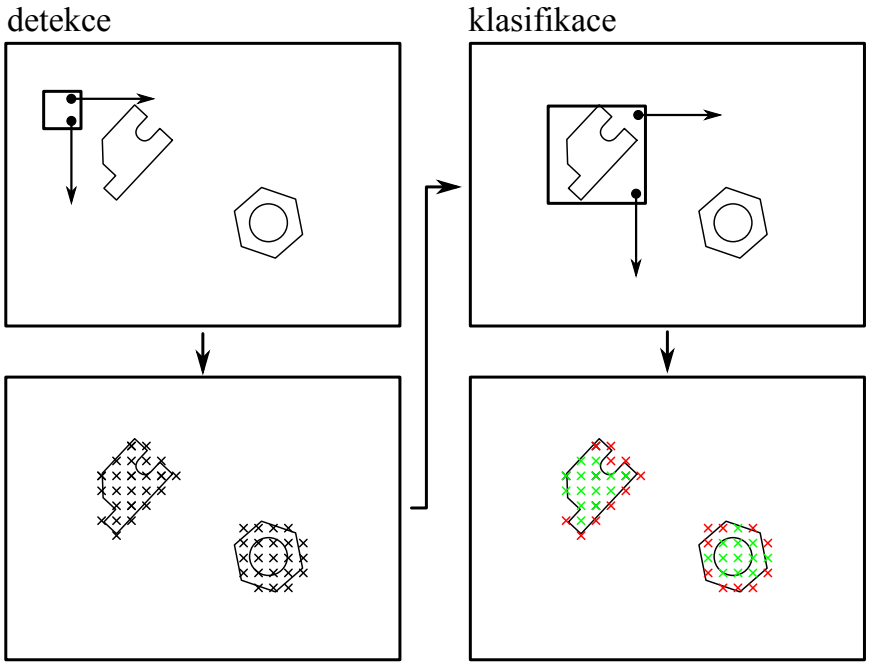
Na obrázku 4.4 je princip práce detektoru a klasifikátoru. Detektor označí jen body, kde najde objekt. Klasifikátor bude spuštěn jen na místech, která detektor označil. Zelené body znamenají shodu objektu nalezeného klasifikátorem a učícího vzoru. Červené body znamenají chybu klasifikace. Tím, že detektor nedovolí klasifikátoru hledat v místech, kde se v posuvném okně nachází například jen polovina objektu, klesne počet chybných klasifikací na minimum.

4.4.3. Výpočet funkce fitness

Pro klasifikátory byl vytvořen vzorec 4.1. Funkce pro výpočet hodnoty fitness zvyšuje svoji hodnotu s klesajícím počtem falešných positiv nebo negativ. Zároveň hodnota fitness funkce roste, pokud roste počet správně detekovaných bodů. Výsledek výpočtu leží v intervalu $< 0,\ 1 >$.

$$Fit(M) = \frac{C}{1 + C + FP + FN} \tag{4.1}$$

- C = počet správně detekovaných míst



Obrázek 4.4: Navrhované chování dvoufázové klasifikace – detektor označí objekty, klasifikátor dodá jejich třídu

- FP = počet falešných positiv (místa nesprávně označená jako objekt)
- FN = počet falešných negativ (místa, která měla být označená, ale nebyla)

Pokud byl použit dvoufázový přístup, výpočet fitness hodnoty pro detektor byl realizován vzorcem 4.2. V tomto vzorci je navíc část, která přidává požadavek na jednoduchost vytvořeného programu. Jinak je vzorec podobný vzorci 4.1, jen se nezajímá o počet správně detekovaných bodů. Výsledek výpočtu leží v intervalu $< 0,\ 1 >$.

$$Fit(M) = w_1 \cdot \frac{1}{1 + FP + FN} + w_2 \cdot \frac{1}{1 + NC} \tag{4.2}$$

- w_1, w_2 = váhy, nastaveny experimentálně
- FP = počet falešných positiv (místa nesprávně označená jako objekt)
- FN = počet falešných negativ (místa, která měla být označená, ale nebyla)
- NC = počet uzlů grafu (vyjadřuje složitost programu)

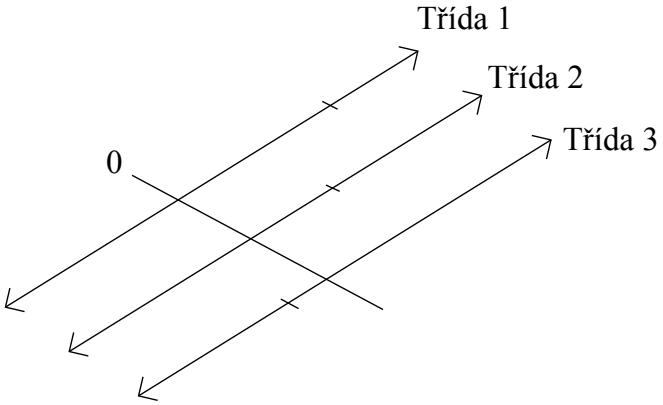
4.5. Stromový program simulující imperativní zápis s vektorovým výstupem

Velkou nevýhodou programů, které produkuje běžná gramatická evoluce je jednohodnotový výstup. Takový výstup vedl často k překrývání intervalu nebo nemožnosti vytvořit kvalitní klasifikátor. Proto byly navrženy terminály gramatiky, které dokážou napodobit chování klasických nestromových programů.

Na obrázku 4.5 je naznačen možný výstup programu, který má více než jednu výstupní hodnotu. U navrhovaných metod lze takový výstup získat, pokud použijeme pro výsledky výpočtů registry – proměnné, do kterých lze uložit výsledky výpočtů.

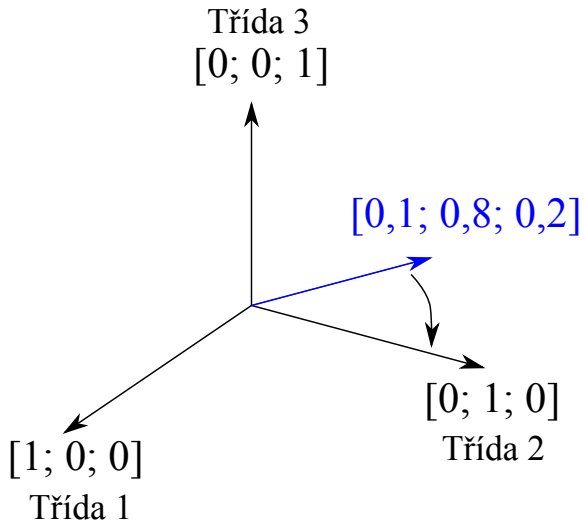
Dalším možným problémem byla u předchozí metody kombinace jednohodnotového výstupu a volby rozsahu intervalů až na základě výstupních hodnot programu. Jelikož měl každý jedinec jinou množinu intervalů, která nebyla kódována v chromozomu, nebylo možné jedince efektivně křížit a mutovat.

Výhoda tohoto přístupu proti předchozímu je taková, že jednotlivé výstupní hodnoty mohou mít v programech samostatné větve stromu a je možné tak programy efektivně kombinovat při křížení a mutaci.



Obrázek 4.5: Program s vektorovým výstupem

Vyhodnocení výstupu se pak nabízí například porovnáním výstupu, což je vektor s hodnotami, s vektory představujícími bázi vektorového prostoru o n dimenzích. Kde n je počet tříd, které klasifikujeme. Jednoduše pak lze vyhodnotit vzdálenost každého vektoru k bázovým vektorům prostoru. Každý bázový vektor je svázán s třídou objektů. Nejbližší bázový vektor je pak prohlášen za výstup. Postup je znázorněn na obrázku 4.6.



Obrázek 4.6: Porovnání výstupu klasifikátoru s bází vektorového prostoru

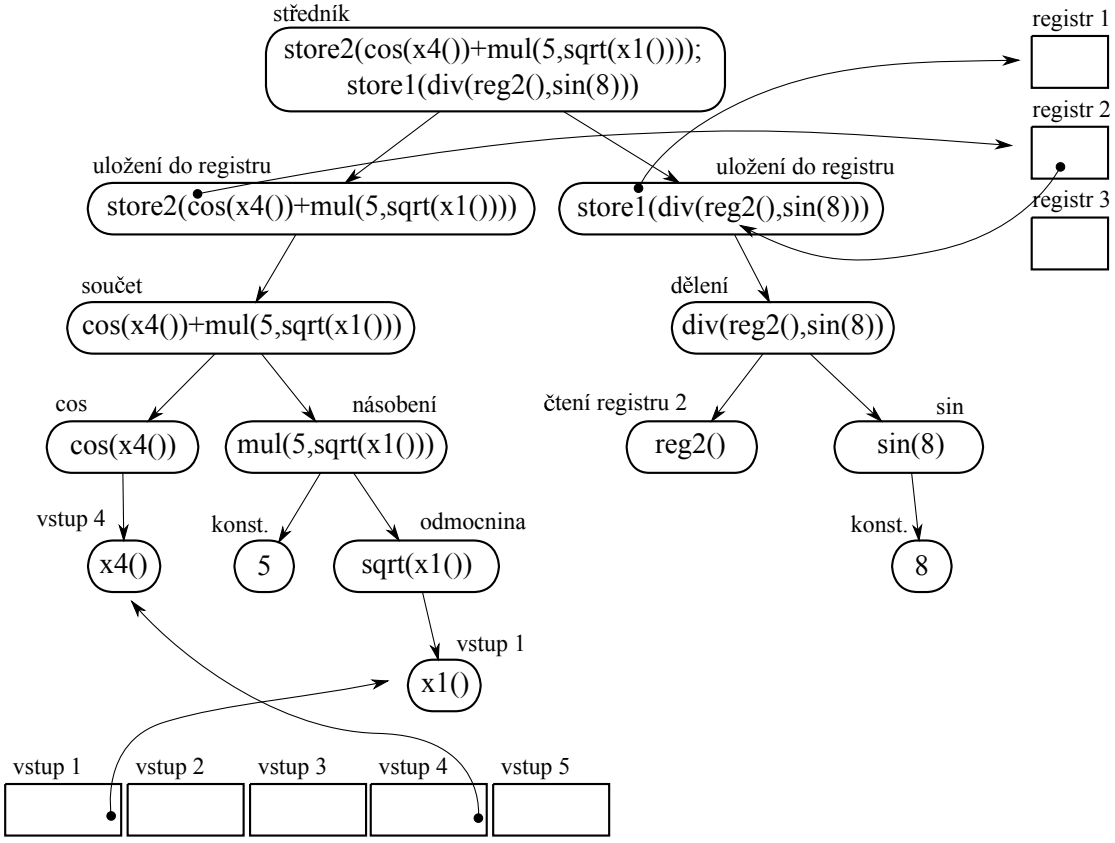
Takový výstup lze použít i k dalšímu zpracování v neuronové síti, nebo jen k transformaci obrazu, pokud bychom se pouze rozhodli, že použijeme gramatickou evoluci k vývoji obrazového filtru.

4.5.1. Registry

Terminální symboly gramatiky pro registry slouží k uchování hodnot mezivýpočtů a zároveň jsou používány k předání výsledků výpočtu programu. Realizovány jsou pomocí pole hodnot. Gramatická evoluce si může zvolit, který registr bude program modifikovat nebo číst.

4.5.2. Operátor středník

Terminál gramatiky středník umožňuje napodobit chování programů zapsaných v klasickém imperativním jazyku. Zachovává ovšem stromovou strukturu programů a tím i možnost velmi snadného křížení a mutace pomocí záměny podstromů programu.



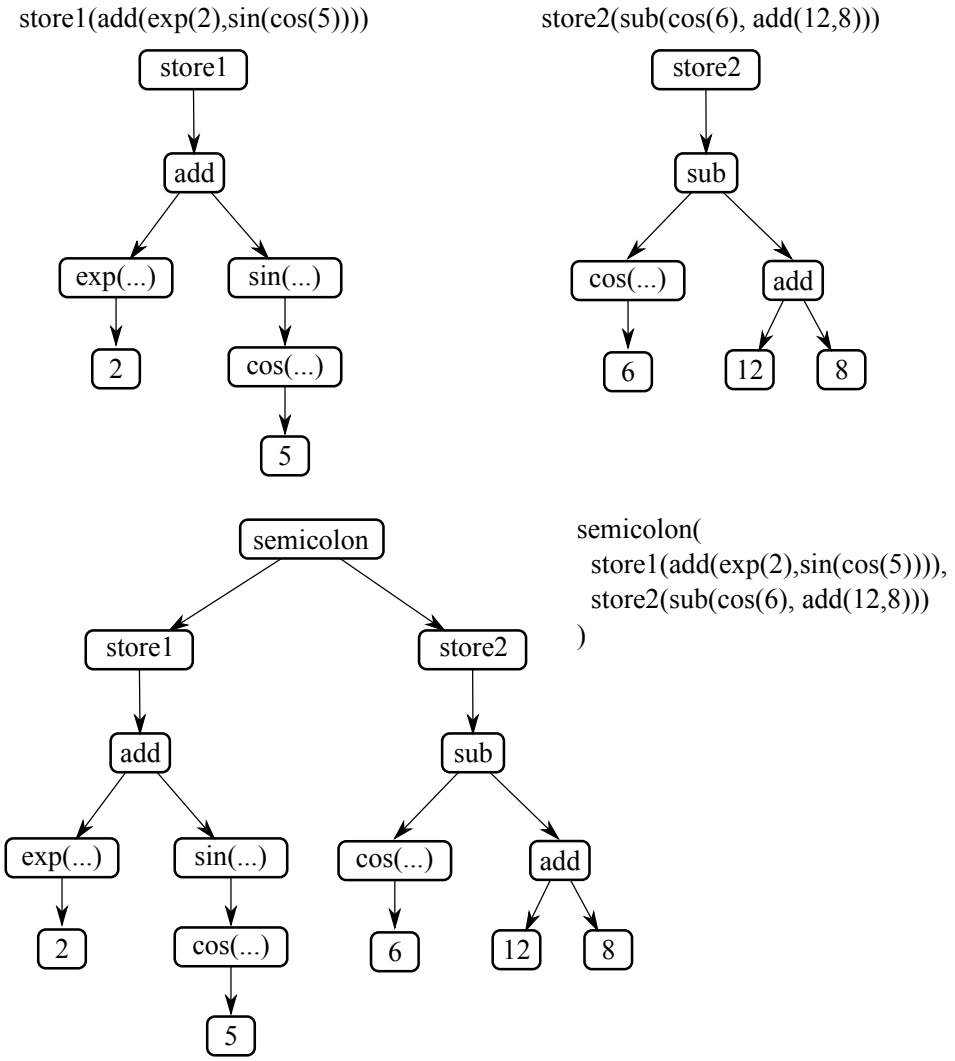
Obrázek 4.7: Struktura programu s uzlem středník, vektorovým vstupem a registry

Stromový graf programu používající operátor středník je na obrázku 4.7. Je zde demonstrován krátký program, který má přístup k pěti vstupním hodnotám a třem registrům. Program dokáže z registrů číst i do nich zapisovat.

I když je struktura programu stále stromová, jeho spuštění simuluje zápis programu *po řádcích*.

4.5.2.1. Modifikace operátoru křížení

S použitím středníku je možné zavést novou variantu křížení, kde se oba programy vybrané ke křížení spojí právě pomocí operátoru středník. Princip je vysvětlen na obrázku 4.8.



Obrázek 4.8: Křížení dvou programů pomocí operátoru středník

Tato operace je prováděna dodatečně po klasickém křížení. Na výstupu metody zodpovědné za křížení je tak o jednoho jedince více. Pravděpodobnost této operace se dá nastavit podobně jako pravděpodobnost standardního křížení nebo mutace.

Teoreticky by se takto dalo cíleně spojovat i více programů naráz. Implementace takovéhoto operátoru by ale byla složitější.

Cílem zavedení této operace bylo umožnit evoluční metodě slučovat klasifikátory. V počátku se v populaci vyskytovaly klasifikátory schopné najít jednu nebo dvě třídy objektů. Mnoho programů přitom umělo najít jiné objekty, než ostatní programy. Pomocí této operace mohly být takovéto programy snadno sloučeny.

Pokud by tato operace nebyla implementována, hrozilo by, že bude vybrán jeden klasifikátor, který umí nejlépe rozpoznat co největší množství tříd hned na začátku a jeho

fenotyp nakonec převládne v celé populaci (uvíznutí v lokálním extrému). Ostatní kvalitní klasifikátory by tak mohly vymizet z populace.

4.5.3. Použití pro klasifikaci

Běh programu je stejný jako bylo popsáno v 4.4.1 nebo 4.4.2. Rozdíl je pouze ve výstupu programu. Vyhodnocení výsledků posuvného okna je opět stejné podle 4.2.4.

Opět by i v tomto případě bylo možné, rozdělit detekci objektů a klasifikaci do dvou kroků. Pro lokalizaci objektů bychom mohli jistě využít vytvořené klasifikátory popsané v sekci 4.4.

4.5.4. Výpočet funkce fitness

Pro výpočet funkce fitness byl navržen vzorec 4.3. Tento vzorec má v sobě integrován požadavek na schopnost klasifikaci co největšího počtu tříd. Tato podmínka je vyjádřena ve fitness funkci, jelikož gramatika jako taková nemusí generovat program, který by dokázal rozpoznat všechny třídy. Naopak může být vygenerován program, který například rozpozná jen některé třídy, ale velice dobře. Hodnota *SR – Success Rate* pak může být vyšší pro takový program, než pro program, který umí rozpoznat více tříd, ale na horší úrovni. Je lepší mít program, který dokáže rozpoznat více tříd a ten upravit pomocí mutace a křížení k lepším výsledkům. Druhá část vzorce pro výpočet fitness přidává na hodnocení právě takovýmto programům. Výsledek výpočtu leží v intervalu $< 0,\ 1 >$.

$$Fit(M) = w_1 \cdot SR + w_2 \cdot \frac{RC}{CC} \tag{4.3}$$

- w_1, w_2 = váhy
- SR = míra úspěšnosti – samostatný vzorec (*Success Rate*)
- RC = počet rozpoznaných tříd (*Recognized Count*)
- CC = počet tříd, které mají být rozpoznány (*Class Count*)

Váhy w_1 a w_2 byly nastaveny podle vzorců 4.4 a 4.5. Rozdělují tak číslo 1 v poměru počet klasifikovaných tříd ku jedné.

$$w_1 = \frac{CC}{CC + 1} \tag{4.4}$$

$$w_2 = \frac{1}{CC + 1} \tag{4.5}$$

Hodnota SR může být vypočtena různými způsoby. Byl zvolen vzorec 4.6, který jednoduše počítá průměrnou hodnotu skóre za úspěch pro všechny třídy. Nezohledňuje se tak, pokud náhodou existuje více trénovacích vzorů v jedné třídě než v ostatních. Číslo n odpovídá počtu tříd.

$$SR = \frac{\sum_{i=1}^{n} \frac{S_i}{MS_i}}{n} \tag{4.6}$$

- S_i = skóre získané za detekce objektů třídy i
- MS_i = maximální skóre získatelné za detekce objektů třídy i

Úspěšná detekce je, pokud je výstup klasifikátoru vyhodnocen jako objekt odpovídající třídy v trénovacím vzoru. Vzhledem k tomu, že klasifikátor pracuje jako posuvné okno, je nutné tyto odpovědi filtrovat podle vzdálenosti. Podle velikosti okna je nastavena určitá hranice.

Pro výpočet skóre S_i byl navržen jednoduchý vzorec 4.7 zohledňující vzdálenost od středu objektu D.

$$S_i = \sum_{points} \frac{1}{1+D} \tag{4.7}$$

Hodnoty MS_i jsou počítány pro celý obraz stejným vzorcem, ale nebere se v úvahu výstup klasifikátoru. Vzdálenost byla vypočtena pomocí vzorce 4.8, kde s_x a s_y jsou souřadnice středu nejbližšího objektu. Pozice okna v obraze je dána vektorem (r, c).

$$D = \sqrt{(s_x - c)^2 + (s_y - r)^2} \tag{4.8}$$

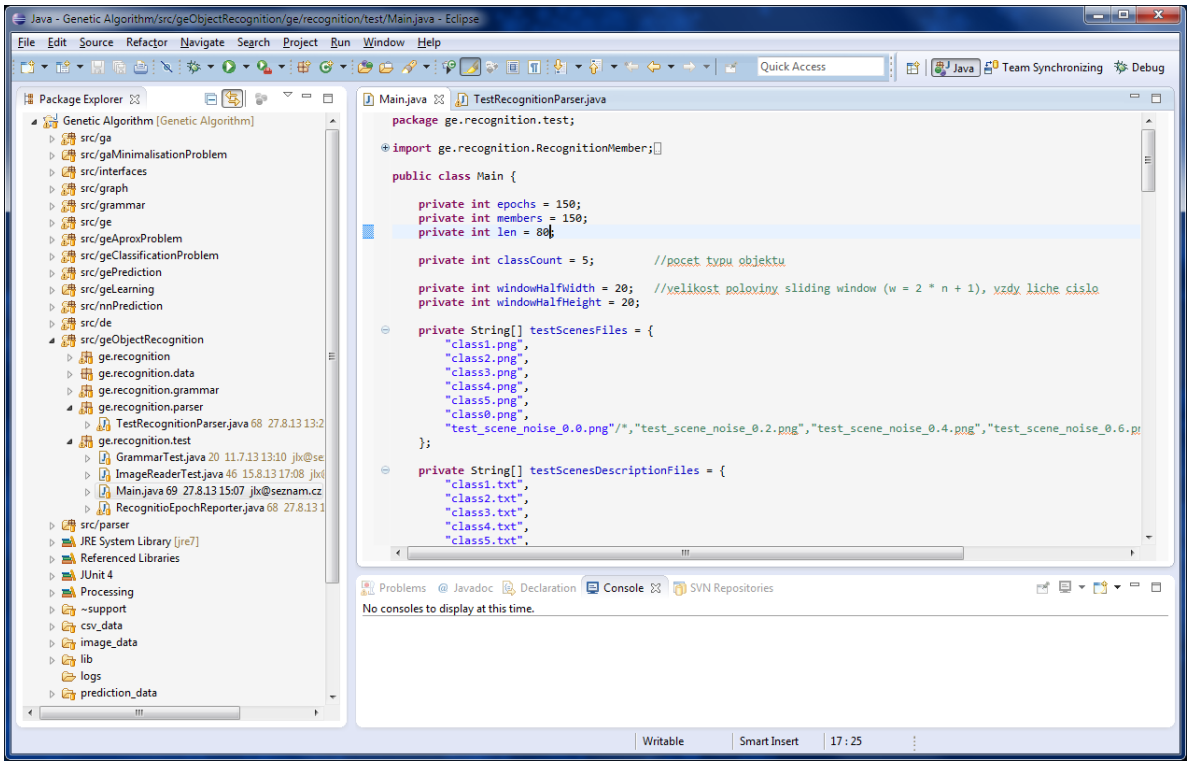
5. Popis a architektura testovacího prostředí

Testovací prostředí bylo vytvořeno v programovacím jazyce Java [26]. Je dostupné jako elektronická příloha této práce. Java je objektově orientovaný, staticky typovaný jazyk se syntaxí odvozenou od jazyka C. Tento programovací jazyk byl zvolen z důvodů jeho vestavěné podpory pro práci s vlákny a synchronizaci přístupu k datům z vláken.

Jako vývojové prostředí byl zvolen program Eclipse [11] (obrázek 5.1). Tento je též napsán v programovacím jazyce Java. Podporuje tvorbu projektů, spouštění, ladění. Další užitečná vlastnost je možnost instalace zásuvných modulů – byl použit systém pro správu verzí SVN [1].

Testovací prostředí má v aktuální verzi pouze minimální uživatelské rozhraní. Úlohy se definují a konfigurují přímo ve zdrojovém kódu. Byl implementován pouze systém zobrazování grafů a obrázků pro vykreslení zadání nebo průběhu některých úloh a průběhu hodnoty fitness funkce.

Implementace evolučních metod není triviální záležitost. V této kapitole jsou popsány podrobněji implementované algoritmy. Některá inovativní řešení, která zrychlují běh celého procesu, jsou popsány podrobně.

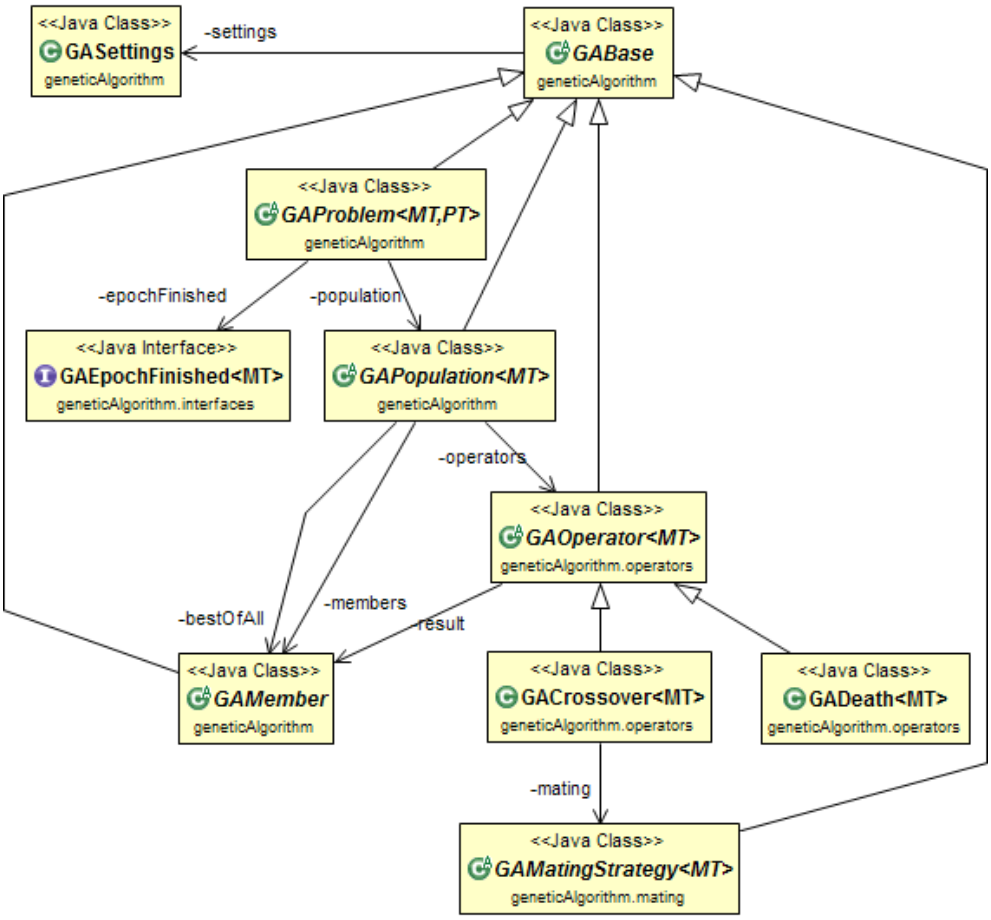


Obrázek 5.1: Vývojové prostředí Eclipse

5.1. Obecná typová struktura

V této části bude podrobněji popsáno vytvořené testovací prostředí. Schema základní objektové hierarchie je na obrázku 5.2 ve formátu UML. Společný předek všech tříd (kromě nastavení) je abstraktní třída *GABase*, která zajišťuje přístup k nastavením.

Ve zdrojovém kódu jsou použita tzv. generika. Při překladu je pak pomocí textového makra nahrazen generický typ MT (Member Type = datový typ jedince) a PT (Population Type = datový typ populace) konkrétním dosazeným datovým typem.



Obrázek 5.2: UML diagram tříd pro obecné jádro evolučních metod. Obrázek byl vygenerován pomocí nástroje ObjectAid UML Explorer pro Eclipse [25]

5.1.1. Definice problému

Každá řešená úloha je definována jako potomek abstraktní třídy *GAProblem*. Na instanci tohoto objektu je následovně spuštěno řešení problému genetickým algoritmem. V tomto objektu je též po každé iteraci spuštěno hlášení o stavu populace, což je objekt implementující rozhraní *GEEpochReporter*.

5.1.2. Nastavení procesu

Každý objekt má přístup k potomku abstraktní třídy *GASettings*. V této třídě jsou uložena obecná nastavení genetického algoritmu (počet iterací, velikost populace, ...). *GASettings* může dále uložit nastavení specifická pro řešený problém. Například abstraktní třída *GESettings*, která slouží k uložení natavení pro gramatickou evoluci přidává možnost uložit délku chromozomu.

Třída nastavení umožňuje nastavit i proměnlivou pravděpodobnost mutace (lineárně roste s počtem iterací, pokud se nezlepší fitness hodnota nejlepšího jedince) a zapnutí elitismu až po určitém počtu iterací. Tyto nastavení spolu souvisí, protože pokud nastavíme velkou pravděpodobnost mutace u konce procesu, téměř určitě dojde ke znehodnocení nejlepšího jedince. Proto je dobré jej vždy kopírovat rovnou do nové populace.

5.1.3. Populace

Populace je vždy implementována jako potomek abstraktní třídy *GAPopulation*. Je to jednoduše seznam jedinců s několika málo metodami. Důležitá metoda implementovaná na populaci je ohodnocení všech jedinců fitness funkcí. Zde je možné v nastavení zvolit, zda proces ohodnocování bude běžet paralelně na všech výpočetních jádrech, nebo sekvenčně na jednom jádru procesoru.

V tabulce 5.1 je ukázka rozdílu v době trvání výpočtu, pokud je při výpočtu použito jedno, nebo více výpočetních jader procesoru. Test byl proveden na čtyř-jádrovém procesoru Intel Core i5-3470 (3,2 GHz). K testování posloužila jedna z implementovaných úloh gramatické evoluce, konkrétně popsaná v 4.5. Počet jedinců byl 100, počet populací také. Prezentován je tedy čas nutný pro ohodnoceni 100×100 jedinců a zpracování jejich křížení a mutací.

Počet jader	Doba výpočtu [ms]
1	152404
2	86017
3	58952
4	55672

Tabulka 5.1: Porovnání průměrné doby trvání výpočtu při použití více výpočetních jader procesoru – Intel Core i5-3470

Z výsledků v tabulce vyplývá, že zrychlení při použití více jader je značné, ale vzhledem k tomu, že paralelně je implementována pouze část programu ohodnocující jedince, čas ušetřený s nasazením většího počtu jader neklesá lineárně. Je to dáno tím, že u gramatické evoluce jsou časově náročné i části programu provádějící křížení a mutaci. Ty by se daly také provádět paralelně, ale vytvořené prostředí to nepodporuje.

Počet jader	Doba výpočtu [ms]
1	353822
2	199277

Tabulka 5.2: Porovnání průměrné doby trvání výpočtu při použití více výpočetních jader procesoru – Intel Core2Duo T6500

V tabulce 5.2 je ještě provedeno srovnání na dvou-jádrovém procesoru Intel Core2Duo T6500 (2,1 GHz). Tento test byl proveden z důvodu zjištění, nakolik nevyužitá jádra odfiltrují ostatní aktivitu uživatele a jiných programů v operačním systému. Je možné spočítat, že poměr zrychlení mezi použitím jednoho jádra nebo dvou je zhruba 1,77 v obou případech. Běžné vytížení procesoru ostatní prací v systému se tedy projeví na obou procesorech stejně. Zajímavý je více než dvojnásobný výkon novějšího procesoru.

5.1.4. Jedinec

Implementace vychází ze třídy abstraktní *GAMember*. Jedinec je implementován v závislosti na řešeném úkolu. Nejdůležitější je metoda umožňující výpočet fitness hodnoty. Instance jedince následovně dokáže říci, zda došlo ke změně chromozomu a je nutné přepočítat fitness hodnotu.

Jedinec umí uchovat svoje ID pro ladící účely a epochu, kdy byl zařazen do populace. Z této informace lze snadno dopočítat věk jedince.

5.1.5. Operátory

Operátory jsou funkce, které jsou spouštěny nad celou populací v každé iteraci algoritmu a jejich použitím se vytváří noví jedinci a usměrňuje se chod algoritmu. Nemusí být použity všechny. Abstraktní třída, ze které vychází všechny operátory je pojmenována *GAOperator*.

Operátory jsou zaregistrovány na začátku procesu do populace a ta je v každé iteraci sekvenčně spouští. Před každým spuštěním operátoru je nutné zajistit vyhodnocení fitness u každého jedince. Použitím více operátorů tedy může znamenat prodloužení celého procesu.

5.1.5.1. Křížení a mutace

Klasickým operátorem je křížení spojené s mutací jedinců. Někdy jsou tyto dvě operace implementovány nezávisle – to znamená, že mutovat může i jedinec, který nevznikl křížením. Ve vytvořeném vývojovém prostředí byly tyto operace zřetězeny tak, že mutací mohl být ovlivněn pouze nově vzniklý jedinec. Operaci křížení s mutací implementuje operátor *GACrossover*.

Jelikož reprezentace jedinců bývá různá, je konkrétní křížení a mutace na úrovni chromozomu implementováno pomocí instance třídy, která je potomkem abstraktní třídy *GAMatingStrategy*.

5.1.5.2. Smrt

Dalším jednoduchým příkladem univerzálního operátoru je smrt [35]. Implementuje je ji třída *GADeath*. Pokud má jedinec věk větší než určitou zadanou hranici, může být z populace pomocí tohoto operátor vyřazen. Usmrcený jedinec může být nahrazen například novým náhodným jedincem nebo potomkem jiných jedinců.

5.1.5.3. Diversita

Diversita může být implementována na populacích, které umožňují porovnat dva jedince z hlediska podobnosti jejich chování nebo fenotypu.

Pokud jsou v populaci dva nebo více velmi podobní jedinci, mohou být takoví jedinci odstraněni a nahrazeni jinými. Tento operátor působí proti jevu, kdy genetický algoritmus uvízne v lokálním minimu a v populaci začne převládat jeden fenotyp.

U gramatické evoluce se nabízí měření podobnosti dvou jedinců pomocí vzdálenosti řetězců – tedy na úrovni fenotypu. Jedna z použitelných metrik je *Levenshteinova* vzdálenost.

5.1.5.4. Další operátory

Operátorů lze implementovat hodně a lze je různě modifikovat a vybírat jejich kombinace. Aby genetický algoritmus pracoval správně, je nutné umožnit minimálně vznik nových řešení kombinací existujících (obvykle pomocí křížení) a umožnit náhodnou změnu (obvykle pomocí mutace) – je nutné umožnit algoritmu prozkoumávat prostor řešení. Další operátory mohou, ale nemusejí zrychlit proces optimalizace.

5.2. Typová struktura gramatické evoluce

Gramatická evoluce rozšiřuje standardní jádro genetických algoritmů a standardizuje přístup k chromozomu a tedy i ke křížení a mutaci. Bylo tedy možné implementovat třídy *GEProblem*, *GEPopulation*, *GEMember*, *GECrossover*, *GEMating* – obrázek 5.3. Tyto třídy již přímo pracují s celočíselným chromozomem a gramatikou pro jeho překlad. Při překladu je získán důležitý značkovací vektor, který umožní nahradit části chromozomu nebo pozměnit jednotlivé geny a tím provést mutaci a křížení.

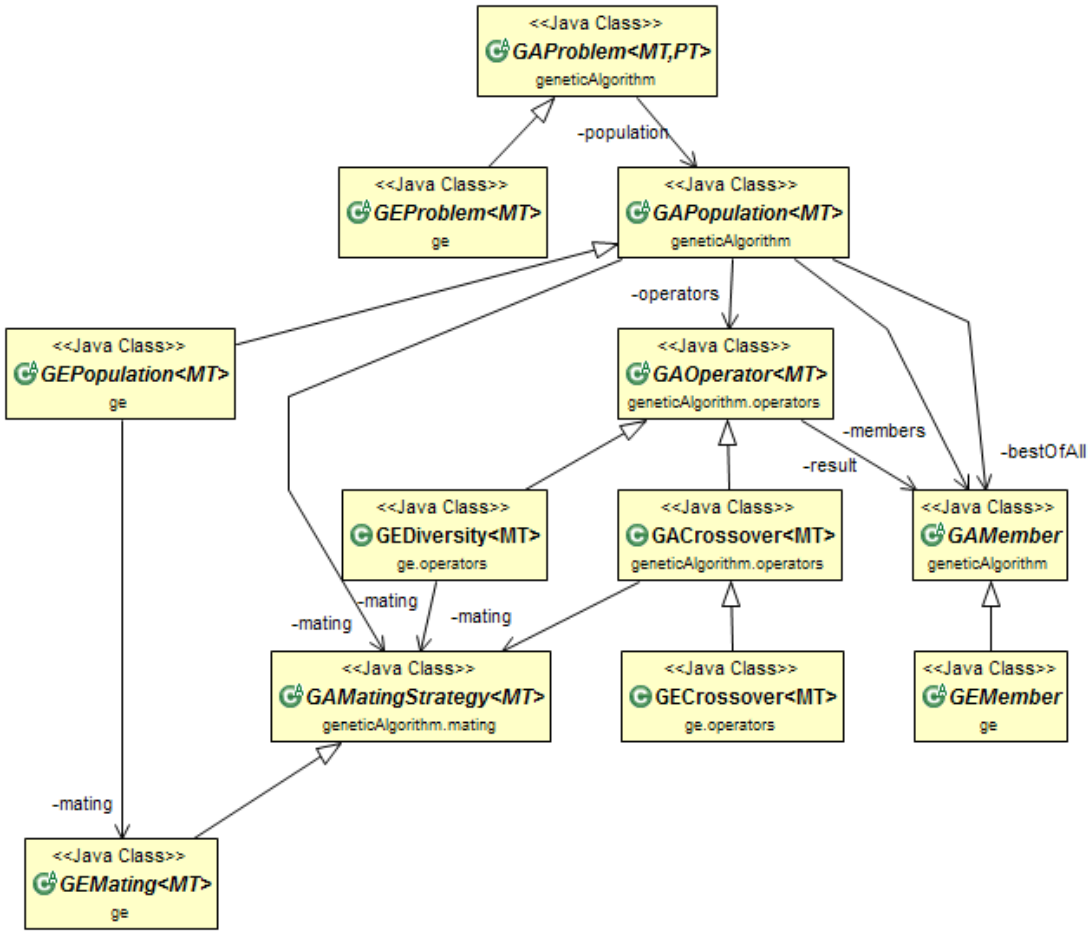
5.3. Implementace podstatných funkcí pro gramatickou evoluci

Pro rychlejší běh byly implementovány funkce, které umožňují získat tzv. značkovací vektor a funkce pro přímé získání spustitelného stromu. V následujících kapitolách budou tyto postupy popsány.

5.3.1. Získání značkovacího vektoru

Na obrázku 5.4 je zachycen výsledek překladu chromozomu pomocí gramatiky. Zároveň je v dolní části zobrazen graf, kde jednotlivé hrany znázorňují použití číslice z chromozomu. V malých kosočtvercích je potom udána pozice, kterou překlad dané větve začal a skončil. Tyto pozice určují hranice značek. Terminální symboly jsou vyznačeny červeně. Terminály jsou vždy vybírány posledním prvkem části chromozomu, kterou značka vymezuje.

Záměnou červeně vyznačené číslice dosáhneme pouze výběru jiného terminálu – nestrukturální mutace. Strukturální mutaci je možné provést tak, že vyměníme celý obsah



Obrázek 5.3: UML diagram tříd pro gramatickou evoluci. Obrázek byl vygenerován pomocí nástroje ObjectAid UML Explorer pro Eclipse [25].

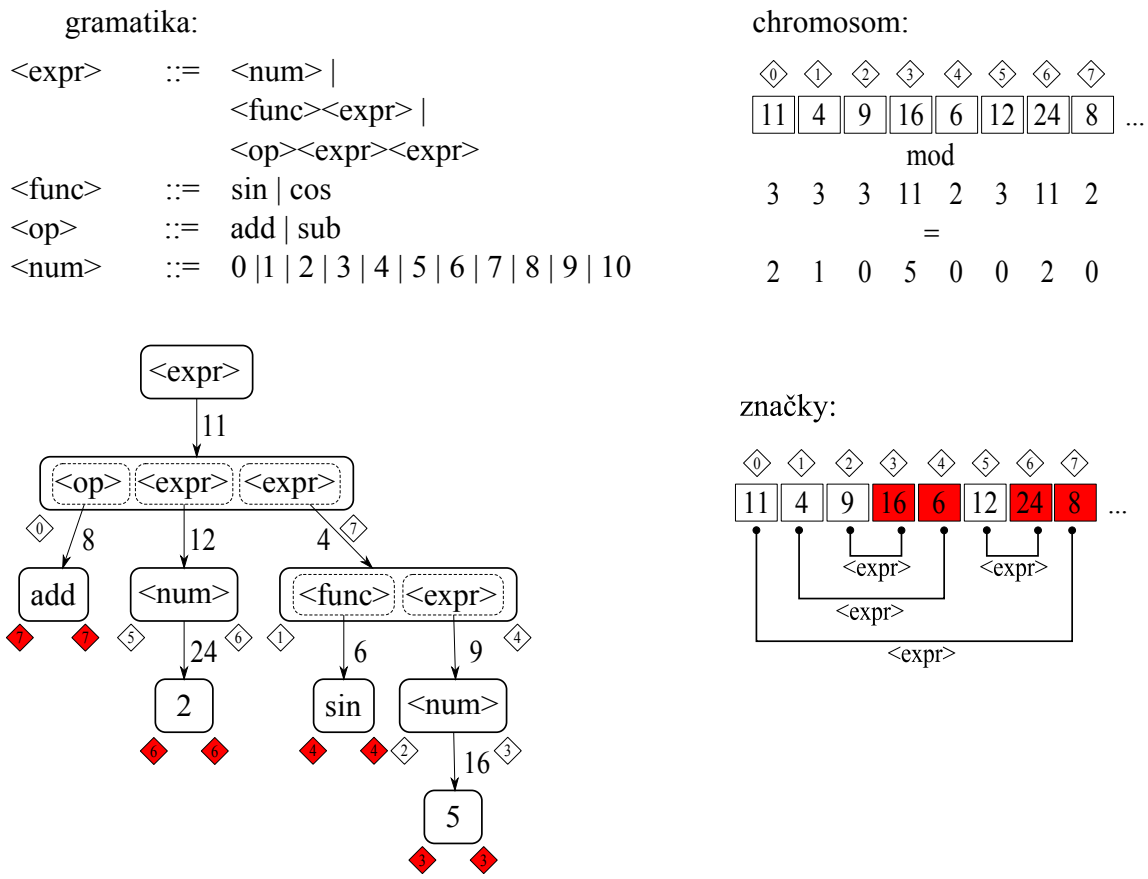
značky (nové pole číslic nemusí být nutně stejně dlouhé). Je ovšem nutné zajistit, aby pole, které nahrazuje obsah značky vycházelo ze stejného neterminálního symbolu. Ke značkám je tedy nutné uložit i informaci, jaký neterminální symbol překládají.

U gramatiky na obrázku 5.4 je pouze jediné pravidlo, které generuje neterminální symboly (<*expr*>). Naproti tomu u gramatiky na obrázku 5.5 jsou pravidla, která generují další neterminální symboly dvě (<*expr*> a <*digits*>). Není proto možné vzít z jiného jedince obsah značky, která přepisuje pravidlo <*expr*> a nahradit s ním obsah značky, která přepisuje značku generující <*expr*>. Důsledkem výměny obsahu značek, které popisují obsah jiného neterminálního symbolu by byl úplně jiný jedinec.

5.3.2. Algoritmus křížení

Odpovídající algoritmus pro křížení dvou jedinců pomocí značek je následující:

1. Zkontroluj, zda značky v obou jedincích mají průnik množin jejich neterminálů.
2. Vyber náhodnou značku z prvního jedince a zjisti její neterminál.
3. Vyber náhodnou značku z druhého jedince, neterminál musí odpovídat neterminálu první značky.



Obrázek 5.4: Získání značkovacího vektoru při překladu chromozomu

4. Zkontroluj, zda je možné zaměnit obsah značek (výsledný chromozom nesmí překročit pevnou délku).

5. Pokud ano – vyměň obsah značek. Pokud ne – vrať se na krok 1.

6. Pokud je některý z chromozomů kratší, doplň jej náhodnými čísly.

5.3.3. Algoritmus mutace

Odpovídající algoritmus pro mutaci jedince pomocí značek je následující:

1. Vyber typ mutace.

2. Strukturální mutace:

   (a) Najdi náhodnou značku.

   (b) Nastav překladové gramatice počáteční symbol stejný jako má značka.

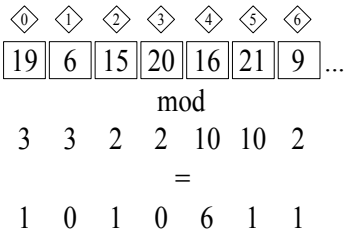
   (c) Pomocí generátoru náhodných čísel vygeneruj sekvenci generující přepis neterminálu.

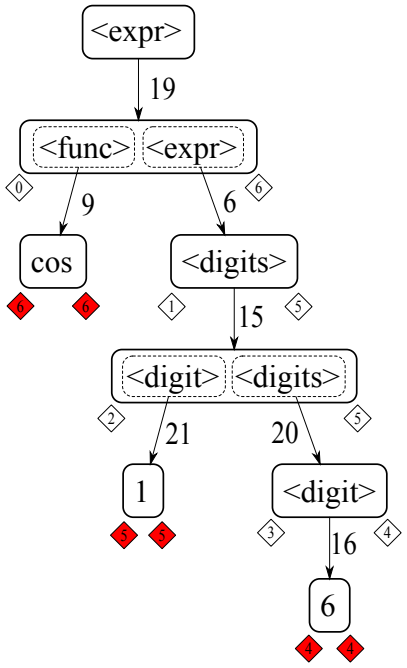
   (d) Zkontroluj, zda je možné zaměnit obsah značky a vygenerovanou sekvenci (výsledný chromozom nesmí překročit pevnou délku).

gramatika:

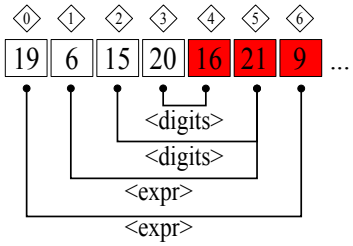
```
<expr>    ::=  <digits> |
               <func><expr> |
               <op><expr><expr>
<digits>  ::=  <digit>|<digit><digits>
<func>    ::=  sin | cos
<op>      ::=  add | sub
<digit>   ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

chromosom:





značky:



Obrázek 5.5: Získání značkovacího vektoru při překladu chromozomu – složitější gramatika

(e) Pokud ano – vyměň obsah značek. Pokud ne – vrať se na krok 2c.

(f) Pokud je chromozom kratší, doplň jej náhodnými čísly.

3. Nestrukturální mutace:

(a) Najdi terminál.

(b) Změň náhodně jeho hodnotu.

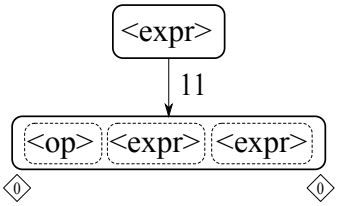
5.3.4. Získání značkovacího vektoru přímo při překladu chromozomu

Důležitý je i postup získávání značek. Na následujících obrázcích bude popsán postup získání značkovacího vektoru. Gramatika a výsledek odpovídá obrázku 5.4.

Značky jsou získávány přímo při překladu použitím dané stromové struktury. Jako kořen stromu je vždy určen počáteční symbol gramatiky. K jeho překladu je použita první číslice chromozomu (11). Jelikož v gramatice máme 3 možnosti překladu neterminálu <*expr*>, použijeme operaci *modulo* (zbytek po dělení) a získáme 11 *mod* 3 =

2. Použijeme tedy přepis s indexem 2 (počítáme od nuly). Výsledek přepisu je tedy <*op*><*expr*><*expr*>. Tento krok je na obrázku 5.6.

Do stromu jsme tedy přidali nový uzel, který je potomkem kořenového uzlu a obsahuje aktuální pracovní řetězec. K tomuto uzlu si můžeme uložit doposud použité indexy chromozomu, ze kterých dostaneme značky. Na obrázcích jsou hodnoty začátku a konce značky reprezentovány malými kosočtverci s čísly. Zatím je to hodnota 0 pro konec i začátek značky, jelikož byla použita první hodnota z chromozomu.

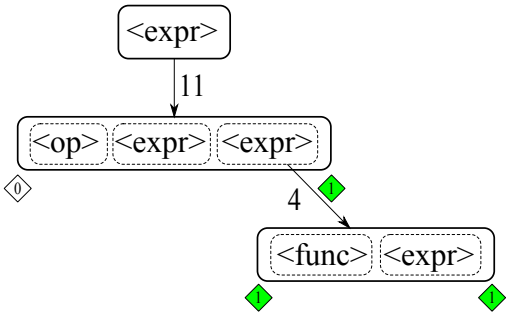


Obrázek 5.6: Získání značkovacího vektoru při překladu chromozomu – krok 1

V dalším kroku, je zobrazen na obrázku 5.7, přepisujeme poslední neterminál v pracovním řetězci <*op*><*expr*><*expr*>. Je to opět neterminál <*expr*>. Číslice v chromozomu je tentokrát 4 a počet voleb opět 3. Zbytek po dělení je 1 a použijeme tedy druhý přepis.

Do stromu přidáváme další uzel, který získá počáteční index značky přičtením hodnoty 1 ke koncovému indexu značky rodičovského uzlu (0+1 = 1). Koncový index je zatím shodný s počátečním, ale rodičovský uzel je nutné aktualizovat tak, že jeho koncovému indexu nastavíme hodnotu konce značky aktuálně přidávaného uzlu. Pokud by rodičovský uzel měl nadřazen další uzly, je nutné rekurzivně aktualizovat i tyto.

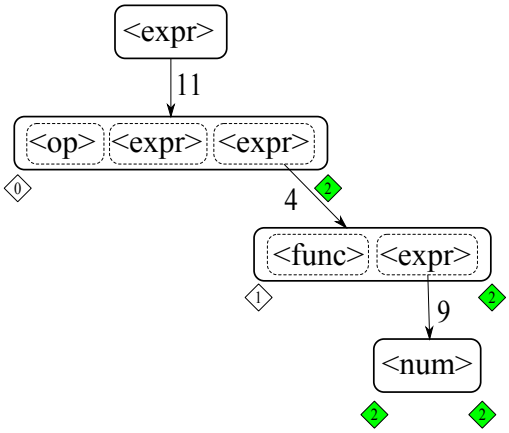
Upravené a nové hodnoty značek jsou značeny zeleně.



Obrázek 5.7: Získání značkovacího vektoru při překladu chromozomu – krok 2

V dalším kroku, je zobrazen na obrázku 5.8, přepisujeme poslední neterminál v pracovním řetězci <*func*><*expr*>. Je to opět neterminál <*expr*>. Číslice v chromozomu je 9 a počet voleb je opět 3. Zbytek po dělení je 0 a použijeme tedy první přepis.

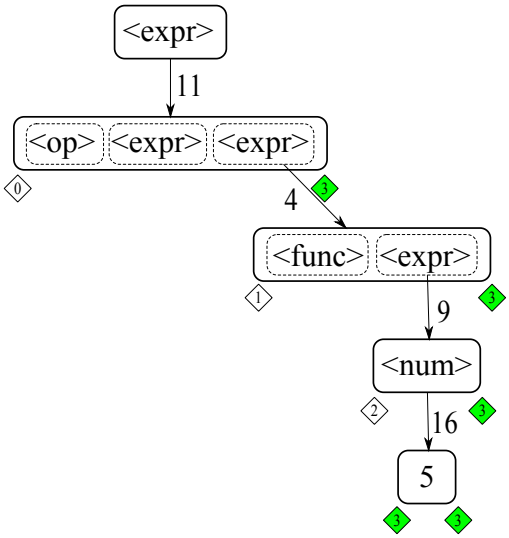
Do stromu přidáváme další uzel, který získá počáteční index značky přičtením hodnoty 1 ke koncovému indexu značky rodičovského uzlu (1 + 1 = 2). Koncový index je zatím shodný s počátečním, ale rodičovský uzel a jeho předchůdce je nutné aktualizovat tak, že jeho koncovému indexu nastavíme hodnotu konce značky aktuálně přidávaného uzlu. Tato hodnota se rekurzivně přenese i nadřazeným uzlům.



Obrázek 5.8: Získání značkovacího vektoru při překladu chromozomu – krok 3

V dalším kroku, je zobrazen na obrázku 5.9, přepisujeme poslední neterminál v pracovním řetězci <*num*>. Číslice v chromozomu je 16 a počet voleb je 11. Zbytek po dělení je 5 a použijeme tedy pátý přepis a získáme terminální symbol *5*.

Do stromu přidáváme další uzel, který získá počáteční index značky přičtením hodnoty 1 ke koncovému indexu značky rodičovského uzlu ($2 + 1 = 3$). Koncový index je zatím shodný s počátečním, ale rodičovský uzel a jeho předchůdce je nutné aktualizovat tak, že jeho koncovému indexu nastavíme hodnotu konce značky aktuálně přidávaného uzlu. Tato hodnota se rekurzivně přenese i nadřazeným uzlům.

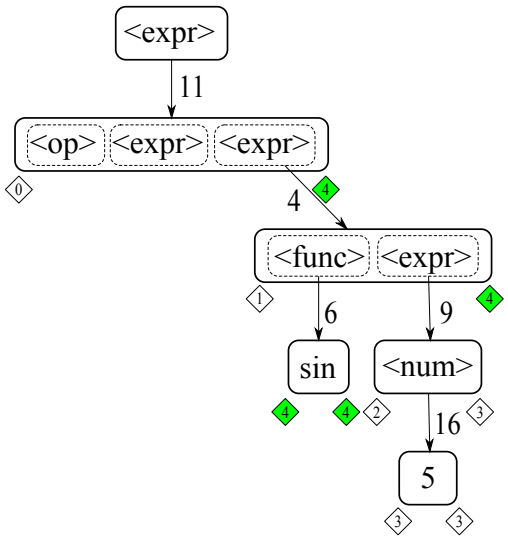


Obrázek 5.9: Získání značkovacího vektoru při překladu chromozomu – krok 4

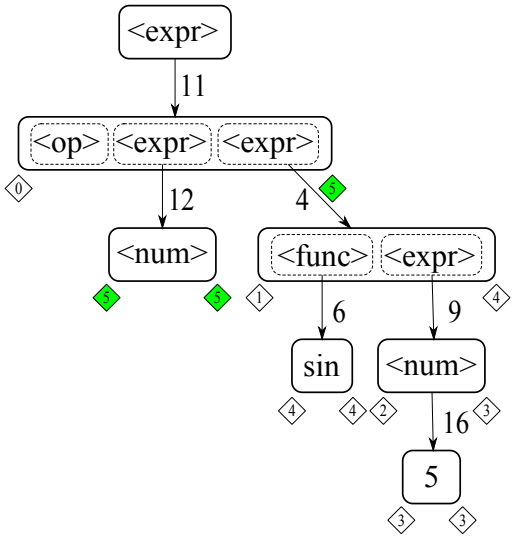
V kroku na obrázku 5.10 je přepsán neterminál <*func*> na terminál *sin*. Opět jsou aktualizovány koncové indexy značek všech nadřazených uzlů.

V kroku na obrázku 5.11 se opět přepisuje neterminál <*expr*>. Výběrem pravidla pomocí chromozomu se přepíše na neterminál <*num*> a ten je potom přepsán na terminál *2*. To je na obrázku 5.12.

Poslední krok překladu, na obrázku 5.13 je přepsán neterminál <*op*> na terminální symbol *add*.



Obrázek 5.10: Získání značkovacího vektoru při překladu chromozomu – krok 5

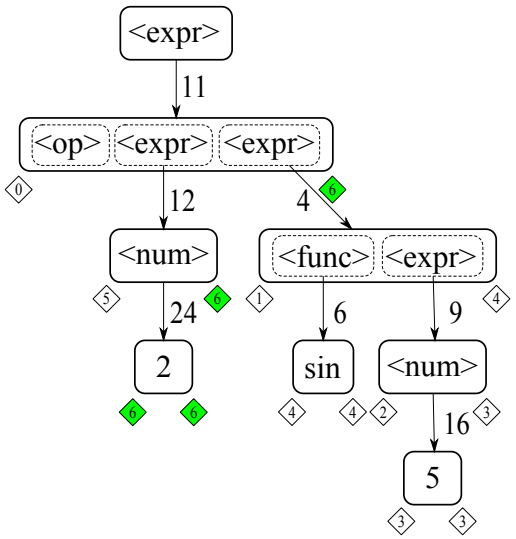


Obrázek 5.11: Získání značkovacího vektoru při překladu chromozomu – krok 6

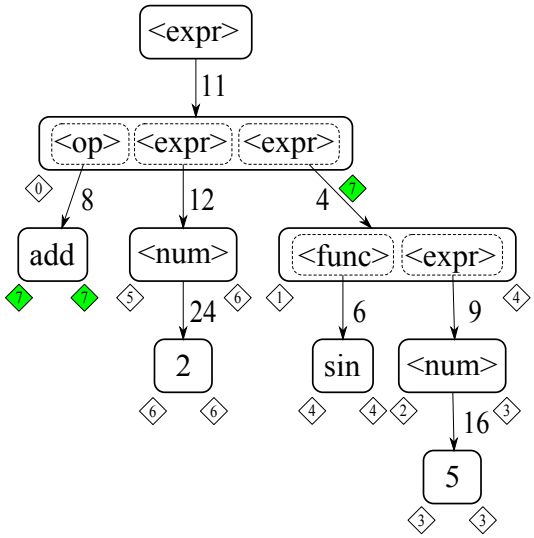
Hotové značky pak jednoduše posbíráme z jednotlivých uzlů grafu a zapíšeme je do seznamu značek k danému chromozomu. S použitím takovéhoto algoritmu automaticky paralelně s tvorbou výsledného fenotypu jedince získáváme vektor značek.

5.3.5. Přímý překlad chromozomu na spustitelný strom

Dalším vylepšením, které urychluje proces gramatické evoluce je přímý překlad chromozomu. Tento krok odstraňuje nutnost použití parseru, který by převáděl textový řetězec na spustitelnou strukturu. Jakmile je při překladu vytvořen terminál, je vložen do zásobníku. Díky zpětnému překladu je jako poslední terminál vložen nejvyšší uzel spustitelného stromu. Hned na ním je jeho první argument a za ním jeho argumenty, pokud má. Každý terminál zná počet svých argumentů, takže pomocí operace *pop* (výběr ze zásobníku) snadno získáme argumenty každého uzlu a přidáme je do stromu. Tento postup je spouš-



Obrázek 5.12: Získání značkovacího vektoru při překladu chromozomu – krok 7

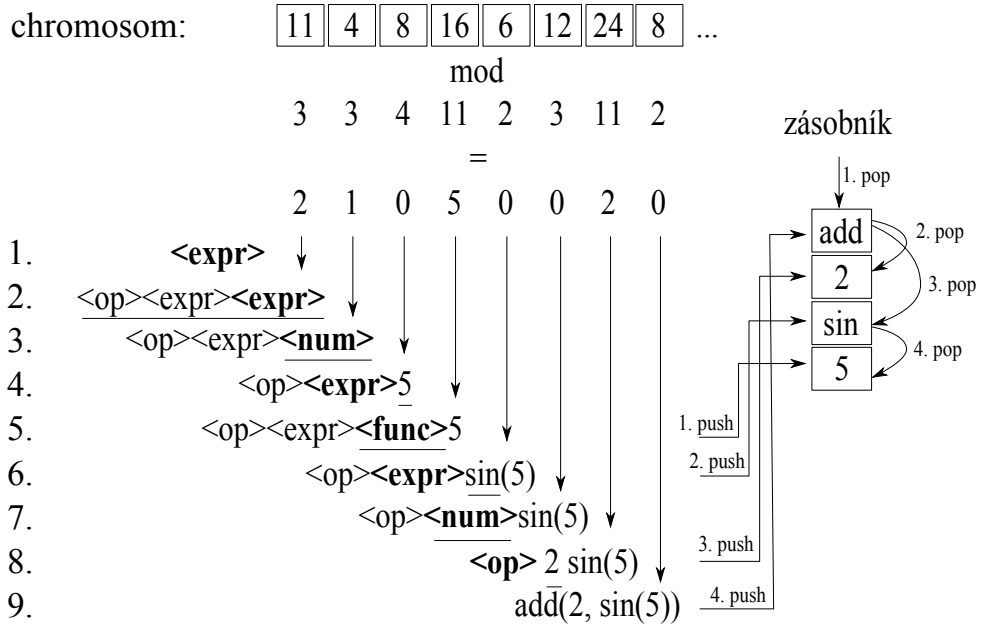


Obrázek 5.13: Získání značkovacího vektoru při překladu chromozomu – krok 8

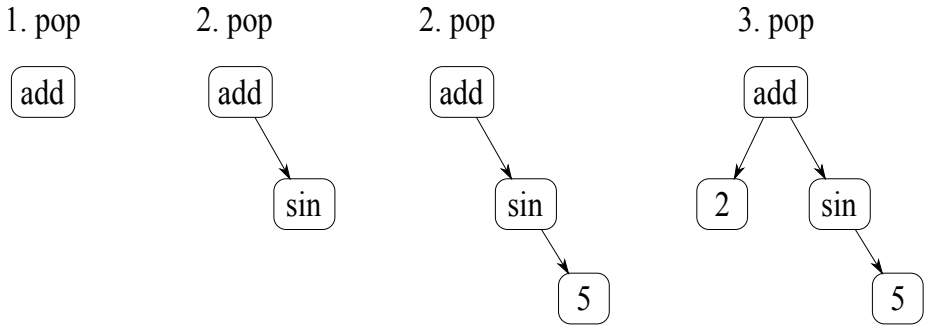
těn rekurzivně pro každý prvek zásobníku. Pokud je gramatika správně nadefinována, skončí překlad tak, že zásobník je prázdný. Tento postup je demonstrován na obrázku 5.14 a 5.15.

5.3.6. Parser

Jednoduchý parser byl samozřejmě implementován také. K jeho funkci je potřeba gramatika, podle které rozeznává terminální symboly a váže je na funkční uzly spustitelného grafu. Ten je vytvořen opět pomocí zásobníku stejně jako v 5.3.5. Parser umí zpracovat pouze prefixovou notaci ve formátu *funkce(argumenty, . . .)*, kde argument může být další funkce nebo číslice.



Obrázek 5.14: Získání spustitelného stromu – překlad chromozomu



Obrázek 5.15: Získání spustitelného stromu – tvorba

5.4. Implementované úlohy

Zde budou stručně popsány implementované úlohy, na kterých byla testována funkce genetických algoritmů a gramatická evoluce a jsou spustitelné ve vytvořeném prostředí.

5.4.1. Hledání minima funkce

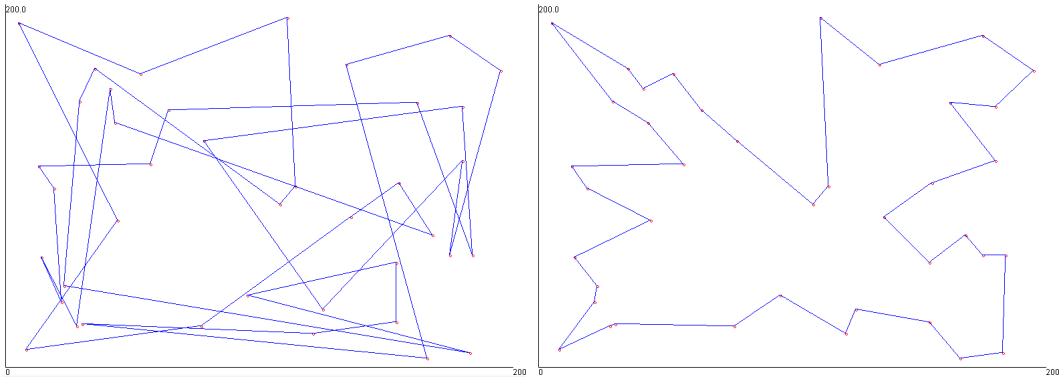
V kapitole 3.4 byla popsána úloha hledání minima matematické funkce s jednou proměnnou. Byla implementována jako první ve vytvořeném systému pro možnosti testování.

5.4.2. Problém obchodního cestujícího

Další úlohou, kterou jsou často demonstrovány schopnosti genetických algoritmů je problém obchodního cestujícího. Tento klasický problém řeší optimalizaci okružní cesty mezi městy, která má obchodní cestující navštívit. Hledá se takové pořadí měst, aby cesta obchodního cestujícího byla co nejkratší. Genetický algoritmus je pro řešení této úlohy překonám [8].

Vstupem do úlohy jsou města se zadanými pozicemi nebo vzdálenostmi mezi nimi. Chromozom v tomto případě kóduje index města, které má být navštíveno v dalším kroku. Pomocí chromozomu je pak sestavena plánovaná cesta a jedinec může být ohodnocen. Při křížení, které je realizováno záměnou sekvencí měst, je v tomto případě nutné kontrolovat, zda upravený chromozom neobsahuje dvakrát stejné město. Při mutaci jsou náhodně zaměněny pozice měst v chromozomu.

Na obrázku 5.16 je ukázka řešení problému obchodního cestujícího pro 40 měst (červené tečky) na ploše 200×200 jednotek. Trasa (modře) byla optimalizována na hodnotu 1120 jednotek.



Obrázek 5.16: Úloha nalezení minimální trasy pro obchodního cestujícího. Počáteční stav (vlevo) a koncový stav (vpravo). Obrázky byly pořízeny přímo v testovacím prostředí.

5.4.3. Úlohy používající gramatickou evoluci

Jelikož těžiště této práce leží hlavně ve využití gramatické evoluce, jsou i další implementované úlohy postavené na tomto principu. V následujících odstavcích budou popsány doplňující úlohy, které se přímo netýkají rozpoznávání objektů.

5.4.3.1. Regrese dat matematickou funkcí

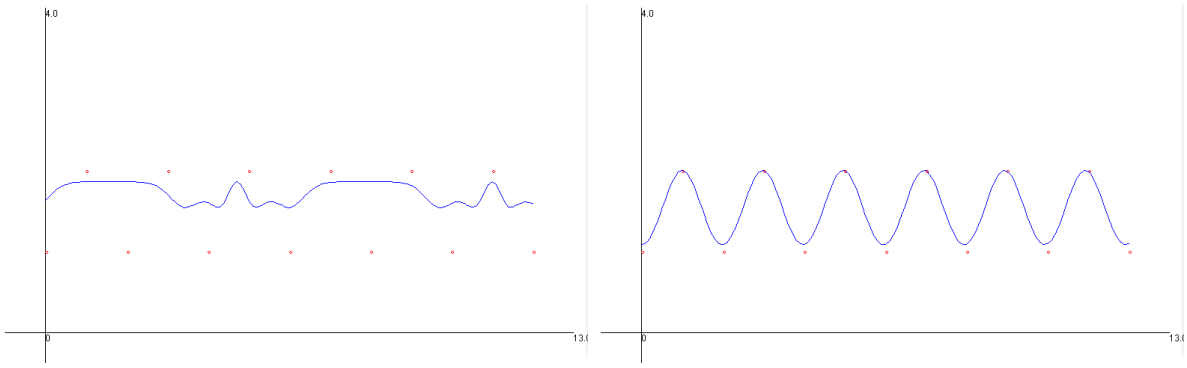
Základní demonstrační úloha, která se řeší pomocí gramatické evoluce, je hledání vhodné matematické funkce pro regresi zadaných dat. Gramatická evoluce je k řešení takovéto úlohy více než vhodná, jelikož není omezena typem matematické funkce (polynom, periodická funkce apod.) a pro zadaná data dokáže vybrat nejvhodnější kombinaci terminálů.

Vstupem do úlohy jsou body a samozřejmě gramatika, která dokáže generovat matematický zápis funkce. Jedinec je ohodnocen pomocí chyby, kterou jeho matematická funkce produkuje (například hodnota SSE – *Sum of Squared Errors*).

Na obrázku 5.16 je zobrazena ukázka aproximace datových bodů (červeně) pomocí matematické funkce (modře).

5.4.3.2. Predikce časových řad

Tato úloha je podobná regresi dat matematickou funkcí, která je zadaná body. Byla vytvořena v rámci práce na [22]. Vstupem je sekvence čísel, které reprezentují datovou řadu. Pomocí gramatické evoluce byla vytvořena matematická funkce, která měla přístup k



Obrázek 5.17: Úloha aproximace zadaných bodů matematickou funkcí. Počáteční stav (vlevo) a koncový stav (vpravo). Obrázky byly pořízeny přímo v testovacím prostředí.

několika hodnotám časové posloupnosti a jejím úkolem bylo odhadnout hodnotu následující. Hodnotila se opět odchylka od učící budoucí hodnoty a predikované hodnoty. Tento prostup byl také srovnán s neuronovou sítí.

6. Testy metod a výsledky

V této kapitole budou prezentovány výsledky rozpoznávání objektů pomocí klasifikátorů vytvořených evoluční metodou. Metody byly popsané v 4, zde se nachází pouze výsledky a nastavení použité pro tvorbu klasifikátorů.

6.1. Testovací scéna a objekty

Pro testování metod byly zvoleny objekty, které jsou vyobrazeny na obrázku 6.1. Tyto objekty byly již dříve použity v [42], kde byla ke klasifikaci použita neuronová síť.

Testovací scéna byla vytvořena podpůrným skriptem, který umisťoval a rotoval objekty náhodně do obrazové matice zvolené velikosti. Velikost testovací scény byla zvolena 320×320 pixelů.

Jelikož jde o malou sadu objektů zasazených do testovací scény, očekávala se od genetického algoritmu vysoká úspěšnost při klasifikaci. Šlo hlavně o ověření schopnosti navržených metod nalézt klasifikátor nebo detektor.

a) Třída 1 b) Třída 2 c) Třída 3 d) Třída 4 e) Třída 5

Obrázek 6.1: Testovací objekty ve stupních šedi

6.2. Klasifikátor s jednohodnotovým výstupem

Jako první byly testovány metody navržené a popsané v 4.4.

6.2.1. Jednofázový přístup

Jednofázový přistup se ukázal jako nevhodný, nicméně výsledek odvozování programu gramatickou evolucí by byl velmi podobný klasifikačnímu programu z následující sekce. Rozdíl je pouze v použití detektoru, který vymezí místa, kde klasifikátor spouštět.

6.2.2. Dvoufázový přístup

Dvoufázový přístup vykazoval lepší výsledky i rychlejší průběh (díky spouštění klasifikátoru pouze nad body označenými detektorem). Důležité bylo natrénovat detektor tak, aby opravdu rozpoznal všechny objekty, pokud se to nepodařilo, nemohl už i kvalitní klasifikátor rozpoznat neoznačený objekt. Klasifikátor i detektor byly tvořeny nezávisle. Tento postup byl publikován v [21].

V tabulce 6.1 jsou uvedeny parametry nastavené evolučnímu procesu.

Parametr	Hodnota	Vysvětlení
Počet epoch	100	Počet iterací algoritmu.
Velikost populace	100	Počet jedinců v populaci.
Elitismus	Enabled	Kopírování nejlepších jedinců do další populace.
Pravděpodobnost křížení	90 %	Pravděpodobnost křížení, pokud křížení neproběhne, jsou jedinci nakopírováni do další populace. Popsáno v 3.5.2.1.
Pravděpodobnost mutace	10 %	Pravděpodobnost náhodných změn v chromozomech – dále se u gramatické evoluce dělí na mutaci strukturální a nestrukturální. Popsáno v 3.5.2.2.
Selekce	Turnaj/3	Turnajová selekce – velikost turnaje jsou tři jedinci. Popsáno v 3.3.3.2.
Délka chromozomu	80	Pro výběr pravidel z gramatiky je možné použít maximálně daný počet genů.

Tabulka 6.1: Parametry nastavené evolučnímu procesu

V tabulce 6.2 je uvedena gramatika, která byla použita k překladu chromozomů na spustitelné programy. Gramatika je postavena tak, aby produkovala programy, které dokážou získat vhodnou sekvenci maticových operací nad odpovídajícím regionem tak, aby výsledná statistická analýza byla schopná rozřadit výstupy nad objekty do nepřekrývajících se intervalů – tím je zaručena bezchybná detekce objektů.

Neterminály		Množina přepisů
`<expr>`	`::=`	`<matrix_stats><matrix><region>`
`<matrix>`	`::=`	`<img_matrix_source> | <math_fun_matrix><matrix> | <math_op_matrix><matrix><matrix>`
`<region>`	`::=`	`<region_generator> | <region_op><region><region>`
`<img_matrix_source>`	`::=`	`s_img() | s_sobel() | s_blur()`
`<math_fun_matrix>`	`::=`	`mat_pow() | mat_p_sqrt() | mat_abs()`
`<math_op_matrix>`	`::=`	`mat_add() | mat_sub() | mat_mul() | mat_p_div()`
`<matrix_stats>`	`::=`	`ms_avg() | ms_stdev() | ms_mode() | ms_min() | ms_max()`
`<region_generator>`	`::=`	`r_center_point() | r_round_quarter() | r_round_half() | r_round_full() | r_square_quarter() | r_square_half()`

Pokračování na další stránce.

Tabulka 6.2 – *Pokračování z předchozí stránky.*

Neterminály	Množina přepisů
`<region_op>`	`::= r_and() | r_or() | r_xor()`

Tabulka 6.2: Gramatika použitá k hledání klasifikačního programu

V tabulce 6.3 jsou vysvětleny funkce jednotlivých terminálů gramatiky.

Terminál	Vysvětlení	Vstup Výstup
s_img()	Vstupní uzel – vrací černobílý obraz.	in: nic out: matice
s_sobel()	Vstupní uzel – vrací obraz získaný Sobelovým operátorem pro detekci hran.	in: nic out: matice
s_blur()	Vstupní uzel – vrací obraz získaný konvolucí obrazu a konvolučního jádra představujícího dvourozměrné Gaussovo rozložení.	in: nic out: matice
mat_pow(*arg*), mat_p_sqrt(*arg*), mat_abs(*arg*)	Základní matematické funkce na jednotlivých buňkách matice. Odmocnina ze záporného čísla je ošetřena a výsledek je nula.	in: matice out: matice
mat_add(*arg1*,*arg2*), mat_sub(*arg1*,*arg2*), mat_mul(*arg1*,*arg2*), mat_p_div(*arg1*,*arg2*)	Základní matematické operace na jednotlivých buňkách matice. Dělení nulou je ošetřeno a výsledek je nula.	in: matice, matice out: matice
ms_avg(*arg1*,*arg2*), ms_stdev(*arg1*,*arg2*), ms_mode(*arg1*,*arg2*), ms_min(*arg1*,*arg2*), ms_max(*arg1*,*arg2*)	Statistické operace nad regionem.	in: matice, matice out: číslo
r_center_point(), r_round_quarter(), r_round_half(), r_round_full(), r_square_quarter(), r_square_half()	Výběry regionů. Byly použity jen kruhové a malé čtvercové regiony pro zachování invariance vůči rotaci.	in: nic out: matice
r_and(*arg1*, *arg2*), r_or(*arg1*, *arg2*), r_xor(*arg1*, *arg2*)	Logické operace nad regiony. Slouží k tvorbě složitějších výběrů.	in: matice, matice out: matice

Tabulka 6.3: Popis terminálů gramatiky

Pomocí gramatiky zadané v tabulce 6.2 byly postupem popsaném v 4.4.2 vytvořeny programy 6.1 a 6.2. Jako první byl nad obrazem program detektoru objektů. Tento označil místa, kde byl nalezen objekt a pouze v odpovídajících místech byl spuštěn klasifikátor.

Program 6.1: Detektor.

```
ms_mode(mat_pow(mat_pow(mat_pow(s_img())),r_square_quarter())
```

Intervaly a nastavení velikosti posuvného okna pro program 6.1:

- Objekt: <-0.004133847245740075, 0.00780259296127764>
- Velikost posuvného okna: 7 pixelů

Program 6.2: Klasifikátor.

```
ms_avg(mat_p_div(mat_p_div(mat_add(s_img(),mat_pow(s_img()
)),mat_pow(mat_add(mat_pow(s_sobel()),s_blur()))),mat_pow(
mat_add(mat_pow(mat_add(mat_pow(s_img()),mat_pow(s_blur())
)),mat_pow(mat_add(mat_pow(s_sobel()),s_blur()))))),  r_rou
nd_full())
```

Intervaly a nastavení velikosti posuvného okna pro program 6.2:

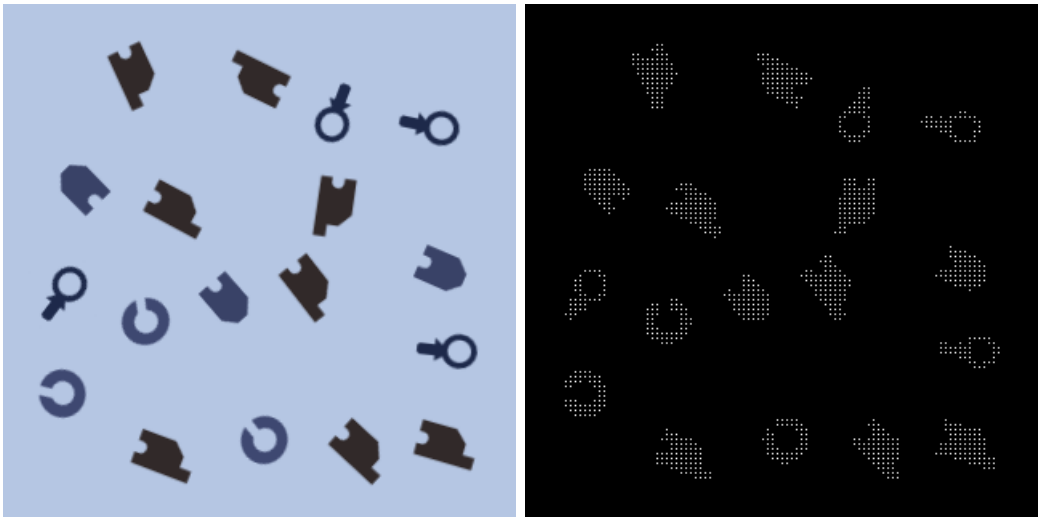
- Class 1: <40.97913746198387, 60.12324904845637>
- Class 2: <385.99089482347165, 566.6794275679941>
- Class 3: <1422.2313057348517, 2091.6064983746714>
- Class 4: <271.6726553671749, 399.10379640103235>
- Class 5: <116.74479981522084, 171.2372449892687>
- Velikost posuvného okna: 45 pixelů

Na obrázku 6.2 je výstup detektoru spuštěného jako posuvné okno s krokem 2. Je vidět, že bílé tečky, značící výskyt objektu dobře překrývají předložené objekty.

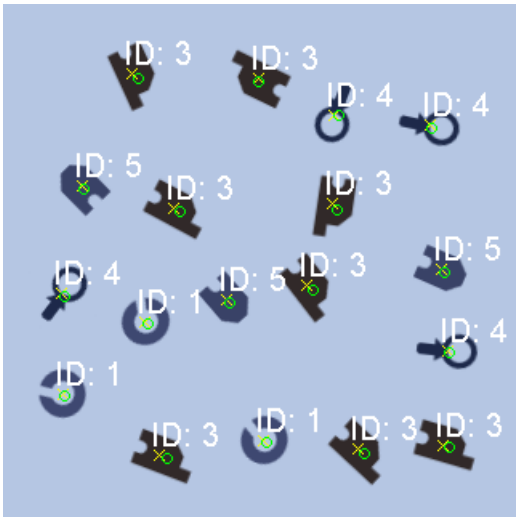
Na obrázku 6.3 je konečné vyhodnocení, které proběhlo na základě analýzy výsledků nad jednotlivými shluky bodů. Nejčastěji udaná hodnota ve shluku byla určena jako výsledek klasifikace objektů. Zelené kolečko značí těžiště objektu nalezeného detektorem a úspěch klasifikace. Žluté křížky jsou středy objektů dané v učících datech.

Další test, který byl proveden se zaměřil na schopnost rozpoznat mírně modifikovaný obraz objektu. Do učící množiny bylo mimo základních obrazů zaneseno i několik obrazů s přidaným náhodným šumem. V tabulce 6.4 jsou výsledky pro testovací obrazy s různými hladinami šumu. Hodnoty šumu jsou udány jako maximální náhodná hodnota, která byla přidána ke každému pixelu. Všechny varianty vstupního obrazu měly hodnoty v intervalu $< 0, 1 >$. Takže například pro řádek tabulky, kde je udána hodnota šumu 0,04 to znamenalo, že k číslům byla přidána až hodnota 0,04.

Čas v tabulce je součtem času detektoru i klasifikátoru. Vždy je ovlivněn počtem nalezených bodů detektorem. Měřeno na procesoru Intel Core2Duo T6500.



Obrázek 6.2: Výstup detektoru



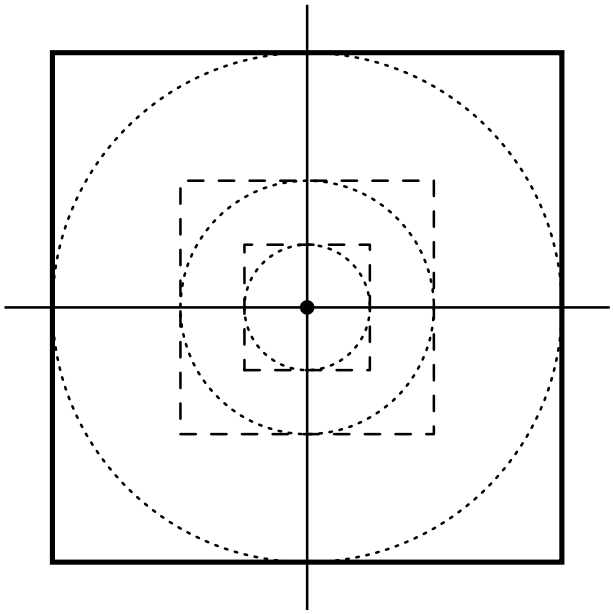
Obrázek 6.3: Vyhodnocený výstup klasifikátoru

Míra šumu	Počet objektů	Počet nalezených bodů detektorem	Úspěšnost [%]	Čas [ms]
0,00	18	1093	100,00	2030
0,02	16	921	100,00	1790
0,04	17	969	94,11	1830
0,06	18	948	38,88	1812
0,08	17	1041	0	1985

Tabulka 6.4: Výsledky klasifikace

6.2.2.1. Regiony pro výpočet statistik

Na obrázku 6.4 jsou znázorněny regiony použité v gramatice definované v tabulce 6.2. Tečkovaně jsou kulaté: *r_round_quarter()*, *r_round_half()* a *r_round_full()*. Čárkovaně jsou čtvercové regiony *r_square_quarter()* a *r_square_half()*. Uprostřed je *r_center_point()*. Tučný okraj znázorňuje hranici posuvného okna.



Obrázek 6.4: Obrazové regiony

Tyto regiony byly kombinovatelné pomocí logických funkcí, například funkcí *r_xor(r_round_half(), r_round_quarter())* bylo možné získat mezikruží. Tyto regiony nebo jejich kombinace pak byly použity jako maska, podle které bylo možné počítat statistické charakteristiky obrazu.

Regiony, které byly použity umožňují vytvořit jen symetrické tvary a zachovávají tak invarianci vůči rotaci a stranovému převrácení.

6.2.3. Klasifikátor využívající operátor středník

V této části práce jsou vypsány nastavení metody popsané v části 4.5 a její výsledky.

Neterminály		Množina přepisů
<expr>	::=	<register_write> | semic(<expr>,<expr>) | if(<logic>,<expr>,<expr>)
<register_write>	::=	rWrite0(<math>) | ... | rWrite*n*(<math>
<math>	::=	<number> | <math_fun>(<math>) | <matrix_stats>(<matrix>) | <register_read>() | <math_op>(<math>,<math>)
<logic>	::=	lgtn(<math>,<math>) | smtn(<math>,<math>)
<matrix>	::=	iIm | iSob | iTh | <matrix_op>(<matrix>,<matrix>) | <matrix_op_simple>(<matrix>,<math>) | <matrix_fun>(<matrix>)

Pokračování na další stránce.

Tabulka 6.5 – *Pokračování z předchozí stránky.*

Neterminály		Množina přepisů
`<number>`	`::=`	`-20.0 | -19.5 | ... | 19.5 | 20.0`
`<math_op>`	`::=`	`add | sub | mul | pdiv`
`<math_fun>`	`::=`	`pow | sqrt | log | exp`
`<matrix_op>`	`::=`	`mAdd | mSub | mMul | mDiv`
`<matrix_op_simple>`	`::=`	`mAddC | mSubC | mMulC`
`<matrix_fun>`	`::=`	`mPow | mSqrt | mLog | mExp`
`<matrix_stats>`	`::=`	`mAvg0_3 | mAvg0_5 | mAvg0_10 | mAvg0_15 | mAvg0_20 | mAvg3_5 | mAvg5_10 | mAvg10_15 | mAvg15_20 | mSum0_3 | mSum0_5 | mSum0_10 | mSum0_15 | mSum0_20 | mSum3_5 | mSum5_10 | mSum10_15 | mSum15_20`
`<register_read>`	`::=`	`rRead0 | ... | rRead`n

Tabulka 6.5: Gramatika použitá k hledání klasifikačního programu

V tabulce 6.6 jsou vysvětleny funkce jednotlivých terminálů gramatiky.

Terminál	Vysvětlení	Vstup Výstup
semic(*arg1*, *arg2*)	Sekvenčně vyhodnotí argumenty *arg1* a *arg2*.	in: cokoliv out: nic
if(*logic*, *arg1*, *arg2*)	Pokud je první argument *logic* vyhodnocen jako pravda, vyhodnotit argument *arg1*, jinak vyhodnotit argument *arg2*.	in: boolean, cokoliv, cokoliv out: nic
lgtn(*arg1*, *arg2*), smtn(*arg1*, *arg2*)	Porovnání argumentů.	in: číslo, číslo out: logická hodnota
iIm	Vstupní uzel – vrací černobílý obraz.	in: nic out: matice
iSob	Vstupní uzel – vrací obraz získaný Sobelovým operátorem pro detekci hran.	in: nic out: matice
iTh	Vstupní uzel – vrací prahovaný obraz.	in: nic out: matice
add(*arg1*, *arg2*), sub(*arg1*, *arg2*), mul(*arg1*, *arg2*), pdiv(*arg1*, *arg2*)	Základní matematické operace: sečtení, odečtení, násobení a dělení. Dělení nulou je ošetřeno, aby výsledek byl nula.	in: číslo, číslo out: číslo

Pokračování na další stránce.

Tabulka 6.6 – *Pokračování z předchozí stránky.*

Terminál	Vysvětlení	Vstup Výstup
pow(*arg*), sqrt(*arg*), log(*arg*), exp(*arg*)	Základní matematické funkce. Odmocnina ze záporného čísla vrací nulu. Logaritmy vrací pro záporná čísla a nulu také nulu.	in: číslo out: číslo
mAdd(*arg1*, *arg2*), mSub(*arg1*, *arg2*), mMul(*arg1*, *arg2*), mDiv(*arg1*, *arg2*)	Základní operace na jednotlivých buňkách matice. Dělení nulou je opět ošetřeno.	in: matice, matice out: matice
mAddC(*arg1*, *arg2*), mSubC(*arg1*, *arg2*), mMulC(*arg1*, *arg2*)	Základní operace na jednotlivých buňkách matice.	in: matice, číslo out: matice
mPow(*arg*,), mSqrt(*arg*), mLog(*arg*), mExp(*arg*)	Základní matematické funkce na jednotlivých buňkách matice. Odmocnina a logaritmus jsou opět ošetřeny.	in: matice out: matice
rWrite$n$(*arg*)	Uložit výsledek vyhodnocení *arg* do registru n.	in: číslo out: nic
rRead$n$()	Čtení hodnoty z registru n.	in: nic out: číslo
mAvgm_n(*arg*)	Vypočte průměrnou hodnotu pixelů v kruhové oblasti mezi poloměry m a n nad maticí *arg*. Kruhové oblasti zajišťují invarianci vůči rotaci.	in: matice out: číslo
mSumm_n(*arg*)	Vypočte součet hodnotu pixelů v kruhové oblasti mezi poloměry m a n nad maticí *arg*. Kruhové oblasti zajišťují invarianci vůči rotaci.	in: matice out: číslo

Tabulka 6.6: Popis terminálů gramatiky

Nejdůležitější parametry nastavené pro hledání klasifikačního programu jsou v tabulce 6.7.

Parametr	Hodnota	Vysvětlení
Počet epoch	150	Počet iterací algoritmu.
Velikost populace	150	Počet jedinců v populaci.

Pokračování na další stránce.

Tabulka 6.7 – *Pokračování z předchozí stránky.*

Parametr	Hodnota	Vysvětlení
Elitismus	Enabled/10 %	Kopírování nejlepších jedinců do další populace. Tato funkce je zapnuta až po 1/10 proběhnutých iterací.
Pravděpodobnost křížení	80 %	Pravděpodobnost křížení, pokud křížení neproběhne, jsou jedinci nakopírováni do další populace. Popsáno v 3.5.2.1.
Pravděpodobnost kombinace programů přes středník	5 %	Pravděpodobnost vytvoření třetího programu jako výsledku křížení, který bude kombinací dvou vybraných (mohou to být jak potomci po křížení, tak nekřížené programy). Popsáno v 4.5.2.1.
Pravděpodobnost mutace	10 %	Pravděpodobnost náhodných změn v chromozomech – dále se u gramatické evoluce dělí na mutaci strukturální a nestrukturální. Popsáno v 3.5.2.2.
Selekce	Turnaj/3	Turnajová selekce – velikost turnaje jsou tři jedinci. Popsáno v 3.3.3.2.
Délka chromozomu	80	Pro výběr pravidel z gramatiky je možné použít maximálně daný počet genů.
Počet registrů	12	Počet registrů, které může program využít. Prvních n se použije pro klasifikaci.

Tabulka 6.7: Parametry nastavené evolučnímu procesu

Program 6.3: Program 1.

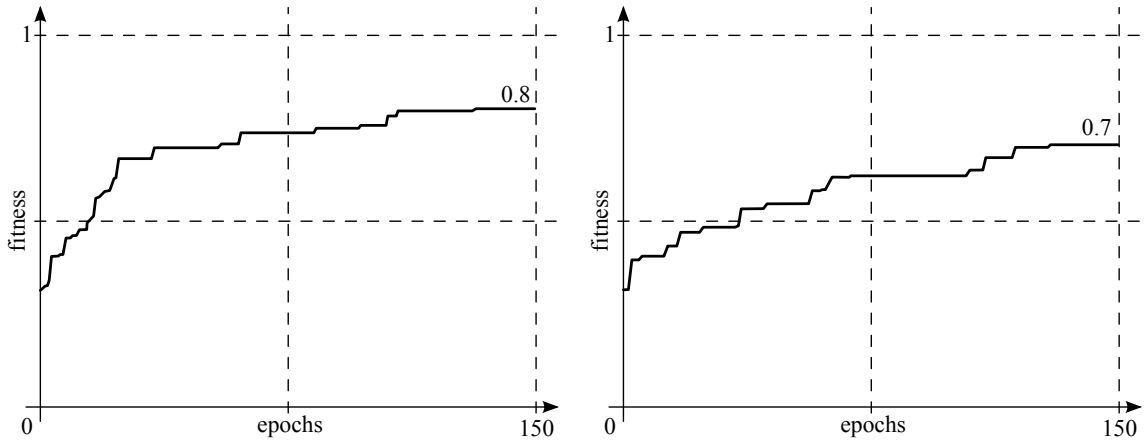
```
semic(semic(rWrite4(mSum10_15(mDiv(iSob,mSub(iIm,iTh)))),rWrite2(log(
add(mSum5_10(iSob),mSum10_15(mDiv(iSob,mSub(iIm,iTh)))))))),if(lgtn(5.
5,mSum5_10(iSob)),semic(rWrite5(pow(log(mSum5_10(mDiv(iIm,mSub(iIm,iT
h))))))),rWrite3(mSum10_15(mDiv(iSob,mDiv(iSob,mSub(iIm,iTh))))))),rWri
te1(mAvg15_20(iTh))))
```

Program 6.4: Program 2.

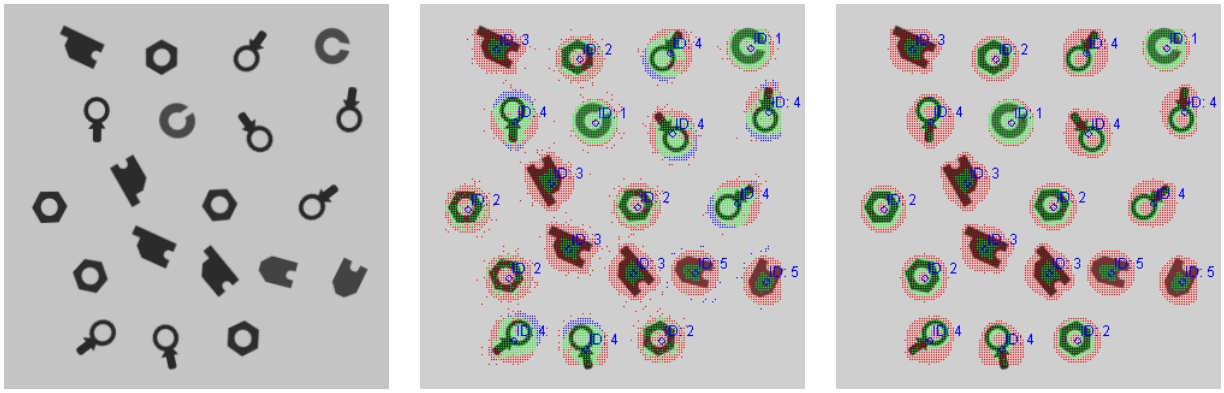
```
semic(if(lgtn(16.5,mSum5_10(iSob)),if(lgtn(5.5,mSum5_10(iSob)),semic(
if(lgtn(5.5,mSum5_10(iTh)),rWrite3(pdiv(mul(5.5,mSum15_20(iSob)),mAvg
0_20(iSob))),rWrite0(16.5)),rWrite5(10.5)),rWrite0(-7.5)),rWrite4(mAv
g0_20(iSob))),rWrite2(mAvg5_10(mExp(mDiv(mSubC(mSubC(iIm,mSum15_20(iS
ob)),mSum15_20(iSob)),iSob)))))
```

Pro ukázku jsou uvedeny 2 programy získané pomocí gramatické evoluce (program 6.3 a program 6.4). Velikost posuvného okna byla nastavena na 41 pixelů.

Programy jsou složitější než v předchozí sekci. První program pracuje s různými obrazovými vstupy, druhý hlavně s obrazem získaným Sobelovým hranovým detektorem (terminál *iSob*). Na obrázku 6.5 je průběh hodnoty funkce fitness při procesu tvorby obou programů.



Obrázek 6.5: Vývoj hodnoty fitness pro program 1 (vlevo) a program 2 (vpravo)

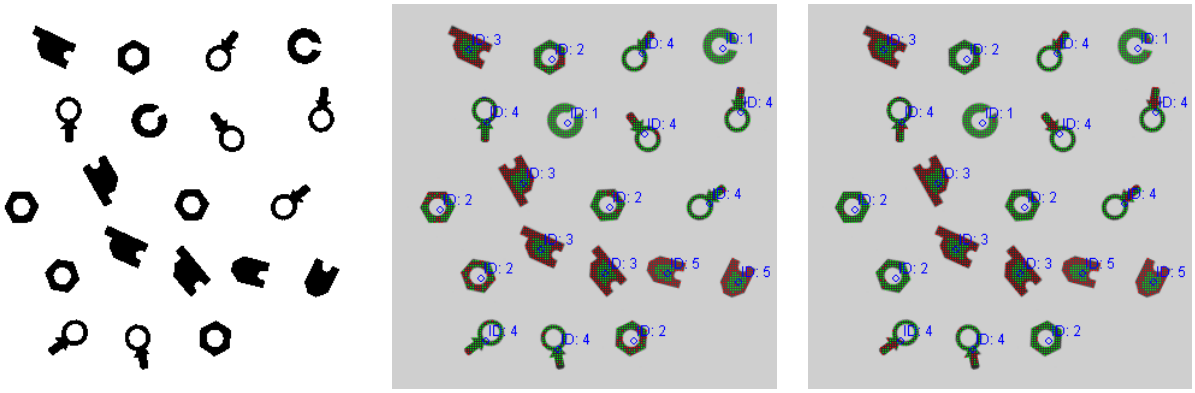


Obrázek 6.6: Testovací scéna (vlevo) a výsledky detekce pro program 1 (uprostřed) a program 2 (vpravo)

Na obrázku 6.6 je výstup obou programů spuštěných nad testovací scénou. Červené tečky jsou nesprávně hlášené výstupy klasifikačního programu. Zelené tečky jsou výstupy shodující se s trénovacími daty v definované vzdálenosti (polovina posuvného okna). Modré tečky jsou správné výstupy ve větší vzdálenosti.

Je vidět, že podobně jako u předchozí verze, má jednofázový výstup hodně nesprávných identifikací v oblasti kolem objektů. Pro zlepšení výstupu bylo provedeno prahování. Klasifikátor byl spouštěn jen v místech, kde prahování nalezlo objekt, obrázek 6.7. Prahování nahradilo v tomto případě program detektoru. Na složitější scéně by ale bylo vhodné použít detektor nebo jinou metodu pro získání objektů z obrazu.

Na výstupech, kde bylo použito prahování je u objektů s otvorem vidět, že definovaná trénovací data, která obsahují střed bodu nejsou úplně vhodná. Jelikož střed objektu leží právě v tomto otvoru. Je to pozorovatelné hlavně u programu 1 (na obrázcích uprostřed). Ten má sice vyšší hodnotu fitness, ale objekty s identifikátorem 2 (matka) rozpoznává hůře. Vyšší hodnotu fitness má, protože má některé správné detekce blíž středu objektu



Obrázek 6.7: Prahovaný obraz (vlevo) a výsledky detekce pro program 1 (vlevo) a program 2 (vpravo) – prahování

(objekt 2 a objekt 4). Možná oprava by byla modifikací trénovacích dat nebo úpravou vzorce pro výpočet skóre 4.7.

Časová náročnost byla měřena jen orientačně. Program 1 potřebuje asi 0,5 sekundy a program 2 potřeboval přibližně sekundu pro ohodnocení všech míst daných prahovaným obrazem. Tedy 2238 bodů pro krok posuvného okna 2. Měřeno na procesoru Intel Core2Duo T6500.

6.3. Zhodnocení výsledků

Navržené metody jsou, co se týká úspěšnosti velice dobré. Problémy nastávají, pokud klasifikátoru předložíme objekty, na které není připraven (například je obraz jinak nasvícen nebo je zašumělý). Toto se dá ovšem odstranit rozšířením množiny trénovacích vzorů a přidáním vhodných terminálů do gramatiky.

Pokud jde o rychlost rozpoznávání, hlavním faktorem, který definuje dobu trvání procesu je počet spuštění programu. S rostoucí plochou obrazů bude kvadraticky růst i počet spuštění. Přístup s detektorem nebo prahováním sice umožní běh programu zrychlit, ale na větších obrazech bude trpět stejným nedostatkem, jen ne v takové míře. Řešením je ponechat lokalizaci objektů jiné metodě, což je ostatně v praxi běžným postupem.

Přístupem rozpoznávání s posuvným oknem poměrně značně prodloužíme i čas tvorby klasifikátoru, neboť je nutné spustit všechny programy v populaci nad všemi vzory pro učení.

V porovnání s neuronovou sítí je proces učení o mnoho delší, neboť při učení neuronové sítě pracujeme jen s jedním klasifikátorem. U evolučních metod máme klasifikátorů právě tolik, kolik je členů populace.

U gramatické evoluce je vhodné spustit proces učení vícekrát, abychom získali více klasifikátorů, ze kterých vybereme ten nejlepší. Zajímavou možností je vložit vybrané kvalitní jedince do nového procesu gramatické evoluce a nechat je dále zlepšovat. Případně je možné výsledek procesu gramatické evoluce ručně doladit. To u neuronové sítě nelze.

V produkční fázi má použitý princip posuvného okna stejný vliv i na neuronovou síť. Struktura představených klasifikátorů je jednodušší než struktura neuronové sítě. Proto bude rozpoznávání objektů pomocí vyvinutých klasifikátorů rychlejší.

Díky tomu, že jsou vyvinuté klasifikátory, co se týká struktury, obvykle jednodušší než neuronové sítě, může být jednodušší jejich aplikace v tzv. *embedded* řešeních (systémy se specifickou funkčností, často realtime). Je to proto, že nejsou nutné speciální knihovny a je menší prostorová i časová náročnost výpočtu. Schopnost vyvinout program přímo v odpovídajícím programovacím jazyku je také využitelná.

Silným argumentem pro použití navržené metody je možnost získat invarianci vůči rotaci a stranovému převrácení na úrovni klasifikátoru. Tímto požadavkem se sice prodlouží učení, jelikož je zmenšena množina obrazových regionů, které můžeme použít k výpočtu statistik, nicméně v produkční fázi se odstraní fáze hledání výchozí polohy objektu.

7. Závěr

Cílem práce bylo navržení metody, která použije gramatickou evoluci k tvorbě krátkých počítačových programů, které jsou použitelné jako klasifikátory pro rozpoznávání objektů v obraze. Hlavní aplikační oblastí výsledků je průmyslová robotika, kde je požadována vysoká přesnost a rychlost rozpoznávání.

Hlavním výstupem této práce je ucelený postup rozpoznávání objektů v obrazech, založený na gramatické evoluci. Dále byly navrženy vzorce pro výpočet hodnot funkce fitness. Tyto postupy jsou použitelné i obecně mimo oblast počítačového vidění. Inovativní je i řešení používající operátor středník a registry, k získání vícehodnotového výstupu z klasifikačního programu. Tento vícehodnotový výstup se dá lépe vyhodnotit a případně může posloužit jako vstup do dalšího stupně klasifikace. Dalším hodnotným výstupem je popsaný způsob implementace algoritmu značkování a překladu chromozomu na spustitelné stromy.

Pro testování algoritmů bylo také vytvořeno programové prostředí v jazyce Java, které může dále posloužit pro výuku nebo řešení reálných problémů. V tomto prostředí není implementována jen gramatická evoluce, ale obecný mechanismus evolučních metod.

Gramatická evoluce a evoluční metody obecně nejsou nejrychlejšími metodami pro řešení optimalizačních úloh. Je to proto, že pracují s populací kandidátních řešení a jejich hromadné ohodnocování zabírá nejvíce času. Pokud tedy existuje algoritmus, který je navržen přímo pro řešení nějaké konkrétní úlohy, obvykle evoluční metodu překoná. Naopak nepřekonatelnou výhodou těchto algoritmů, je jejich schopnost hledat řešení problémů, které jsou z pohledu klasických metod neřešitelné ať už proto, že neexistuje konkrétní metoda nebo je prohledávaný prostor řešení příliš velký. Vysvětlením je, že evoluční algoritmy poskytují jakýsi obecný návod, jak tyto složité problémy řešit, aniž bychom o nich museli příliš vědět. Stačí najít způsob jak možná řešení problému zakódovat do chromozomu a ohodnotit.

Metody gramatické evoluce jdou ještě dále a oprošťují jejich uživatele od nutnosti navrhnout reprezentaci chromozomu a mechanismu křížení či mutace. Stačí jen dodat funkci pro výpočet hodnoty fitness a gramatiku. Výhodou může být i výstup ve zvoleném formátu – dáno použitou gramatikou. Navíc dokáže gramatická evoluce optimalizovat i strukturu výsledného počítačového programu.

Pro použití gramatické evoluce je tedy nejdůležitější navrhnout správně funkci pro výpočet hodnoty fitness a dát evoluční metodě takové nástroje, aby mohla zvyšovat kvalitu jedinců podle této funkce. Je důležité si uvědomit, že pokud je fitness funkce chybně nadefinována, evoluční metoda bude optimalizovat jedince k co nejlepším výsledkům, ale ti budou k řešení úkolu nepoužitelní.

Výhoda navržených metod je v tom, že navržené klasifikátory mají obecně méně složitou strukturu než neuronové sítě. Jejich použití je proto vhodné právě v aplikacích, kde je k dispozici omezený výpočetní výkon a operační paměť.

7.1. Přínos práce

7.1.1. Přínos vědecký

Hlavním přínosem práce jsou tyto dosažené výsledky:

- Ucelený postup rozpoznávání objektů v obrazech, založený na gramatické evoluci.
- Vzorce pro výpočet fitness použitelné v klasifikačních problémech.
- Navržení terminálů středník a registr, které umožní programu vyvinutému gramatickou evolucí mít vícehodnotový výstup a zachovat snadné křížení a mutaci pomocí výměny podstromů.
- Popis efektivního algoritmu pro značkování chromozomu.
- Popis tvorby spustitelného programového stromu během překladu chromozomu.

Během řešení bylo publikováno několik článků na téma gramatická evoluce a její využití v aplikacích. Hlavním tématem bylo rozpoznávání objektů, ale gramatická evoluce byla testována i pro klasifikaci jiných dat a pro predikci časových řad: [21, 22, 23, 41].

7.1.2. Praktický

Praktickým přínosem je vytvoření testovacího prostředí pro spouštění a vývoj metod založených nejen na gramatické evoluci a aplikace gramatické evoluce do oblasti, kde se tradičně používají jiné metody umělé inteligence.

Testovací prostředí může být využito k demonstraci funkce evolučních metod studentům. Případně použito k řešení dalších úloh.

Literatura

[1] APACHE FOUNDATION: *Subversion*. Online: http://subversion.apache.org/

[2] BAY, H. ESS, A., TUYTELLAARS, T.: Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*. Vol. 110, 2008, No. 3, pp. 346-–359. ISSN 1077-3142.

[3] BEAUMONT, D., STEPNEY, S.: Grammatical Evolution of L-systems. In *CEC '09. IEEE Congress on Evolutionary Computation*, 2009, pp. 2446–2453, 2009. ISBN 978-1-4244-2958-5.

[4] BEBIS, G., et al.: Genetic Object Recognition Using Combinations of Views. *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 2, pp. 132–146, 2002, ISSN 1089-778X

[5] BHANU, B., SUNGKEE, L., DAS, S.: Adaptive image segmentation using genetic and hybrid search methods. *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 31, No. 4, pp. 1268–1291, 1995. ISSN 0018-9251.

[6] BOHÁČ, M., LÝSEK, J., et al.: Face Recognition by Means of New Algorithms. In *MENDEL 2011, 17th International Conference on Soft Computing*. Brno University of Technology, Czech Republic, pp. 324–329, 2011, ISBN 978-80-214-4302-0.

[7] BURKE, E.K. GUSTAFSON, S. KENDALL, G.: Diversity in Genetic Programming: An Analysis of Measures and Correlation With Fitness. In *IEEE Transactions on Evolutionary Computation*. Vol. 8, No. 1, pp. 47–62, 2004. ISSN 1089-778X.

[8] ČÍŽEK, L., ŠŤASTNÝ, J.: Comparison of Genetic Algorithm and Graph-based Algorithm for the TSP. In *MENDEL 2013, 19th International Conference on Soft Computing*. Brno University of Technology, Czech Republic, pp. 433–438, 2013. ISSN 1803-381, ISBN 978-80-214-4755-4.

[9] CORTES, C., VAPNIK, V. N.: Support-Vector Networks. *Machine Learning*, Vol. 20, 1995.

[10] DAE N. CHUN, HYUN S. YANG: Robust Image Segmentation Using Genetic Algorithm with a Fuzzy Measure. *Pattern Recognition*, Vol. 29, No. 7, pp. 1195–-1211, 1996. ISBN 1-59593-186-4.

[11] ECLIPSE FOUNDATION: *Eclipse*. Online: http://www.eclipse.org/

[12] GARAI, G., CHAUDHURI, B. B.: A Novel Grid Pattern Matching Technique with Hybrid Genetic Algorithm. In *TENCON 2009 - IEEE Region 10 Conference*, pp. 1–6, ISBN 978-1-4244-4546-2.

[13] GHOSH, P., MITCHELL, M.: Segmentation of Medical Images Using a Genetic Algorithm. In *GECCO '06 Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pp. 1171–1178, 2006.

[14] HAN LIU, DING LIU, JING XIN: Real-time Recognition of Road Traffice Sign in Motion Image Based on Genetic Algorithm. In *Proceedings of International Conference on Machine Learning and Cybernetics*, pp. 83–86, 2002, ISBN 0-7803-7508-4

[15] HOPCROFT, J. E., ULLMAN, J. D.: *Formal languages and their relation to automata*. Boston: Addison-Wesley, 1969, 288 p. ISBN 0-201-02983-9.

[16] HOPFIELD, J. J.: Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences of the USA*, Vol. 79, No. 8, pp. 2554–2558, 1982.

[17] KOHONEN, T.: Self-Organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, 43. 79, No. 1, pp. 59–69, 1982.

[18] KOZA, J. R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, 1992, ISBN 0-262-11170-5.

[19] KYU-PHIL HAN, KUN-WOEN SONG, EUI-YOON CHUNG, SEOK-JE CHO, YEONG-HO HA: Stereo matching using genetic algorithm with adaptive chromosomes. *Pattern Recognition*, Vol. 34, No. 9, pp. 1729–1740, 2000. ISSN 0031-3203

[20] LOWE, G. D.: Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, Vol. 60, No. 2, pp. 91–110, 2004, ISSN 0920-5691.

[21] LÝSEK, J., ŠŤASTNÝ, J., MOTYČKA, A.: Object Recognition by means of Evolved Detector and Classifier Program. In *MENDEL 2012, 18th International Conference of Soft Computing*, Brno University of Technology, Czech Republic, pp. 82-–87, 2012, ISSN 1803-3814, ISBN 978-80-214-4540-6.

[22] LÝSEK, J., ŠŤASTNÝ, J., MOTYČKA, A., Comparison of Neural Network and Grammatical Evolution for Time Series Prediction. In *MENDEL 2013, 19th International Conference on Soft Computing*. Brno University of Technology, Czech Republic, 2013, pp. 215-220, ISSN 1803-3814, ISBN 978-80-214-4755-4.

[23] LÝSEK, J., ŠŤASTNÝ, J., Classification of Economic Data into Multiple Classes by Means of Evolutionary Methods. In *Enterprise and Competitive Environment 2013*, Brno, Czech Republic, 2013.

[24] McCULLOCH, W., PITTS, W.: A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, pp. 115–133, 1943.

[25] OBJECT AID: *Object Aid UML Explorer*. Online: http://www.objectaid.com/

[26] ORACLE CORP.: *JAVA*. Online: http://www.java.com/

[27] ORTEGA, A., DALHOUM, A. A., ALFONSECA, M.: Grammatical evolution to design fractal curves with a given dimension. *IBM Journal of Research and Development*, Vol. 47, No. 4, pp. 483–493, 2003. ISSN 0018-8646.

[28] PEREZ, B. C., OLAGUE, G.: Evolutionary learning of local descriptor operators for object. In *GECCO '09, Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 1051–1058, USA, 2009, ISBN 978-1-60558-325-9.

[29] POPELKA, O., RUKOVANSKÝ, I., OŠMERA, P.: Grammatical evolution with backward processing. In *Proceedings of Fourth internation conference on soft computing applied in computer and economic enviroments*, pp. 89-96, 2006. ISBN 80-7314-084-5.

[30] POPELKA, O. ŠŤASTNÝ, J.: *Uplatnění metod umělé inteligence v zemědělsko-ekonomických predikčních úlohách*, 48 p., MENDELU, 2009, ISBN 978-80-7375-340-5

[31] POPELKA, O., ŠŤASTNÝ, J.: WWW Portal Usage Analysis Using Genetic Algorithms. In *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis*, 2009, Vol. 6, pp. 201–208, ISSN 1211-8516.

[32] PROCESSING ORG.: *Processing*. Online: http://www.processing.org/

[33] PRUSINIKEWICZ, P., LINDENMAYER, A.: *The algorithmic beauty of plants*, 228 p., 1991. ISBN 0-387-97297-8.

[34] ROTH, G., LEVINE, D. M.: Geometric Primitive Extraction Using a Genetic Algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 16, No. 9, pp. 901–905, 1994, ISSN 0162-8828

[35] ROUPEC, J.: *Vývoj genetického algoritmu pro optimalizaci parametrů fuzzy regulátorů*, disertační práce. Brno: VUT, 2001.

[36] RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J.: Learning representations by back-propagating errors. *Nature*, Vol. 323, pp. 533–536, 1986.

[37] RYAN, C., O'NEILL, M.: *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer, 2003. 160 pp. ISBN 978-1-4020-7444-8

[38] SANG KYOON KIM, DAE WOOK KIM, HANG JOON KIM: A recognition of vehicle licence plate using a genetic algorithm based segmentation. In *Proceedings of International Conference on Image Processing*, pp. 661–664, 1996. ISBN 0-7803-3259-8.

[39] SEETHARAMAN, G. S.: A Genetic Coding Structure and Algorithms for Image Segmentation. *Biologically Structured Computing Systems*, 1997

[40] SUGANTHAN, P. N.: Structural pattern recognition using genetic algorithms. *Pattern Recognition*, Vol. 35, No. 9, pp. 1883–-1893, 2002, ISSN 0031-3203

[41] ŠKORPIL, V., LÝSEK, J., et al.: Grammatical Evolution as a Learning Process for Multiclass Object Detection. In *Recent Researches in Communications and Computers*, 16th WSEAS International Conference on Communications, Kos, Greece, 2012, pp. 101-105, ISBN 978-1-61804-109-8.

[42] ŠŤASTNÝ, J., ŠKORPIL, V.: *Analysis of Algorithms for Radial Basis Function Neural Network*, Personal Wireless Communications, Springer, 2007, Issue 1, pp. 54–62, ISSN 1571-5736, ISBN 978-0-387-74158-1.

[43] TSANG, P. W. M.: A genetic algorithm for affine invariant recognition of object shapes from broken boundaries. *Pattern Recognition Letters*, Vol. 18, No. 7, 1997, pp. 631–-639, ISSN 0167-8655

[44] WINKLER, J. F., MANJUNATH, B. S.: Genetic Programming for Object Detection. *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 330–335, 1997.

[45] ZHANG, M., CIESIELSKI, V. B., ANDREAE, P.: A domain-independent window approach to multiclass object detection using genetic programming. *EURASIP Journal on Applied Signal Processing*, Vol. 2003, No. 8, pp. 841–859, 2003, ISSN 1687-6180.

[46] ZHANG, M., CIESIELSKI, V.: Genetic Programming for Multiple Class Object Detection. *Advanced Topics in Artificial Intelligence*, pp. 180–192, 1999, ISBN 978-3-540-66822-0.

8. Seznam použitých zkratek a symbolů

CCD	technologie digitálních snímačů
CMOS	technologie digitálních snímačů
RGB	barevný model – červená, modrá zelená
HSV	barevný model – tón, sytost, jas
RGBA	barevný model – červená, modrá zelená, alfa
CMYK	barevný model – azurová, purpurová, žlutá, černá
MLP	multi layer perceptron – dopředná neuronová síť
SVM	support vector machines – architektura klasifikátoru
g	gen chromozomu
ch	chromozom – vektor genů
G	genotyp – soubor chromozomů
M	jedinec
P(t)	populace v daném čase
Fit(M)	vzorec pro výpočet hodnoty fitness jedince

Seznam obrázků

Seznam tabulek

9. Seznam příloh

Na přiloženém CD je program jako projekt (*Genetic Algorithm*) pro vývojové prostředí Eclipse. Jsou přiloženy i další vytvořené projekty (*NeuralNet*, *Helpers*, *Graph*), které hlavní projekt používá. Vše je nutné naimportovat do Eclipse a hlavnímu projektu přidat do *Build Path* podpůrné projekty. Poté lze spouštět jednotlivé úlohy. Některé projekty vyžadují k vykreslování grafiky knihovnu Processing [32].

Seznam spustitelných zdrojových kódů v Java projektu Genetic Algorithm:

- /src/gaTSP/ga/tsp/test/Main.java – ukázka řešení problému obchodního cestujícího
- /src/gaMinimalisationProblem/ga/minimalisation/test/Main.java – ukázka hledání minimální hodnoty funkce
- /src/de/de/test/Main.java – ukázka diferenciální evoluce, hledání souřadnic minima funkce tří proměnných
- /src/geAproxProblem/ge/aprox/test/Main.java – ukázka regrese dat
- /src/geObjectRecognition/ge/recognition/test/Main.java – rozpoznávání objektů, hledání klasifikátorů
- /src/geClassificationProblem/ge/classify/test/Main.java – ukázka klasifikace dat do tříd
- /src/gePrediction/ge/prediction/test/Main.java – ukázka predikce
- /src/nnPrediction/nn/prediction/Main.java – ukázka predikce řešené neuronovou sítí, vyžaduje projekt *NeuralNet*