KATEDRA INFORMATIKY A POČÍTAČŮ


# Studentská vědecká konference katedry informatiky a počítačů 2017

11.5.2017

# Sborník příspěvků

Název: Studentská vědecká konference katedry informatiky a počítačů 2017
Editor: RNDr. Tomáš Sochor
Nakladatel: Ostravská univerzita, Dvořákova 7, 701 03 Ostrava
Rok prvního vydání: 2017
ISBN: 978-80-7464-919-6
Počet stránek: 13
Vydání: první



Texty příspěvků neprošly jazykovou úpravou

Obsah

# Pavla Grossmannová:
# Procedurální generování terénu s možností hledání cest

Studium: Ostravská univerzita, bakalářský studijní obor Informatika, ročník třetí
Studijní číslo: R14314

Bakalářská práce se zabývá generováním terénu a hledáním cesty v mapě. Její první část popisuje vybrané metody. Druhá část se zabývá vytvořenou desktopovou aplikací, která slouží k demonstraci vybraných algoritmů.

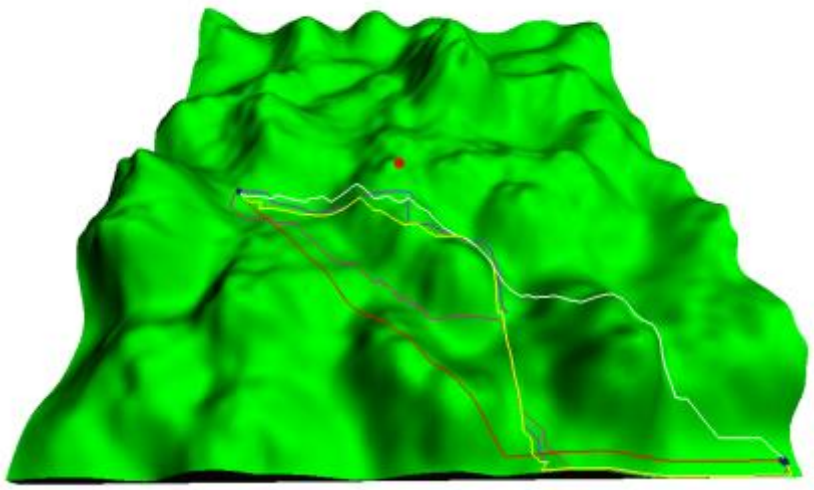
Aplikace je napsána pomocí programovacího jazyku Java, verze 1.8.0_20. Pro usnadnění práce bylo použito vývojové prostředí NetBeans IDE 8.0.2. Dále bylo použito OpenGL, což je programové rozhraní grafického hardwaru, ke kterému se přistupuje přes knihovnu LWJGL.

Grafické rozhraní aplikace se skládá z tří hlavních částí. První je okno pro zobrazení terénu. Druhou je část umístěná vpravo, umožňující komunikaci uživatele s programem a poslední částí je textový výpis dole pod oknem, kde po průběhu hledání cesty můžeme vidět hodnoty výsledků pro jednotlivé algoritmy.

Ze všech metod pro vytváření terénu byly vybrány 3 a to metoda náhodných poruch, kopcový algoritmus a hodnotový šum. Důvodem tohoto výběru byl fakt, že právě tyto metody neposkytují na první pohled příliš líbivý výstup, na rozdíl například od Perlinova šumu. Šlo o to zjistit, zda z něčeho takového lze následnými úpravami získat dobře vypadající terén.

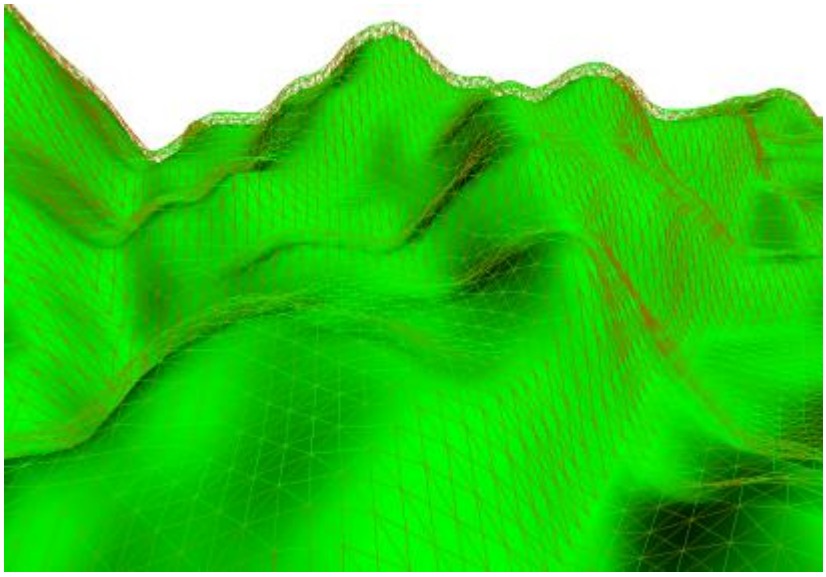
Vhodnou kombinací úprav terénu vzniká velká škála různých tvarů. Zde je použito vyhlazování, dilatace a dvě možnosti eroze.

Pro demonstraci funkce algoritmů, které hledají cestu, byl zvolen Dijkstrův algoritmus, který vždy najde nejkratší cestu, pak také A* ve třech různých variantách a jako poslední algoritmus prohledávání do šířky, ten ovšem na rozdíl od předchozích nezohledňuje ohodnocení hran.



*Obrázek 1. Ukázka nalezených cest*

Aplikace rovněž umožňuje grafické znázornění náročnosti přechodů mezi jednotlivými uzly grafu, mezi kterými se hledají cesty. Navíc je díky pohyblivé kameře možno nahlížet na vytvořený terén z různých úhlů a výšky.



*Obrázek 2. Barevné ohodnocení hran*

# David Číž:
# Rozpoznávání číslic pomocí hlubokých neuronových sítí

Studium: Ostravská univerzita, bakalářský studijní obor Aplikovaná informatika, ročník třetí
Studijní číslo: R14428

Aplikace hlubokých neuronových sítí se dnes ve světě využívají zcela nekompromisním způsobem a je masivně nasazováno na nepřeberné množství problémů. Metody rozpoznávání znaků se dnes běžně používají na poštách pro čtení adres a popisných čísel. Tyto techniky se stále zlepšují a se současným trendem neuronových sítí se jejich využití rozšiřuje. Jsou použity při rozeznávání SPZ, čtení značek, překladu textu v reálném čase pomocí kamery v telefonu a dalších. Díky velkému množství dat, které poskytuje internet a výkonu dnešních superpočítačů se neuronové sítě mohou učit téměř cokoliv. V současné době se zlepšují algoritmy pro rozeznávání objektů a kategorizaci, ale i generování hlasu, hudby, textu.

Tento příspěvek pojednává o základních konceptech neuronových sítí, způsobu jejich učení a jejich využití pro rozpoznávání obrazu. Cílem praktické části této studie bylo vytvoření hluboké neuronové sítě jako metody, která dokáže rozpoznávat obrázky z databáze MNIST, čili číslice. K tomuto účelu bylo použito prostředí TensorFlow, kvůli jejich tutoriálům a přehlednému způsobu psaní kódu. Vytvořená metoda byla implementována v jazyce Python Nejlepším způsobem pro rozeznávání obrazu se ukázalo využití konvoluční neuronové sítě, dosahující přesnosti 99,2 %.


***Obrázek 1: Graf výsledné přesnosti konvoluční sítě***

Na obrázku č.1 můžete vidět, že okolo 12 000. iterace dosahuje chybová funkce lokálního minima a rozpoznává číslice téměř bezchybně. Celá navržená metoda byla porovnána v oblasti úspěšnosti rozpoznávání číslic s klasickou vícevrstvou neuronovou sítí adaptovanou metodou backpropagation, která byla v základním nastavení horší o 7,2 % celkové úspěšnosti rozpoznávání. Pro dosažení vyšší přesnosti by se musely otestovat jiné architektury, pravděpodobně daleko hlubší a robustnější.

Hluboké neuronové sítě se masivně využívají i pro zpracování BigData a vzhledem k jejich výpočetní náročnosti by bylo velice zajímavé ve studii nadále pokračovat formou trénování a testování těchto sítí, kde by se využilo paralelní zpracování dat pomocí superpočítačů například na IT4I.

# Michal Jalůvka:
# Strojové učení pro adaptivní báze pravidel

**Úvod:**
Tato práce se zabývá problematikou strojového učení, a to konkrétně oblastí, jakým způsobem je možné modifikovat znalosti systému. Motivací této práce je najít stejný přístup strojového učení jako učení člověka. Jako například ukázat systému, jak se má chovat, pohybovat, nebo mu přikázat, co má dělat a později to opakovat. Znalosti systému jsou reprezentovány bázemi pravidel, jehož jazykový popis je ve tvaru IF-THEN pravidel. Cílem práce je navrhnout přístup strojového učení umožňující adaptaci bází pravidel.

**Rozbor současného stavu:**
Vzhledem k tomu, že nebyly dohledatelné žádné práce zaměřující se na modifikaci znalostí systému představující báze pravidel, která se adaptuje dle předloženého vzoru, byl navržen vlastní přístup.

**Návrh vlastního přístupu:**
Adaptace báze pravidel je reprezentována změnou a výběrem určitých pravidel, které jsou vhodné pro danou úlohu. Tento výběr lze chápat jako deduktivní učení. Pokud máme docílit vhodnosti správných pravidel, lze použít přístup strojového učení s učitelem nebo zpětnovazebného učení (učení posilováním). Při použití přístupu strojového učení s učitelem je důležitá existence trénovací množiny (vzorů). Jako způsob získání vzoru byl volen online přístup, tj. získání vzoru během učení.
Tyto výše zmíněné přístupy strojového učení (deduktivní učení, učení s učitelem, online učení) tvoří základ vlastního přístupu při návrhu metody strojového učení.
Architektura systému, která realizuje vlastní přístup strojového učení, se skládá ze tří částí:

- Fuzzy inferenční systém
- Učitel (Supervisor)
- Konečný automat

Fuzzy inferenční systém slouží pro rozhodování. Znalosti, z nichž čerpá a rozhoduje se, jsou popsány v RB souborech. Proces rozhodování je zajištěn jádrem LFLCore, což je jádro softwaru LFLC (IFRAM, 2003), který byl vytvořen v ústavu IRAFM.
Konečný automat slouží pro přepínání bází pravidel a zamezuje používání jedné robustní báze pravidel. Učitel slouží ke generování, posilování a vybírání nejvhodnějších pravidel pro danou činnost (dovednost), která má být naučena.
Průběh učení, jimž se učitel řídí, je rozdělen do těchto základních fází:

- Načtení vzoru
- Výběr pravidel
- Inference a chybovost
- Posilování pravidel
- Aktualizace pravidel

Ve vzorech, které učitel používá, jsou obsaženy tyto informace:

- Požadovaný stav, který má nabývat
- Číslo kroku daného požadovaného stavu
- Bázi pravidel, nad kterým se bude provádět proces rozhodování

**Dosažení výsledků:**

Vlastní přístup strojového učení byl odzkoušen na několika příkladech, ve kterých učíme systém vykonat činnosti skládající se z určitých kroků. Každý krok představuje pohyb jednotlivého kloubu. V průběhu učení každého experimentu docházelo v každém kroku k výběru a posilování pravidla a podle tohoto pravidla se kloub pohnul. Ve výsledku se podařilo adaptovat jednotlivé báze a vybrat nejvhodnější pravidla pro danou činnost.

**Literatura:**IFRAM. (2003). *LFLC.* Ostrava: Ostravská univerzita.
Machová, K. (2002). *Strojové učenie: principy a algoritmy.* Košice: Elfa.
Mitchell, T. M. (1997). *Machine learning.* Boston: McGraw-Hill.
Novák, V. (2000). *Základy fuzzy modelování* (1. vyd.). Praha: BEN - technická literatura.

# Milan Zenka:
# Colander - nástroj pre analýzu sieťovej prevádzky

Práca sa zaoberá tvorbou NIDS programu pre výskumné účely. Program bol nazvaný Colander. NIDS, teda „Network intrusion detection system“, je program určený na monitorovanie sieťovej komunikácie za účelom odhalenia potencionálne škodlivej aktivity. Hlavným spôsobom detekcie podozrivej aktivity je využitie signatúr. Signatúry sú súborom pravidiel, zameraných napríklad na podozrivé vzory dátovej časti paketov, porty využívané na iné ako bežné účely, alebo neštandardný tvar paketov (hlavička, či údaje v oblastiach, kde sa bežne nevyskytujú, porušovanie RFC).Ďalším spôsobom detekcie je nájdenie anomálii. Tento spôsob spočíva v analýze bežnej sieťovej komunikácie a v následnom vyhľadávaní anomálneho správania. Táto práca sa zaoberá len signatúrovou detekciou.
Najrozšírenejšie existujúce NIDS riešenia analyzované pri tvorbe tejto práce a k nim relevantné údaje:

- Snort
  - najstarší NIDS projekt,
  - najväčšia komunita používateľov ,
  - zameraný čo najširšie, od nadšencov, cez výskumníkov, až po firemnú sféru,
  - najrozšírenejšie pravidlá – SNORT/TALOS,
  - pravidlá nie sú špecializované, používateľ buď použije všetky, alebo musí sám vybrať vhodné,
  - technicky zastaraný, nepodporuje multithreading.
- Suricata
  - zameraný na firemnú sféru,
  - používa pravidlá Emerging Threats , SNORT/TALOS, a pre detekciu anomálii skriptovací jazyk Lua,
  - Emerging Threats pravidlá sú rozdelené podľa typov hrozieb alebo oblasti záujmu, čo uľahčuje ich použitie,
  - technicky pokročilý, podporuje účinný multithreading a dokonca GPU akceleráciu na veľmi rýchle paralelné porovnávanie.
- Bro Network Security Monitor
  - zameraný na univerzitné a firemné prostredie, ako nástroj na komplexnú analýzu sieťovej prevádzky,
  - používa pravidlá Emerging Threats , SNORT/TALOS, a pre detekciu anomálii skriptovací jazyk Bro scripting language,
  - zameriava sa hlavne na detekciu anomálii, signatúrová detekcia je sekundárna,
  - technicky komplexný, podporuje multithreading a distribúciu záťaže na ostatné uzly v sieti používajúce Bro.

Hlavnými rozdielom medzi existujúcimi riešeniami a touto prácou, je jej distribuovaná podstata, zameranie na OS Windows, a jednoduché použitie. Colander je inštalovaný na čo najvyššom počte bežných užívateľských počítačov s OS Windows, klientov, na ktorých prebieha analýza sieťovej komunikácie. Distribuovaná podstata spočíva v tom, že zachytené hrozby a informácie o nich sa z klientov posiela na server k spracovaniu bezpečnostným analytikom, pričom užívateľ klientského počítača nemusí do tohto procesu nijak zasahovať.

Všetky existujúce riešenia vyžadujú relatívne zložitú inštaláciu a nastavovanie, pričom len Bro poskytuje možnosť zasielania zachytených hrozieb na centrálny server. V prípade Colandera užívateľ len nechá prebehnúť inštalátor a môže na neho zabudnúť, funguje autonómne. V prípade záujmu je mu poskytnuté jednoduché GUI, ktoré však nemusí používať.
Na tvorbu boli využité jazyky C# pre Windows klienta a GUI, a Python pre serverovú časť.
Colander využíva upravené pravidlá SNORT/TALOS, vzhľadom na ich všeobecné zameranie. Úprava spočíva v odstránení nepotrebných a nepoužitých údajov pravidiel za účelom zníženia ich veľkosti. Použité pravidlá môžu byť menené bezpečnostným analytikom podľa jeho posúdenia a aktuálnej situácie. V prípade potreby je však možné použiť neupravenéSNORT/TALOS pravidlá.
Klientský program má dve verzie. Prvá bola vytvorený s dôrazom na optimalizáciu a minimálne vyťaženie zdrojov klientského počítača a nepoužíva multithreading a je určená hlavne na jednojadrové procesory . Je však schopná analyzovať menej paketov z celkovej komunikácie. Druhá verzia využíva multithreading, pri ktorom je vytvorené vlákno pre každý analyzovaný paket, čím pri bežných prenosových rýchlostiach minimalizuje strata paketou. Pri prenášaní veľkého množstva dát pri vysokých rýchlostiachvšak môže dôjsť k značnému vyťaženiu procesora, čo môže vadiť užívateľom. Táto verzia je určená pre viac jadrové procesory.
Serverová časť ukladá dáta zaslané klientami do MySQL databázy. Obsahuje aj jednoduché webové stránky určené na zobrazenie prehľadu zachytených hrozieb.
V testovacej prevádzke bol klient aktívny na malom počte užívateľských počítačov. Ukázala sa pri nej prílišná všeobecnosť SNORT/TALOS pravidiel, spôsobujúca mnohé zachytenia falošných hrozieb, hlavne pravidlá pre „denial of service“ útoky považujúce za hrozbu aktualizácie Windows 10.
Pri lokálnom testovaní na infikovanom systéme bol klient schopný zachytiť škodlivý softvér, ak využíval sieťovú komunikáciu, hlavne trójske kone.
Colander je účinný nástroj pre zber dát o sieťových hrozbách na bežných užívateľských počítačoch. Analytik ho môže zmenou a tvorbou nových pravidiel upraviť pre aktuálne hrozby. Plánujem jeho ďalší rozvoj, pridanie nových funkcii a anomálnej detekcie pri štúdiu PhD.

# Diferenciální evoluce s kovarianční maticí

Jessica Křížková

May 11, 2017

## Diferenciální evoluce

Diferenciální evoluce (*DE*) je efektivní evoluční algoritmus, který je široce používán pro řešení optimalizačních problémů. DE navrhli autoři R. Storn a K. Price a byla poprvé publikována v roce 1995. Algoritmus DE pracuje s populací a vytváří novou generaci tak, že postupně pro každý bod $\vec{x}_i$ populace vytvoří jeho konkurenta $\vec{u}_i$ a do nové populace zařadí lepšího z této dvojice, podle nižší objektivní funkce $f(\vec{u}_i) \leq f(\vec{x}_i)$.

## Algoritmus CoBiDE

Algoritmus *CoBiDE* je rozšíření algoritmu DE o kovarianční matici a byl poprvé publikován v roce 2014 [1]. Tento algoritmus kromě kovarianční matice využívá, stejně jako klasická DE, dva hlavní parametry: mutační faktor $F$ a kontrolní parametr křížení $CR$. Tyto parametry mají významný vliv na výkon *DE*, a proto jsou v rámci *CoBiDE* generovány z bimodálního Cauchyho rozložení.

V mé práci je algoritmus implementován v prostředí NetBeans v jazyce Java. Algoritmus je testován na nových testovacích funkcích CEC2014. Tyto funkce jsou v porovnání s testovacími funkcemi CEC2005 ,na kterých byl CoBiDE původně navržen, rozšířené o dalších 5 testovacích funkcí. Na každou z těchto funkcí bylo spuštěno 25 běhů a pro každý běh byla zaznamenána odchylka od skutečného řešení - *error*. Z vypočtených hodnot erroru byly sestaveny následující experimentální statistické hodnocení.

## Hodnocení experimentu

Cílem experimentu je porovnat účinnost algoritmu *CoBiDE* s jinými algoritmy DE, které patří mezi tzv. state-of-the-art: *JADE*, *jDE*, *SaDE*, *CoDE*, *IDE*, *EPSDE* a *SHADE*. Podle porovnání algoritmu *CoBiDE* s každým uvedeným algoritmem, byly ve většině funkcí rozdíly. Z tohoto důvodu byl proveden nejprve obecnější Friedmanův test. Pro nastavení dimenze vektorů v algoritmu *CoBiDE* $D = 10$ i pro nastavení dimenze $D = 30$ (viz tabulka 1), byl algoritmus v tomto testování vždy na 3. místě v pořadí hned za algoritmy *IDE* a *JADE*.

Pro Friedmanův test byly použity naměřené hodnoty mediánů pro všechny testovací funkce. Můžeme vidět průměrné pořadí každého algoritmu a konečné seřazení od nejúčinnějšího algoritmu.

Table 1: Friedmanův test $D = 10$ a $D = 30$

| D=10 | | | D=10 | | |
|---|---|---|---|---|---|
| Alg | Median | Mean rank | Alg | Median | Mean rank |
| IDE | 0.31 | 2.68 | IDE | 17.76 | 3 |
| JADE | 0.45 | 3.13 | JADE | 53.66 | 3.87 |
| CoBiDE | 0.20 | 3.53 | CoBiDE | 48.44 | 4.13 |
| SHADE | 1.31 | 4.22 | jDE | 19.11 | 4.38 |
| jDE | 0.46 | 4.65 | SaDE | 44.37 | 4.73 |
| SaDE | 0.82 | 4.80 | SHADE | 49.47 | 5.15 |
| EPSDE | 2.08 | 6.17 | EPSDE | 68.16 | 5.22 |
| CoDE | 2.29 | 6.82 | CoDE | 62.06 | 5.52 |

Pro detailní analýzu rozdílů mezi CoBiDE a vybranými algoritmy byl aplikován Wilcoxonův dvouvýběrový test. Výsledky tohoto testu můžeme vidět v tabulce 2. V tabulce jsou znázorněny počty, kolikrát byl algoritmus *CoBiDE* významně lepší/kolikrát byl *CoBiDE* významně horší/počet funkcí, kde porovnávané hodnoty byly shodné. Můžeme vidět, že nejhorší výsledky dosáhl CoBiDE porovnáním s IDE. Naopak u zbývajících algoritmů byly výsledky CoBiDE přinejmenším porovnatelné, mnohdy podstatně lepší.

Table 2: Wilcoxonův test

| CoBiDE vs. | CoDE | EPSDE | IDE | JADE | jDE | SaDE | SHADE |
|---|---|---|---|---|---|---|---|
| *D = 10* | 24/1/5 | 24/2/4 | 9/17/4 | 10/15/5 | 19/6/5 | 19/6/5 | 14/12/4 |
| *D = 30* | 20/7/3 | 19/8/3 | 8/18/4 | 12/14/4 | 13/13/4 | 10/16/4 | 16/14/0 |

Dále pro ověření účinnosti kovarianční matice a bimodálních parametrů ($F$ a $CR$) byla každá z těchto části v algoritmu odebrána a testována bez nich. U výsledků algoritmu bez kovarianční matice, kde byly ponechány bimodální parametry, byl významnější rozdíl než u výsledků algoritmu bez bimodálních parametrů, kde byla ponecháno křížení kovarianční maticí. Z tohoto experimentu jsme ověřili, že využití kovarianční matice má zásadní vliv na výsledky CoBiDE. S kombinací těchto parametrů a kovarianční matice můžeme algoritmus *CoBiDE* zařadit mezi užitečné evoluční algoritmy pro hledání globálního optima. Naopak samotná kovarianční matice nejlepších výsledků nedosahovala.

# References

[1] Yong Wang, Han-Xiong Li, Tingwen Huang, Long Li: Differential evolution based on covariance matrix learning and bimodal distribution parameter setting. In: *Applied Soft Computing*, vol. 18, May 2014, pp. 232–247, ISSN 1568-4946.