Syntéza neuronových sítí

Neural network synthesis

Bc. Marek Malinka

Diplomová práce
2011

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2010/2011

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Marek MALINKA**
Osobní číslo:	**A09514**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Syntéza neuronových sítí**

Zásady pro vypracování:

1. **Vypracujte literární rešerši na dané téma.**
2. **Popište metodu syntézy neuronových sítí pomocí Analytického Programování.**
3. **Seznamte se s existující asynchronní programovou implementací.**
4. **Navrhněte experimenty zaměřené na syntézu neuronových sítí.**
5. **Proveďte navrženou sérii experimentů.**
6. **Statisticky vyhodnoťte provedené experimenty.**
7. **Vypracuje závěr a doporučení z výsledků experimentů vyplívající.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. ZELINKA, Ivan, et al. Evoluční výpočetní techniky – principy a aplikace. Vyd. 1. Praha : BEN technická literatura, 2008. 534 s. ISBN 80-7300-218-3.
2. HYNEK, J. Genetické algoritmy a genetické programování. 1.st ed. 2008. ISBN 978-80-247-2695-3.
3. MAŘÍK, V.; ŠTĚPÁNKOVÁ, O.; LAŽANSKÝ, J. Umělá inteligence (4). 1.st ed. 2003. ISBN 80-200-1044-0.
4. ZELINKA, I. Umělá inteligence I. 1st ed. 1998. ISBN 80-214-1163-5.
5. BOSE, N.K.; LIANG, P. Neural network fundamentals with graphs, algorithms, and applications. 1.st ed. 1996. ISBN 0-07-006618-3.

Vedoucí diplomové práce: **Ing. Bc. Pavel Vařacha**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **24. února 2011**

Termín odevzdání diplomové práce: **18. května 2011**

Ve Zlíně dne 24. února 2011

L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

ABSTRAKT

Cílem diplomové práce je vypracování literární rešerše na téma syntézy neuronových sítí a vysvětlení principu algoritmu Analytického programování. Dále popis použité implementace algoritmu Asynchronního analytického programování. V praktické části jsou navrženy experimenty, které mají za úkol zjistit, jakou má implementovaný algoritmus použitelnost v různých úlohách. A z těchto experimentů vyplívající doporučení.

Klíčová slova: Syntéza neuronových sítí, Asynchronní analytické programování, Umělá neuronová síť, Topologie neuronové sítě

ABSTRACT

Subject of diploma work is elaboration literary research on the topic synthesis neural network and explication principle algorithm analytic programming. Description used implementation algorithm asynchronous analytic programming. In practical parts are designed experiments that be tasked find out, of what has implemented algorithm usability in different exercises. And of these experiments resulting recommendation.

Keywords: Neural network synthesis, Asynchronous analytic programming, Artificial neural network, Neural network topology

Za odborné vedení, poskytnutí podkladů pro mou práci a trpělivost při konzultacích děkuji vedoucímu bakalářské práce Ing. Bc. Pavlu Vařachovi.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně ……

podpis diplomanta

OBSAH

ÚVOD

Umělé neuronové sítě mohou principielně napodobit funkce nervových soustav živých organismů. Konvenční počítače pracují s předem daným algoritmem. Neuronové sítě pracují bez algoritmu a jejich funkce je založena na principu učení. Síť je schopná se postupně přizpůsobit k řešení dané úlohy.

Základním prvkem neuronové sítě je neuron. Funkce neuronové sítě je ovlivněna především její topologií a způsobem, jakým jednotlivé neurony transformují signál, který se šíří ze vstupů sítě na její výstupy. Pod pojmem topologie neboli struktura neuronové sítě je myšleno konkrétní umístění neuronů v síti a jejich vzájemné propojení.

Každá neuronová síť jako celek realizuje určitou transformační funkci. Tedy převádí (transformuje) hodnoty vstupních veličin na hodnoty výstupní. Každá naučená neuronová síť tedy realizuje jednu konkrétní matematickou funkci.

Implementace algoritmu asynchronního Analytického programování popsaná v této práci umí neuronovou síť nejenom naučit tedy adaptovat její váhy, ale také syntetizovat optimálně její strukturu. Využívá k tomu schopnosti Analytického programování hledat matematické funkce. Jak již bylo řečeno, takovou matematickou funkcí je také neuronová síť. Analytické programování je metodou symbolické regrese a pro svůj běh potřebuje nějaký evoluční algoritmus. V této práci je použit evoluční algoritmus SOMA.

Práce si dále klade za cíl navržení a provedení experimentů s touto implementací a vypracování konkrétních doporučení pro práci s tímto algoritmem.

I. TEORETICKÁ ČÁST

1 LITERÁRNÍ REŠERŠE NA TÉMA SYNTÉZA NEURONOVÝCH SÍTÍ

1.1 Evoluční výpočetní techniky – principy a aplikace

[1] ZELINKA, Ivan, et al. Evoluční výpočetní techniky – principy a aplikace. Vyd. 1. Praha : BEN technická literatura, 2008.

Asi nejrozsáhlejší a nejobsáhlejší publikace v českém jazyce podávající kompletní přehled o evolučních výpočetních technikách a optimalizačních metodách. Kniha obsahuje také část, kde je popis principu algoritmu SOMA. Dále je zde také poměrně rozsáhlá kapitola týkající se evoluce symbolických struktur včetně symbolické regrese a Analytického programování. Dále také genetické programování a gramatická evoluce. Publikace obsahuje také malou kapitolu o syntéze neuronových sítí. Celá kniha je velmi dobře a čtivě napsána a rozhodně stojí za pozornost.

1.2 Genetické algoritmy a genetické programování

[2] HYNEK, J. Genetické algoritmy a genetické programování. 1.st ed. 2008. ISBN 978-80-247-2695-3.

Kniha Josefa Hynka přináší ucelený pohled na problematiku genetických algoritmů. Jsou v ní vysvětleny principy fungování genetických algoritmů, cesty k efektivnímu využívání evolučních technik a genetické programování.

Součástí jsou i příklady konkrétních problémů a rozbor různých algoritmů použitelných pro jejich řešení.

1.3 Umělá inteligence (4)

[3] MARÍK, V.; ŠTEPÁNKOVÁ, O.; LAŽANSKÝ, J. Umelá inteligence (4). 1.st ed. 2003. ISBN 80-200-1044-0.

Kniha kolektivu autorů Mařík, Štěpánková, Lažanský z vícedílné rozsáhlé série knih o umělé inteligenci. V tomto čtvrtém dílu je věnováno několik kapitol neuronovým sítím. Celkem asi 100 stran ze 470ti, ve kterých je úvod do problematiky neuronových sítí, popis různých modelů sítí, komprese dat a také aproximace funkcí neuronovými sítěmi.

1.4 Umělá inteligence I

[4] ZELINKA, I. Umelá inteligence I. 1st ed. 1998. ISBN 80-214-1163-5.

Tato skripta autora Ivana Zelinky jsou výstižně a stručně napsaným ale, obsáhlým výkladem neuronových sítí. Vysvětlení jejich funkcí, popisu modelů, ale také navržení optimální topologie a gradientní a evoluční algoritmy učení neuronových sítí.

V další části je popisována příprava tréninkové množiny, optimální start učení a obecnější příklady použití. Tyto skripta lze doporučit jako velmi čtivě a zajímavě napsané i když, v některých věcech trochu stručné.

1.5 Syntéza neuronových sítí metodou symbolické regrese

[5] VAŘACHA, Pavel. Syntéza neuronových sítí metodou symbolické regrese. Zlín, 2006. 83 s. Diplomová práce. UTB.

Tato diplomová práce se zabývá syntézou NS pomocí evolučního prohledávání za pomoci Analytického programování. Jedná se o práci, která je asi nejblíže tématu řešenému zde v této diplomové prací. Dá se říci, že tato práce na ni navazuje a snaží se rozšířit poznatky z oblasti syntézy NS a použitelnosti Analytického programování. Motivací je automatizace syntézy NS a nalezení dosud neznámých řešení. Zároveň však klade další otázky týkající se syntézy NS. Například jak zhodnotit kvalitu nalezených řešení které úspěšně řeší daný problém nebo pro jaké typy problémů je tato metoda použitelná.

1.6 Analytické programování v C#

[6] KASPŘÍKOVÁ, Eva. Analytické programování v C#. Zlín, 2008. 67 s. Diplomová práce. UTB.

V této diplomové práci jsou popsány tři algoritmy symbolické regrese. A to Genetické programování, Gramatická evoluce a Analytické programování. Jako evoluční algoritmus pro chod Analytického programování je popisována Diferenciální evoluce a algoritmus SOMA. Součástí práce je také popis implementace Analytického programování v jazyce C# a vyhodnocení výsledků s touto implementací dosažených.

1.7 Analytic programming

[7] OPLATKOVÁ, Zuzana. Analytic programming. Zlín, 2003. 73 s. Diplomová práce. UTB.

Diplomová práce zabývající se tématem Analytického programování. Je zde srovnání metody genetického programování s Analytickým programováním pomocí simulačních testů provedených na čtyřech problémech popsaných J.Kozou v [9]. Výsledky těchto simulací ukazují, že Analytické programování je srovnatelné s genetickým programováním. Výhodou Analytického programování však je, že na rozdíl od genetického programování lze použít jakýkoliv evoluční algoritmus.

1.8 Analytic programming - Symbolic regression by means of arbitrary evolutionary algorithms

[10] ZELINKA, Ivan; OPLATKOVÁ, Zuzana; NOLLE, Lars. Analytic programming. 2004. Symbolic regression by means of arbitrary evolutionary algorithms, s. 44-56.

Článek představující Analytické programování jako novou metodu symbolické regrese. Demonstruje hlavní pravidla Analytického programování a jeho schopnosti syntetizovat vhodná řešení. Srovnává také Analytické programování s genetickým programování a gramatickou evolucí. Pro Analytické programování je zde použita diferenciální evoluce a algoritmus SOMA.

1.9 Proben1

[12] PRECHELT, L. Proben1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules. [online]. Available from www: http://digbib.ubka.uni-karlsruhe.de/volltexte/39794>

Jedná se o důležitou publikaci autora Lutze Prechelta, která předkládá jednak pravidla a dohody pro provádění experimentu s neuronovými sítěmi, ale také osahuje odkaz na konkrétní reálná data vhodné pro testování výkonu neuronových sítí. Jde v podstatě o povinnou četbu pro toho, kdo chce seriózně provádět nějaké experimenty s výkonem učících algoritmů na neuronových sítích. Tedy tak, aby byly srovnatelné a reprodukovatelné.

1.10 Optimal feed-forward neural network arcitectures

[13] BABIS, G., GEORGIOPOULOS, .M. Optimal feed-forward neural network architectures. [online]. Available from www: http://www.cse.unr.edu/~bebis/optimal_arch.pdf

Článek Gerge Bebise a Michaela Georgiopoulose zabývající se nalezením optimální topologie pro neuronové sítě s dopředným šířením. Jsou zde uvedeny také zajímavé algoritmy pro automatické nalezení vhodné topologie sítě pro konkrétní řešený problém. Tedy algoritmy modifikace síťové architektury. A to konstruktivní a destruktivní metoda. Jde o postupné budování a testování struktury NS od nejmenší po větší (konstruktivní) nebo od velké po menší (destruktivní).

1.11 Neuroevolúcia cez rozširovanie topologie

[14] KRIŠKA, J., MAKULA, M. Neuroevolúcia cez rozširovanie topológie. [online]. Available from www: <http://hilbert.chtf.stuba.sk/KUZV/download/kuzv-kriska-makula.pdf>

Jedná se o článek Jozefa Kriška a Mateje Makula popisující přístup NEAT, tedy evoluční adaptaci topologie a vah neuronové sítě začínající od minimální možné topologie a její postupné rozšiřování.

Pro lepší efektivitu zapracovali do svého algoritmu způsob jak identifikovat shodnost genů, ochranu nových částí genu a minimalizaci jejich struktury. Provedli také několik experimentů.

2 METODA SYNTÉZY NEURONOVÝCH SÍTÍ POMOCÍ ANALYTICKÉHO PROGRAMOVÁNÍ

2.1 SOMA: SamoOrganizující se Migrační Algoritmus

SOMA je algoritmus, který je založený podobně jako DE na vektorových operacích. Pracuje také s populacemi podobně jako GA. Není však tvořena vždy nová populace, ale pracuje se v migračních kolech. Kdy je populace pouze přeskupena. Přesnější je tedy řadit tento algoritmus mezi memerické či hejnové algoritmy.

Na rozdíl od GA v algoritmu SOMA neprobíhá v migračním kole křížení rodičů, ale je kooperativně prohledávám prostor všech možných řešení. Řídí se tedy principy odpozorovanými ze spolupráce inteligentních biologických jedinců při migraci v krajině. Jedinci se tedy vzájemně ovlivňují při hledání nejlepšího řešení čímž vzniká v algoritmu SOMA samoorganizace.

2.1.1 Parametry algoritmu SOMA

Běh SOMA je ovlivňován nastavením svých parametrů, které se dělí na dva druhy a to řídící a ukončovací. Řídící parametry mají vliv na kvalitu běhu algoritmu a ukončovací mají vliv na ukončení běhu algoritmu, určují tedy kdy už je hledané řešení dostatečně přesné.

- **PathLength** $\in (1,5]$**.** Délka cesty určuje, jak daleko se jedinec zastaví od vedoucího jedince. Pokud je PathLength=1, pak se zastaví přesně na pozici vedoucího jedince. Pokud je PathLength=2, pak se zastaví za vedoucím jedincem ve stejné vzdálenosti, ve které od něj stál. Většinou je dostatečná PathLength=3.
- **Step** $\in (0.11; PathLength]$**.** Krok určuje, jak jemně bude prozkoumána cesta jedince. Při nízké hodnotě tohoto parametru bude tedy cesta jedince prozkoumána podrobněji z hlediska možných řešení.
- **PRT** $\in [0,1]$**.** PRT je perturbace. Tímto parametrem je ovlivňován perturbační vektor. Perturbace v SOMA je obdobou mutace u klasických GA. Je to důležitý parametr a určuje, zda se aktivní jedinec pohybuje přímo k vedoucímu jedinci. Pro PRT=1 se SOMA chová pouze podle deterministických pravidel.

- **D** je dimenze účelové funkce. Tento parametr je určen řešeným problémem.
- **PopSize** $\in [10...]$**.** Počet jedinců v populaci
- **Migrace** $\in [10...]$**.** Je ekvivalentem počtu generací u GA. Udává tedy počet migračních kol v algoritmu.
- **MinDiv** je ukončovací parametr. Minimální diverzita definuje, jaký je maximální rozdíl mezi nejlepším a nejhorším jedincem povolen než je algoritmus ukončen.

2.1.2 Princip algoritmu SOMA

Jak již bylo naznačeno, SOMA pracuje v migračních kolech a tyto migrační kola jsou podobné generacím v GA. Jedinci jsou však v každém migračním kole přemísťováni pomocí sekvence pozic do finální pozice. Tyto sekvence jsou počítány k pozici vedoucího jedince a počet kroků v sekvenci je dán hodnotou parametru Step.

2.1.3 Fáze SOMA

Zde je popsána základní verze strategie SOMA a to AllToOne [11]:

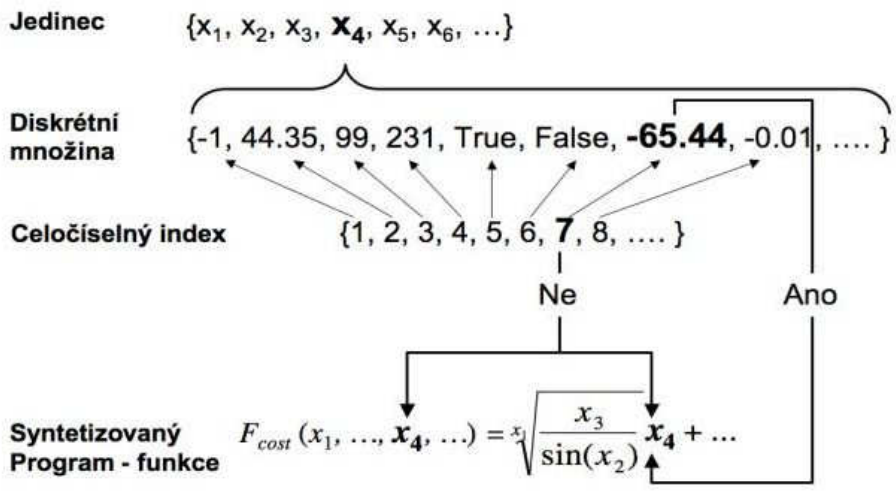
1. **Definice parametrů.** Před startem algoritmu je pochopitelně potřeba nadefinovat všechny parametry a účelovou funkci.
2. **Tvorba populace.** Počáteční populace je pomocí specimenu náhodně vygenerována.
3. **Migrační kola.** Každý jedinec je ohodnocen účelovou funkcí a dle hodnot účelové funkce je zvolen vedoucí jedinec. Potom se začnou ostatní jedinci pohybovat směrem k vedoucímu jedinci pomocí skoků. Velikost skoků je dána parametrem Step. Po každém skoku jedinec zjišťuje svoji aktuální hodnotu účelové funkce a pokud je zatím nejlepší, tak si ji zapamatuje. Pohyb po skocích pokračuje, dokud není dosaženo pozice dané parametrem PathLength. Po dosažení této pozice se jedinec vrací do místa skoku s nejlepší hodnotou účelové funkce. Všichni jedinci kromě vedoucího jsou tedy přemístěni. Před tím, něž jedinec začne skákat směrem k vedoucímu jedinci je vygenerován prázdný PRTvektor s dimenzí D. K tomu je vygenerována řada náhodných čísel o počtu D. Tyto náhodné čísla se porovnávají se zadaným parametrem PRT. Jestli je N-té náhodné číslo větší, než PRT, potom je

N-tý parametr PRTvektoru nastaven na nulu, jinak na jedničku. PRTvektor se potom násobí vektor udávající pohyb jedince směrem k vedoucímu. V dimenzích, kre je v PRTvektoru nula je tedy pohyb jedince zmražen. Tomuto rušení pohybu jedince směrem k vedoucím se právě říká perturbace. Pokud by tato perturbace neexistovala, choval by se algoritmus deterministicky.

4. **Testování ukončovacích parametrů.** Zde je testováno zda je rozdíl mezi nejlepším a nejhorším jedincem menší, než MinDiv. A také zda byly vykonány migrační cykly v počtu, jenž je dán parametrem Migrace. Pokud není žádní z těchto podmínek splněna pokračuje se krokem 3.
5. **Stop.** Návrat nejlepšího jedince.

2.2 Práce s diskrétními hodnotami DSH

Obecně jsou evoluční algoritmy schopné pracovat s binárními, reálnými, celočíselnými, ale také s diskrétními hodnotami. Pro množinu diskrétních hodnot, např. {-1; 3.14; true; false; 20} je sestaven celočíselný index, který odpovídá konkrétní hodnotě z diskrétní množiny. Parametr jedince v evolučním algoritmu je potom sestaven z těchto celočíselných indexů, které nabývají hodnot {1, 2, 3, 4, 5 … }. S takovým jedincem sestaveným z náhradních celočíselných hodnot se potom v evolučním algoritmu normálně pracuje. Rozdíl je jenom v ohodnocení účelové funkce, do které se dosadí konkrétní diskrétní hodnota reprezentovaná celočíselným indexem v parametru jedince. Tento princip vystihuje



Obrázek 1 Princip DSH [1]

2.3 Analytické programování

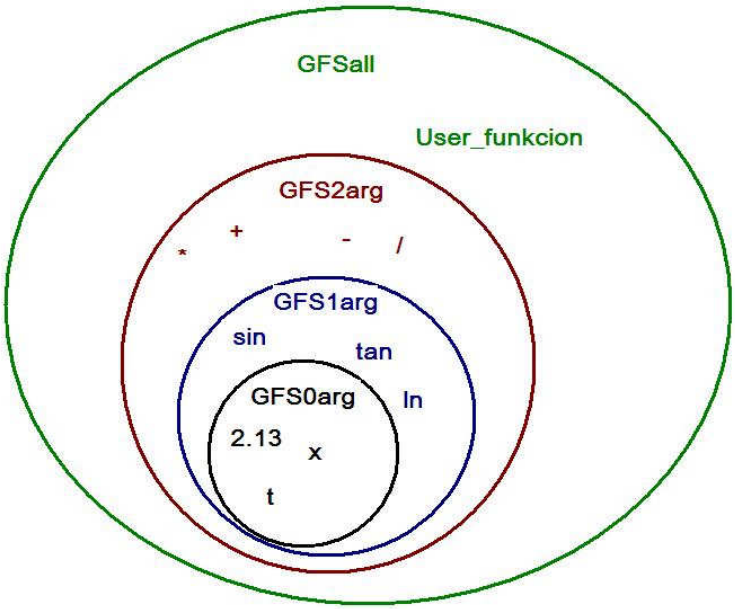
Analytické programování je experimentální metoda, kterou je možné brát jako alternativu ke GP (genetickému programování) a GE (gramatické evoluci). GP i GE jsou podrobně popsány např. v [1] GP potom také v [2]. Na rozdíl od GP a GE však AP není vázáno na jeden konkrétní evoluční algoritmus. AP tedy není samostatný evoluční algoritmus, ale spíše transformace z množiny nějakých symbolických objektů do množiny konstrukcí (programů), které je možné z těchto základních symbolických objektů zkonstruovat. AP tedy pro svůj běh potřebuje nějaký evoluční algoritmus, jako vhodné se jeví DE (diferenciální evoluce) nebo SOMA [5].

2.3.1 Základní princip AP

AP bylo inspirováno jednak GP a také numerickými metodami v Hilbertově funkcionálním prostoru. Stejně jako GP je i AP založeno na množině funkcí, operátorů a terminálů což jsou již zmíněné základní symbolické objekty. Z Hilbertových prostorů je pak převzata myšlenka funkcionálního prostoru a budování výsledné funkce pomocí prohledávání tohoto prostoru. U AP se jedná o evoluční prohledávání. V množině všech možných funkcí, které lze sestavit ze základních symbolických objektů však budou zcela jistě také funkce patologické, tedy takové, které by principiálně nemohly fungovat. Vyřazení takových funkcí z evoluční syntézy je úkolem účelové funkce.

- Funkce: Sin, Cos, Tan, Tanh, And, Or, ...
- Operátory: +, -, *, /, dt, ...
- Terminály: 2, 87, 1.345, x, t …

U AP se takováto množina objektů nazývá GFS (general funkcional set). V AP se tyto objekty setřídí podle počtu argumentů. Počet argumentů je potom určujícím faktorem pro použití symbolického objektu. Podle počtu argumentů je tedy vytvořena hierarchická struktura podmnožin. Jednotlivé podmnožiny obsahují objekty se stejným počtem argumentů Obrázek 2. Tyto podmnožiny jsou označeny GFS_{all} (množina všech objektů GFS), $GFS_{0\,\mathrm{arg}}$ (terminály, tedy funkce s 0 argumenty), $GFS_{1\,\mathrm{arg}}$ (funkce s jedním argumentem), $GFS_{2\,\mathrm{arg}}$ (operátory, tj. funkce se dvěma argumenty).

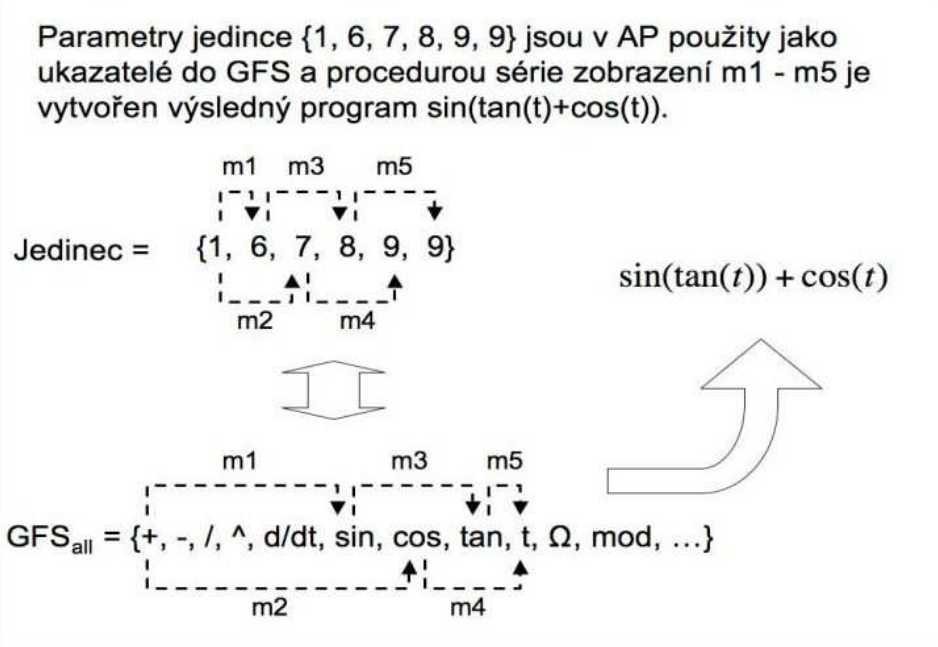


Obrázek 2 Hierarchie GFS podle počtu argumentů

Základní princip AP je založen na DSH (Discrete Set Handling) manipulaci s diskrétními množinami. Syntéza výsledných funkcí běží zpočátku v GFS_{all}, ale postupně se zaměřuje na objekty z oblasti funkcí s nižším počtem argumentů.

Toto je prováděno pomocí mapovacích operací, které transformují jedince na použitelnou funkci. Toto mapování je složeno ze dvou částí a to DSH a bezpečnostních procedur. Je totiž nezbytné zajistit, že každý jedinec bude reprezentovat jednoznačnou a nedefektní funkci.

Nejlépe osvětlí mapování jedince na funkci příklad převzatý z [1] :



Obrázek 3 Mapování celočíselného jedince pomocí AP na funkci [1]

První parametr jedince má hodnotu 1. Je tedy vybrán první objekt z GFS_{all} což je operátor +, ten chápeme jako funkci dvou argumentů. Je tedy nutné zjistit, které další dva objekty budou z GFS_{all} vybrány jako argumenty. Jsou to indexy 6 a 7. Označení $m1$ je mapování prvního argumentu operátoru + a $m2$ je mapování druhého argumentu. V tomto případě jsou to funkce sin a cos, které mají po jednom argumentu. Následuje mapování $m3$ a $m4$ argumentů těchto dvou funkcí. Mapování $m3$ ukazuje na funkci tan, která tedy bude argumentem funkce sin a $m4$ namapuje jako argument funkce cos terminál t. Zbývá tedy doplnit argument funkce tan pomocí $m5$, což je opět terminál t. Jedinec {1,6,7,8,9,9} potom odpovídá výrazu $\sin(\tan(t)) + \cos(t)$. Toto není jediná možná reprezentace jedince, pokud bude pořadí funkcí v GFS_{all} jiné, bude jiná také reprezentace jedince. Tak mohou vnikat také patologičtí jedinci. Aby k tomu nedocházelo slouží právě již zmíněné bezpečnostní procedury, které neustále měří vzdálenost do konce celočíselného jedince. A podle této vzdálenosti se určuje z jaké podmnožiny GFS se budou funkce mapovat. Pokud je již konec jedince blízko jsou vybírány funkce z podmnožiny GFS s menším počtem argumentů. Pokud by byla v předchozím příkladě na poslední pozici jedince ne 9, ale 3. Pak by do funkce tan bylo namapováno /, které má však dva argumenty a vznikla by tak patologická funkce $\sin(\tan(?/?)) + \cos(t)$. Zbývá-li tedy poslední volný index a poslední

funkce s chybějícím argumentem, pak je vybrán objekt z podmnožiny $GFS_{0\arg}$. Tímto způsobem je výraz uzavřen a není patologický.

2.3.2 Posílená evoluce

Během evoluce jsou syntetizováni různě úspěšní jedinci. Některé úspěšnější jedince je možné použít k posílení probíhající evoluce. Princip spočívá v tom, že vybraní jedinci jsou přidáni do množiny terminálů. Tedy do GFS. Výběr jedinců je řízen uživatelem nastaveným prahem a aktuální hodnotou CF (Cost Function) jedince. AP potom v další evoluci pracuje s takto posílenou množinou GFS obsahující částečně úspěšná řešení.

Takto posílené AP je schopné najít řešení mnohem rychleji. Tento fakt byl verifikován úspěšnými simulacemi [5].

2.3.3 Cost function

AP lze použít na evoluční hledání funkce, která co nejlépe proloží zadané body. Jako vhodná CF je potom součet absolutních hodnot rozdílu zadaných bodů a hodnot aktuálně hodnocené funkce v těchto bodech. AP se snaží minimalizovat tento rozdíl a tedy proložit funkcí co nejlépe zadané body.

2.4 Syntéza neuronových sítí pomocí AP

Základem použití AP pro syntézu NS je vhodně zvolená množina GFS a definování CF. Nejprve je ale třeba osvětlit vztah mezi NS a funkcemi.

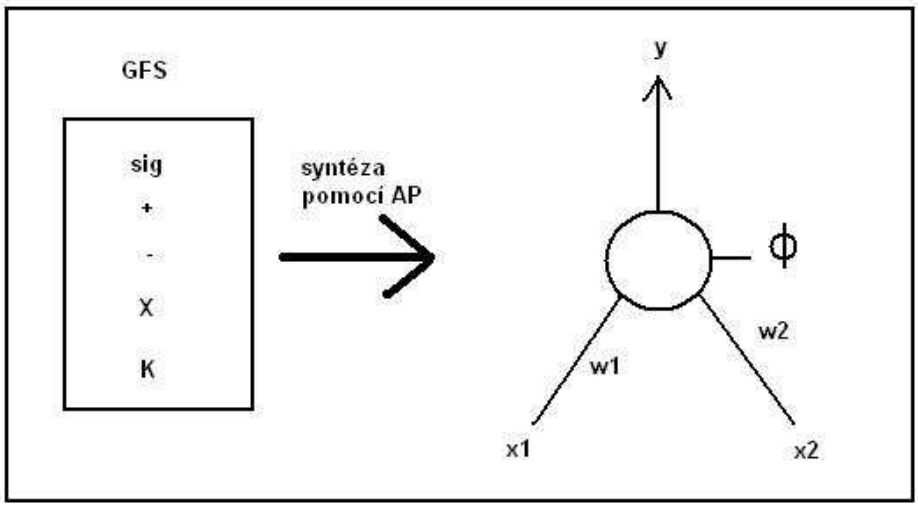
Každou konkrétně naučenou NS je možné chápat jako konkrétní matematickou funkci. NS totiž nedělá nic jiného, než že přiřazuje vstupním hodnotám hodnoty výstupní.

Potom každá ještě nenaučená NS odpovídá třídě funkcí f^*. Syntéza NS, tedy hledání struktury sítě je vlastně hledání vhodné f^* z množiny tříd funkcí F. Množina tříd funkcí F obsahuje všechny třídy, které vzniknou kombinacemi bázových funkcí. Naučení NS znamená nalezení vhodných konstant, které odlišují funkce f z třídy f^*. Výsledná naučená NS se potom rovná funkci f.

2.4.1 GFS u syntézy neuronové sítě

Pro syntézy neuronové sítě pomocí AP je důležité zvolit vhodnou množinu bázových funkcí. Základním stavebním kamenem neuronové sítě je neuron. Jako přenosová funkce se v neuronu často používá sigmoida odvozená od chování skutečných neuronů v mozku. GFS by tedy měla obsahovat sigmoidu. Další součástí neuronové sítě jsou vstupní nezávisle proměnné X, které GFS také musí obsahovat a také konstanty K reprezentující prahy a váhy neuronu. Prahy a váhy je ale potřeba na neuron nějak navázat a k tomu bude v GFS sloužit funkce + a * .

Na základě této úvahy je možné sestavit základní a úplnou $GFS = \{sig, +, *, X, K\}$. GFS je úplná z toho hlediska, že je možné pomocí ní řešit syntézu NS.



Obrázek 4 Syntéza neuronu pomocí AP

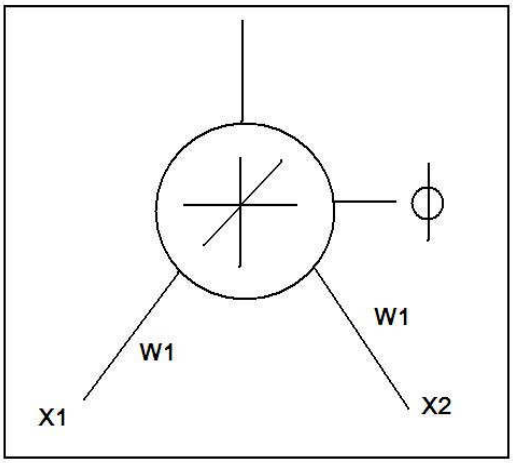
Taková definice GFS však může syntetizovat nejenom klasické NS, ale také pseudoneuronové nebo úplně neneuronové funkce. Tato otázka byla částečně řešena v [5]. Tímto způsobem se budou kombinovat vlastnosti NS spolu s matematickými transformacemi.

Tento způsob vytvoření GFS samozřejmě není jediný možný.

Vhodným uspořádáním prvků z $GFS_{LIN} = \{+, *, X, K\}$ vynikne [5] Obrázek 5:

$$K[1] * X[1] + X[2] + K[2]$$

Což je subfunkce, kterou lze považovat za lineární neuron, jehož přenosovou funkcí je přímka a její sklon a posunutí ovlivňují dále konstanty K.

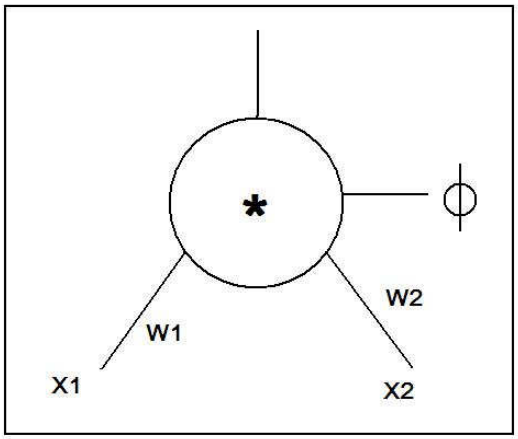


Obrázek 5 Lineární neuron

Další možností může být množina prvků $GFS_{NAS} = \{*, X, K\}$ pomocí ní může vzniknout subfunkce:

$$X[1] * X[2] * K[1]$$

Což lze považovat za násobící neuron [5], který z klasické teorie NS neznáme. Je to jakýsi pseudoneron Obrázek 6.

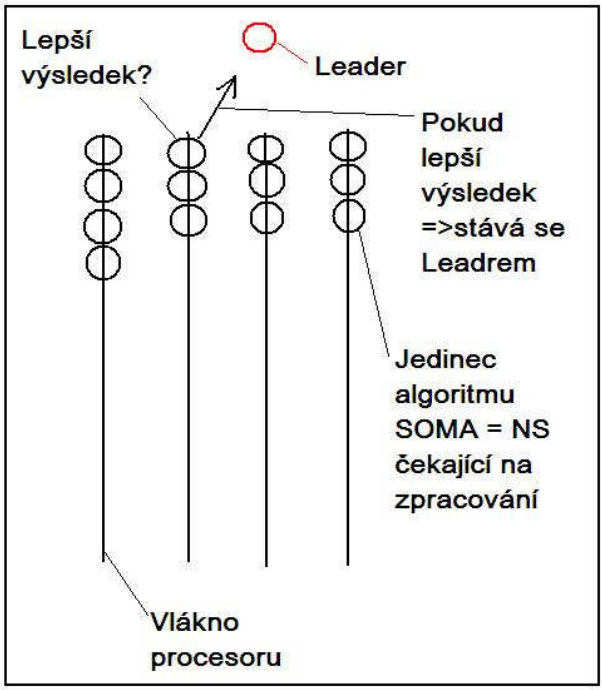


Obrázek 6 Násobící pseudoneuron

3 EXISTUJÍCÍ ASYNCHRONNÍ PROGRAMOVÁ IMPLEMENTACE SYNTÉZY NEURONOVÝCH SÍTÍ

Implementace Asynchonního algoritmu syntézy NS pomocí AP je naprogramována v jazyce C# na platformě .NET. Tento jazyk kombinuje vlastnosti známých objektově orientovaných jazyků a přidává v nim vlastnosti nové. Pro tuto implementaci je zejména důležitá podpora vícevláknového zpracování.

Algoritmus AP potřebuje ke svému běhu nějaký evoluční algoritmus. V této implementaci je právě použita asynchonní verze algoritmu SOMA. Asynchronnost algoritmu spočívá v tom, že není dokončováno celé migrační kolo a pak vybírán nový leader, ale nový leader je volen hned po ohodnocení každého jedince. Jedinci pro zpracování jsou rovnoměrně rozděleni mezi více procesů. Každý takový jedinec reprezentuje jednu strukturu NS. Taková NS je vždy naučena pomocí TD a dalšího algoritmu SOMA a ohodnocena účelovou funkcí. Hned po tomto ohodnocení je hodnota porovnána s aktuálním leaderem a pokud je lepší nahradí dosavadního leadera. Tuto situaci znázorňuje Obrázek 7.



Obrázek 7 Princip asynchronního výběru leadera v algoritmu SOMA

Běh algoritmu je řízen pomocí zastavovacího kritéria minimální globální chyby NS. Vnitřní běh SOMA, který učí NS je zastaven, pokud se v aktuálním migračním kole nepodaří snížit globální chybu NS nebo je dosaženo menší globální chyby, než je zastavovací kritérium.

II. PRAKTICKÁ ČÁST

4 NÁVRH EXPERIMENTŮ ZAMĚŘENÝCH NA SYNTÉZU NEURONOVÝCH SÍTÍ

Cílem experimentů s implementací algoritmu asynchronního analytického programování je nahlédnout na použitelnost této implementace v reálných úlohách. Vzhledem k tomu, že se jedná o stochastický algoritmus, poběží každá úloha minimálně desetkrát pro získání dat na statistické vyhodnocení.

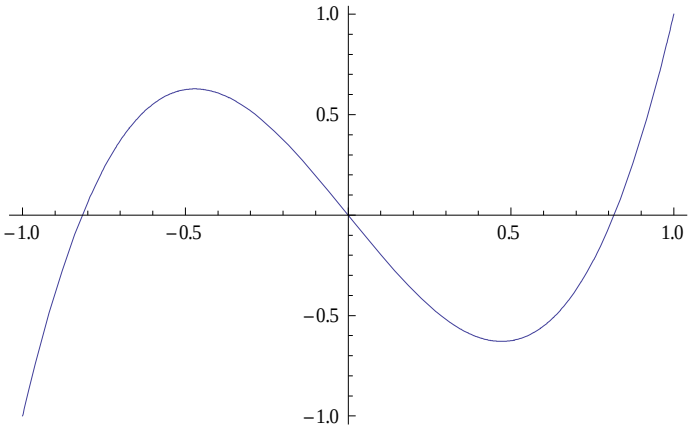
4.1 Závislosti na počtu tréninkových dat

Jak se ukazuje v experimentech na reálných úlohách z testovacího benchmarku Proben1 [8] je globální chyba naučené NS do značné míry závislá na počtu tréninkových dat. Zjednodušeně lze říci, že čím více tréninkových dat, tím menší je globální chyba sítě. A to jak na tréninkových, tak i validačních a testovacích datech. Je tedy žádoucí, aby syntéza NS probíhala s pokud možno dostatečně velkou množinou tréninkových dat. Vzhledem ke struktuře použitého algoritmu je pravděpodobné, že čas potřebný na syntézu NS poroste s množstvím tréninkových dat lineárně. Otázkou však je, jak se bude měnit struktura syntetizované sítě. Parametrem vystihujícím složitost struktury NS je hloubka sítě.

Dalším sledovaným parametrem je maximální počet tréninkových dat, který se dá ještě rozumě použít. Úlohy z benchmarku Proben1 obsahují přibližně 350 až 4000 tréninkových dat. Pro kvalitní naučení NS o více vstupech na reálnou úlohu by mohlo stačit 400 až 600 TD.

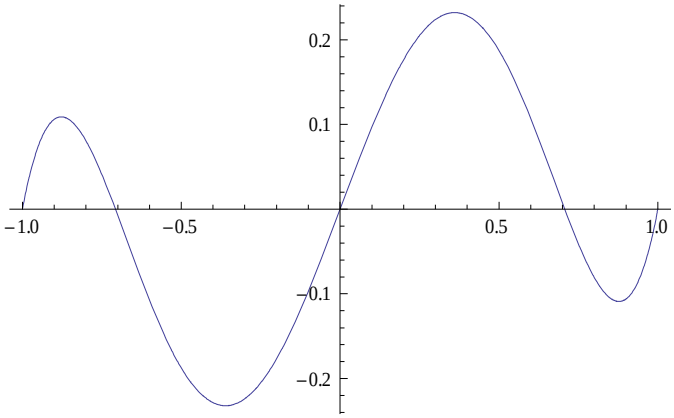
Pro experimenty s počtem TD byly zvoleny dvě funkce, z nichž jedna má složitější a jedna jednodušší průběh na sledovaném intervalu. Jedná se tedy o úlohu aproximace. Jsou to funkce o jedné proměnné, proto syntetizovaná NS bude mít jeden vstup a jeden výstup.

Jednodušší funkce má tvar $f_1 = 3 * a^3 - 2 * a$ a její průběh na intervalu $< -1,1 >$ je na Obrázek 8



Obrázek 8 Průběh funkce $f_1 = 3*a^3 - 2*a$ pro experimenty s různým počtem TD

Složitější funkce má tvar $f_2 = 2*a^5 - 3*a^3 + a$ a její průběh na intervalu $<-1,1>$ je vidět na Obrázek 9.



Obrázek 9 Průběh funkce $f_2 = 2*a^5 - 3*a^3 + a$ pro experimenty s různým počtem TD

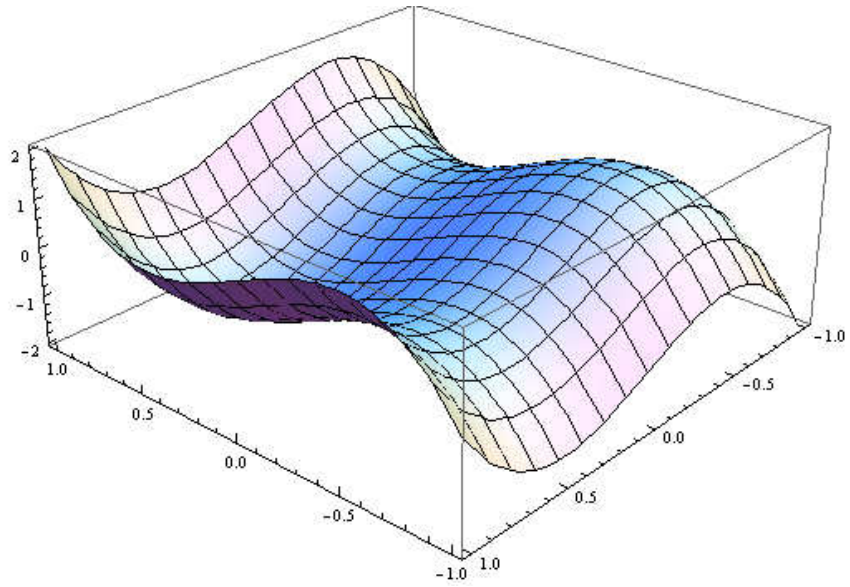
Cílem experimentů je otestovat 10 až 1000 TD. Přičemž se dá předpokládat, že 1000 TD již bude příliš pro syntézu NS v rozumném čase.

4.2 Závislosti na počtu vstupů syntetizované neuronové sítě

Pro reálné uplatnění je důležité kolik může mít syntetizovaná NS vstupů. Tato vlastnost je dána řešeným problémem. Je zřejmé, že složitost sítě a čas potřebný k syntéze a naučení NS bude s přibývajícím počtem vstupů růst. Cílem experimentů je tedy nahlédnout na tyto závislosti a určit kolik maximálně vstupů sítě je ještě rozumě použitelných pro reálnou úlohu a výpočet syntézy ve zvládnutelném čase. Aby však byly výsledky statisticky

porovnatelné z časového hlediska pro různé počty vstupů NS a tím i dimenze problému je zvolen pro všechny experimenty stejný počet tréninkových dat. Rovněž byla pro experimenty použita stejná konfigurace počítače na kterém běžely.

Jako základ byla zvolena funkce $f_1 = 3*a^3 - 2*a$, která je použita i pro experimenty se závislostmi na počtu TD. Pro rozšiřování do dalších dimenzí je jako závislost v další dimenzi zvolena opět stejná funkce. Tedy pro dva vstupy sítě (dimenze řešené úlohy) má funkce tvar $f_3 = 3*a^3 - 2*a + 3*b^3 - 2*b$ jejíž průběh na intervalu $a =< -1,1 >$ a $b =< -1,1 >$ je na Obrázek 10. Pro tři vstupy má potom funkce tvar $f_4 = 3*a^3 - 2*a + 3*b^3 - 2*b + 3*c^3 - 2*c$ atd.



Obrázek 10 Průběh funkce f_3 ve dvou rozměrech

Další sada experimentů s počtem vstupů NS má za úkol nahlédnout na vliv použité základní GFS. Je možné, že rozšíření GFS bude mít pozitivní vliv na rychlost syntézy a tedy i maximální možný počet vstupů NS, kterého je možné dosáhnout.

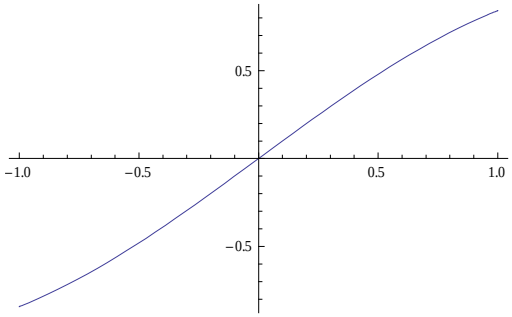
První sada experimentů je navržena s $GFS1 = \{\text{WeighInput, Neuron, Plus}\}$ a druhá sada s $GFS2 = \{\text{WeighInput, Neuron, Plus, Sin}\}$. Samozřejmě pokud má mít syntetizovaná NS více vstupů, obsahuje příslušná GFS také potřebný počet vstupních neuronů.

4.3 Závislosti na složitosti a typu řešené úlohy

Toto je asi nejhůře postižitelná otázka. Při řešení úlohy v praxi samozřejmě dopředu neznáme, jak průběh funkce aproximované neuronovou sítí vypadá. Známe jen typ úlohy, tedy zda se jedná o aproximaci, klasifikaci, predikci apod. Dále známe počet vstupů a výstupů NS. Vhledem k tomu, že klasifikační úloha má binární výstupy, bude pravděpodobně NS která ji řeší méně náročná na syntézu. Proto jsou prováděné experimenty zaměřené na aproximaci pro dosažení silnějších výsledků. Je navržena sada matematických funkcí, které jsou intuitivně seřazeny podle složitosti svého průběhu na testovacím intervalu. Nelze však dopředu říct, která funkce bude jak náročná z hlediska syntézy NS, která ji aproximuje.

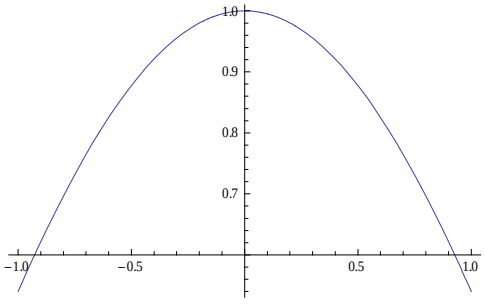
Jsou to tyto funkce jedné proměnné a jejich průběhy na intervalu $< -1,1 >$:

- $f_{01} = \sin(x)$



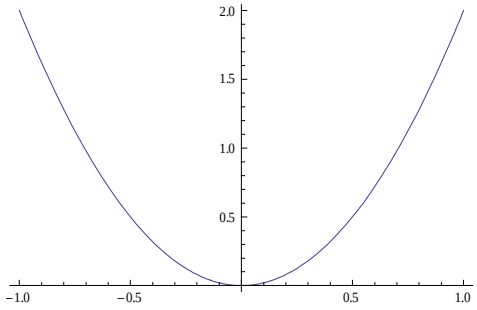
Obrázek 11 Funkce $f_{01} = \sin(x)$

- $f_{02} = \cos(x)$



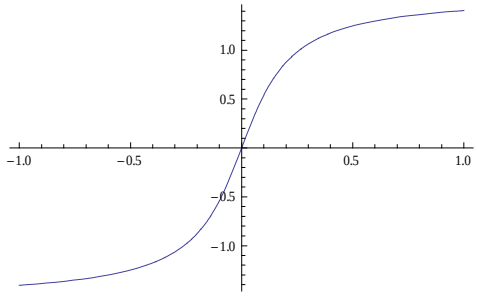
Obrázek 12 Funkce $f_{02} = \cos(x)$

- $f_{03} = 2x^2$



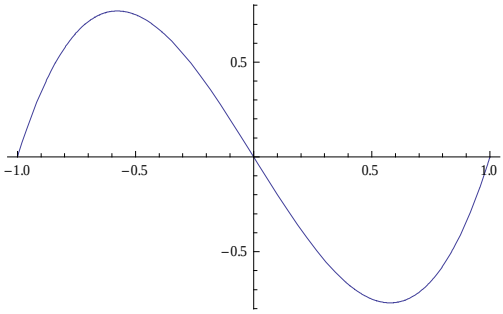
Obrázek 13 Funkce $f_{03} = 2x^2$

- $f_{04} = \arctan(6x)$



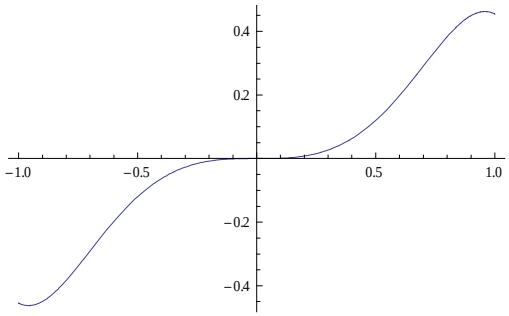
Obrázek 14 Funkce $f_{04} = \arctan(6x)$

- $f_{05} = 2x^3 - 2x$



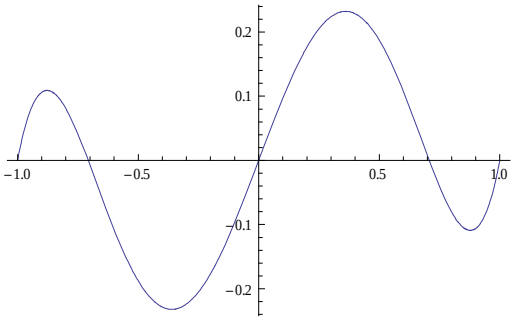
Obrázek 15 Funkce $f_{05} = 2x^3 - 2x$

- $f_{06} = \sin(x^2)\cos(x^2)x$



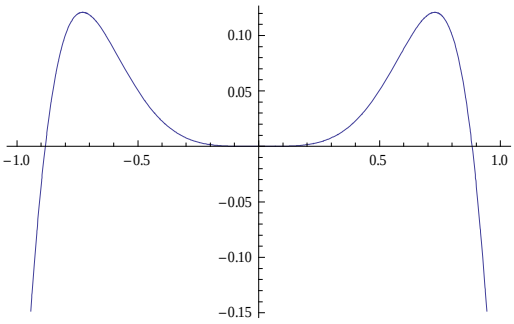
Obrázek 16 Funkce $f_{06} = \sin(x^2)\cos(x^2)x$

- $f_{07} = 2x^5 - 3x^3 + x$



Obrázek 17 Funkce $f_{07} = 2x^5 - 3x^3 + x$

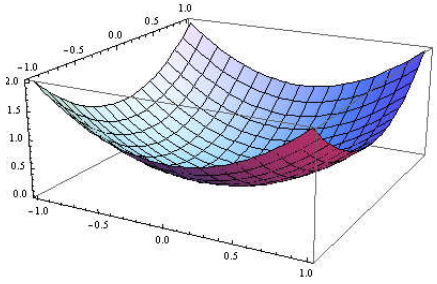
- $f_{08} = \sin(x^3)\cos(x^3 + x)x$



Obrázek 18 Funkce $f_{08} = \sin(x^3)\cos(x^3 + x)x$

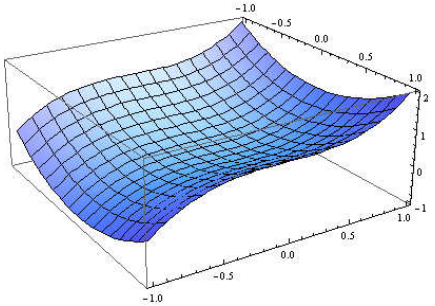
Další je sada funkcí dvou proměnných :

- $f_{001} = x^2 + y^2$



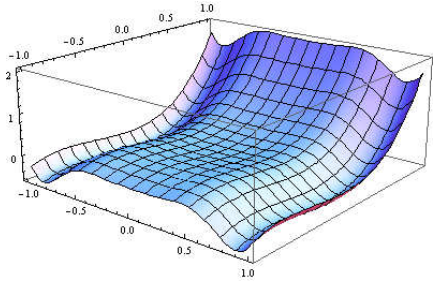
Obrázek 19 Funkce $f_{001} = x^2 + y^2$

- $f_{002} = x^2 + y^3$



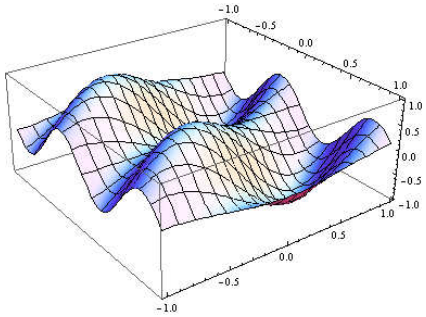
Obrázek 20 Funkce $f_{002} = x^2 + y^3$

- $f_{003} = x^3 \sin(6x) + y^3 + y^2$



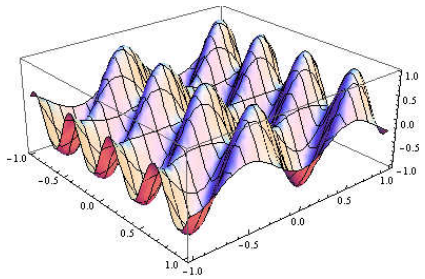
Obrázek 21 Funkce $f_{003} = x^3 \sin(6x) + y^3 + y^2$

- $f_{004} = \sin(3x)\cos(3x + 3y)$



Obrázek 22 Funkce $f_{004} = \sin(3x)\cos(3x+3y)$

- $f_{005} = \sin(6x)\cos(6x+6y)$



Obrázek 23 Funkce $f_{005} = \sin(6x)\cos(6x+6y)$

5 STATISTICKÉ VYHODNOCENÍ PROVEDENÝCH EXPERIMENTŮ

Statistické vyhodnocení experimentů pro nahlédnutí na chovaní algoritmu asymetrické implementace syntézy NS.

5.1 Závislosti na počtu tréninkových dat

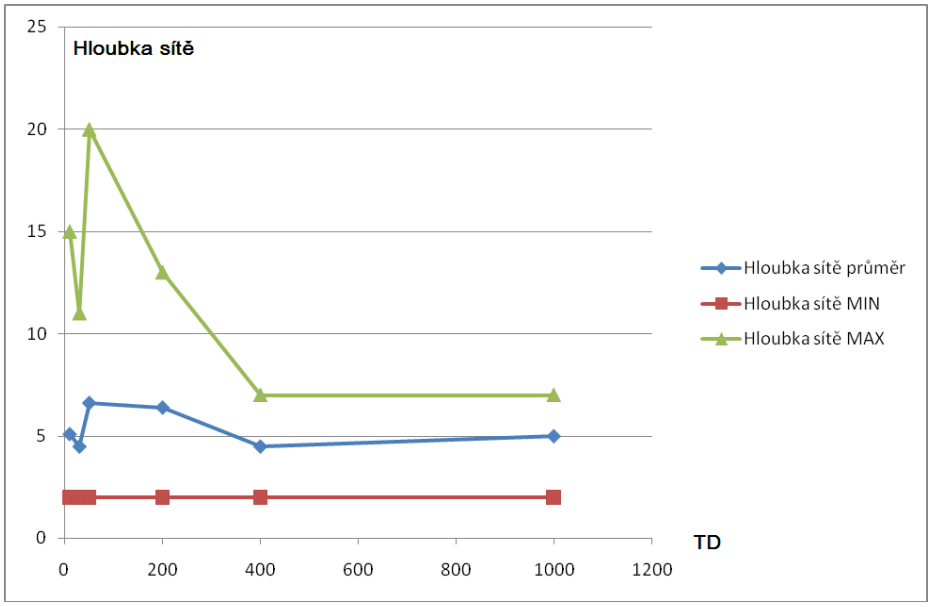
Z hlediska počtu TD běžely pokusy na dvoujádrovém procesoru Intel Core 2 Duo s frekvencí jader 1,8Ghz. Jako ukončovací kritérium je použita hodnota globální chyby 0,03. A vždy je použita GFS1.

5.1.1 Hloubka sítě v závislosti na počtu tréninkových dat

Jak již bylo řečeno, hloubka sítě je zajímavým parametrem určujícím složitost výsledné syntetizované NS. Ideálně NS řeší úlohu s dostatečnou přesností globální chyby a zároveň je žádoucí, aby měla co nejmenší složitost. V těchto experimentech je vždy aproximována stejná funkce, ale mění se počet TD. TD jsou vlastně funkční hodnotou aproximované funkce v různých bodech na daném intervalu. Struktura NS by tedy měla být ve všech případech přibližně stejná. Takových výsledků bylo dosaženo přibližně pro 200 až 400 TD. Menší počet TD znamená zřejmě nejistý výsledek, protože roste odchylka hloubky NS mezi jednotlivými pokusy. Velký počet TD znamená nejmenší odchylku, ale časová náročnost výpočtu je již neúnosná.

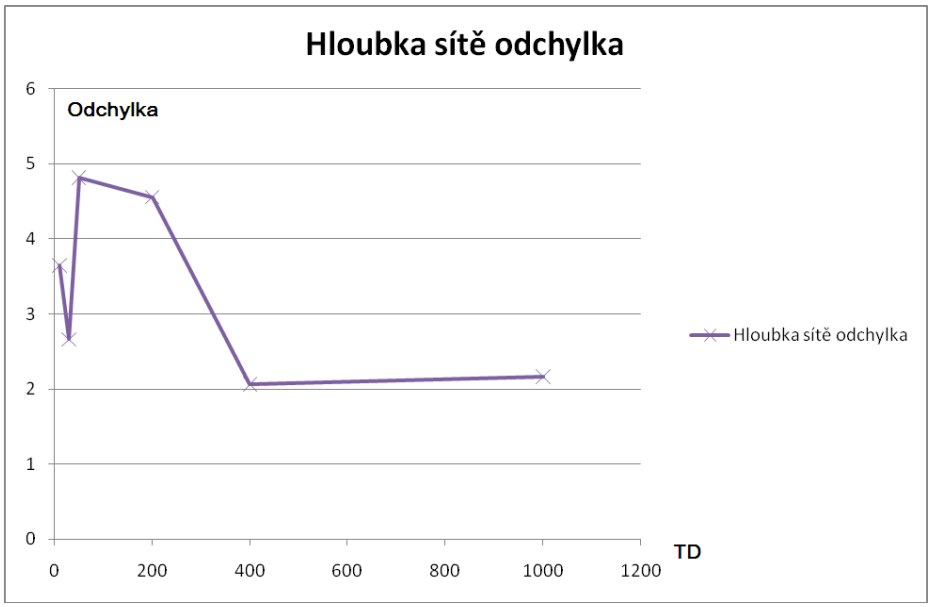
Tabulka 1 Hloubka sítě v závislosti na počtu tréninkových dat $f_1 = 3 * a^3 - 2 * a$

Počet tréninkových dat	10	30	50	200	400	1000
Hloubka sítě průměr	5,1	4,5	6,6	6,38	4,5	5
Hloubka sítě MIN	2	2	2	2	2	2
Hloubka sítě MAX	15	11	20	13	7	7
Hloubka sítě odchylka	3,65	2,66	4,82	4,56	2,06	2,16



Graf 1 Hloubka sítě v závislosti na počtu tréninkových dat.

Hodnoty MIN, MAX a průměr $f_1 = 3*a^3 - 2*a$



Graf 2 Hloubka sítě v závislosti na počtu tréninkových dat.

Hodnoty odchylky $f_1 = 3*a^3 - 2*a$

Z Tabulka 1 a příslušných grafů je vidět, že s přibývajícím množstvím TD roste pravděpodobnost, že bude syntetizována NS s nejjednodušší strukturou, protože odchylka hloubky sítě mezi jednotlivými pokusy klesá. Zároveň je ale vidět, že přes různý počet TD byla vždy v některém pokusu nalezena NS s minimální hloubkou 2, která aproximuje danou funkci se zadanou přesností.

Poznámka: *V průběhu pokusů se naskytla příležitost pro zajímavé srovnání výsledků pro 400 TD na dvoujádrovém procesoru Pentium D 3GHz a dvoujádrovém Core 2 Duo. Je trochu zarážející, že výsledky jsou poměrně odlišné a demonstruje je* Tabulka 2. *Nabízí se otázka, zda má použitý procesor takový vliv nebo je to dílem náhody.*

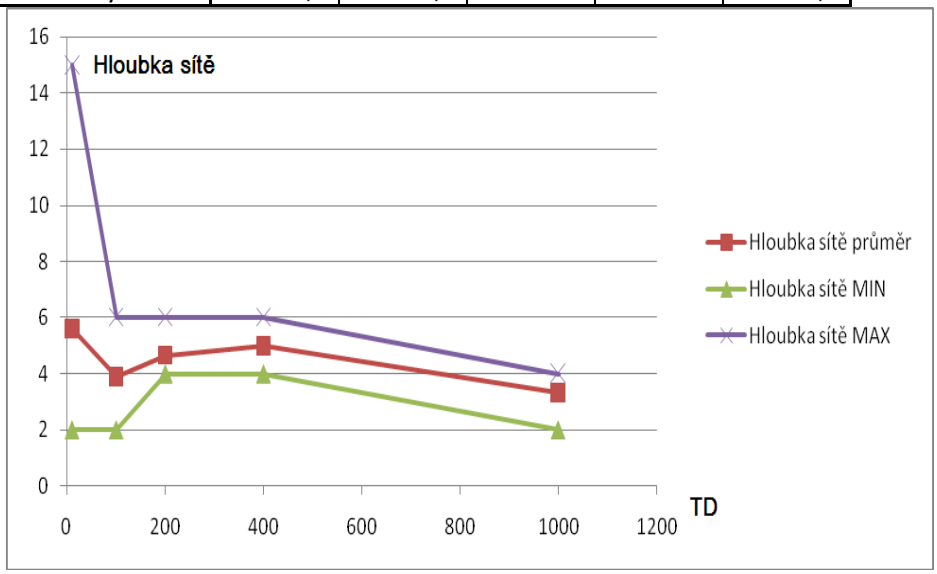
Tabulka 2 Srovnání pro dvoujadrové Pentium 4 a dvoujadrový Code 2 Duo na 400 TD

Procesor	Core 2 Duo	Pentium D
Hloubka sítě průměr	4,5	7,38
Hloubka sítě MIN	2	2
Hloubka sítě MAX	7	25
Hloubka sítě odchylka	2,06	7,81

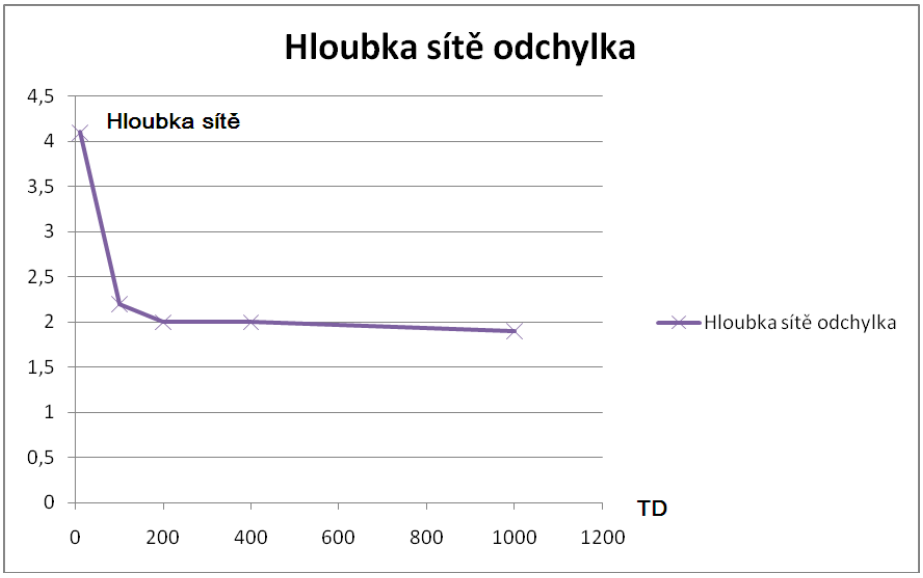
Hodnoty hloubky sítě pro f_2 v Tabulka 3 a k ní příslušné grafy potvrzují, že odchylka v hloubce sítě mezi jednotlivými pokusy skutečně klesá v závislosti na množství TD. Pro 200 a 400 TD však nebyla syntetizována NS s nejmenší možnou strukturou.

Tabulka 3 Hloubka sítě v závislosti na počtu tréninkových dat $f_2 = 2*a^5 - 3*a^3 + a$

Počet tréninkových dat	10	100	200	400	1000
Hloubka sítě průměr	5,6	3,9	4,66	5	3,33
Hloubka sítě MIN	2	2	4	4	2
Hloubka sítě MAX	15	6	6	6	4
Hloubka sítě odchylka	4,1	2,2	2	2	1,9



Graf 3 Hloubka sítě v závislosti na počtu tréninkových dat. Hodnoty MIN, MAX a průměr pro $f_2 = 2*a^5 - 3*a^3 + a$

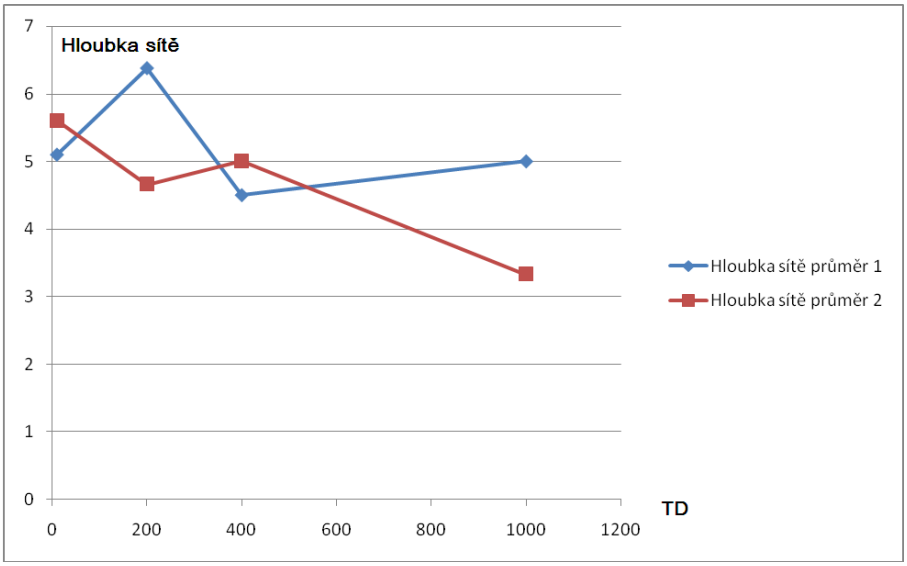


Graf 4 Hloubka sítě v závislosti na počtu tréninkových dat. Hodnoty odchylky hloubky sítě mezi jednotlivými pokusy pro $f_2 = 2*a^5 - 3*a^3 + a$

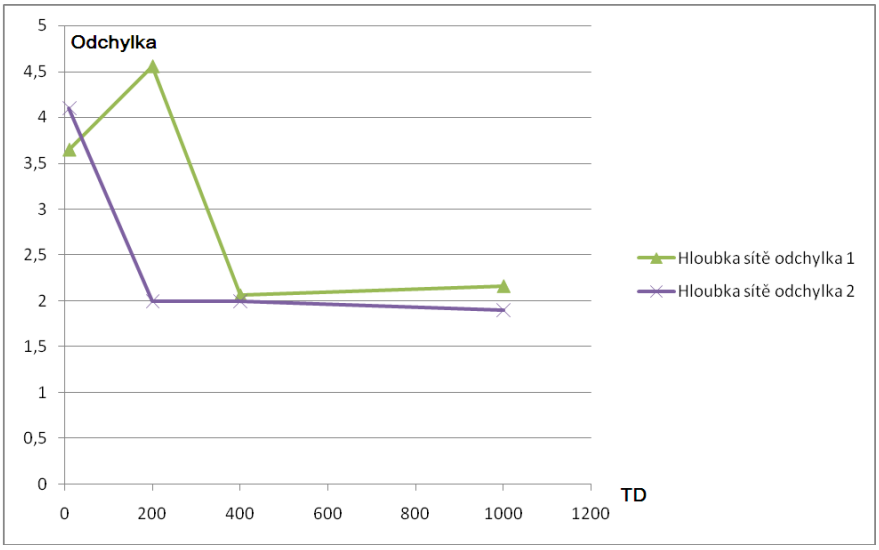
Tabulka 4 srovnává hodnoty průměrné hloubky sítě a odchylky pro funkce f_1 a f_2. Je vidět, že výsledky jsou podobné. A přestože f_2 má složitější průběh v některých případech je průměrná hodnota hloubky dokonce menší než je tomu u f_1.

Tabulka 4 Hloubka sítě v závislosti na počtu tréninkových dat. Srovnání průměrné hloubky a odchylky $f_1 = 3*a^3 - 2*a$ a $f_2 = 2*a^5 - 3*a^3 + a$.

Počet tréninkových dat	10	200	400	1000
Hloubka sítě průměr 1	5,1	6,38	4,5	5
Hloubka sítě průměr 2	5,6	4,66	5	3,33
Hloubka sítě odchylka 1	3,65	4,56	2,06	2,16
Hloubka sítě odchylka 2	4,1	2	2	1,9



Graf 5 Srovnání průměrné hloubky sítě pro $f_1 = 3*a^3 - 2*a$ a $f_2 = 2*a^5 - 3*a^3 + a$



Graf 6 Srovnání odchylky hloubky sítě pro $f_1 = 3*a^3 - 2*a$ a $f_2 = 2*a^5 - 3*a^3 + a$

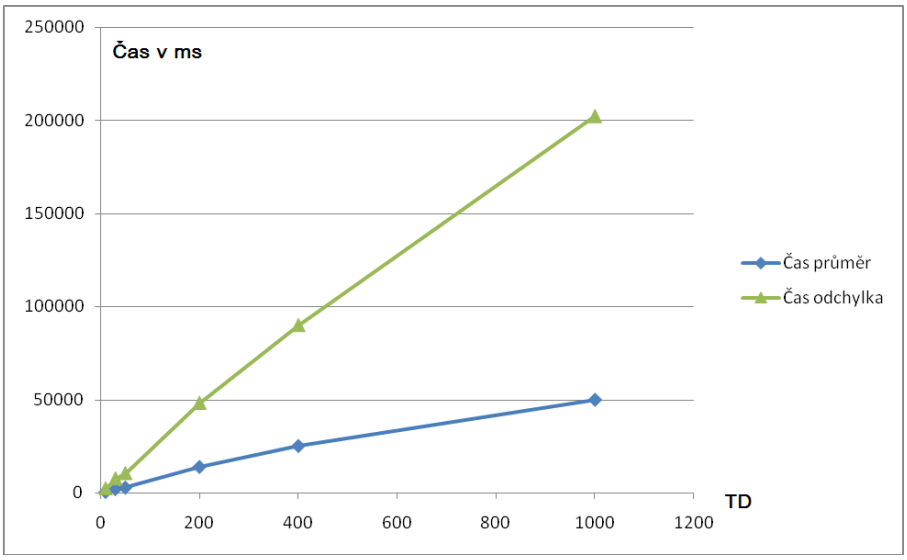
5.1.2 Čas výpočtu NS v závislosti na počtu tréninkových dat

Zde se potvrzuje domněnka, že čas potřebný k výpočtu poroste se zvyšujícím se počtem TD přibližně lineárně. A to jak průměrná hodnota, tak hodnota maximální. Toto se potvrdilo pro obě testované funkce. Zde je třeba si uvědomit, že časy jsou v tabulce uvedeny v milisekundách a týkají se naučení jedné struktury NS. Je to tedy čas výpočtu jednoho pokusu o syntézu výsledné NS. Největší hodnota funkce u f_1 je 3751578ms což

představuje více než hodinu. U funkce f_2 je to potom 2672406ms tedy přibližně tři čtvrtě hodiny.

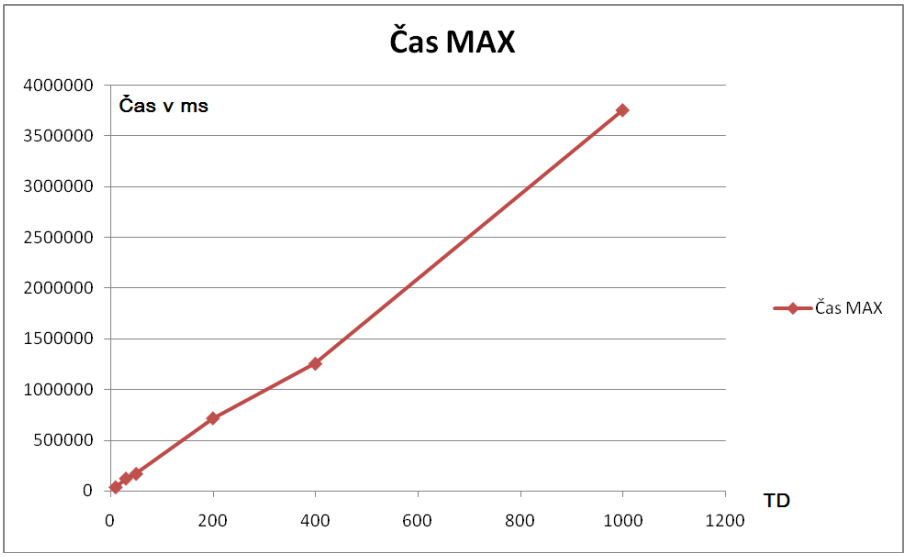
Tabulka 5 Čas výpočtu NS v závislosti na počtu tréninkových dat $f_1 = 3*a^3 - 2*a$

Počet tréninkových dat	10	30	50	200	400	1000
Čas průměr	649,14	2190,21	2930,38	14029,53	25226,36	49964,73
Čas MAX	35859,38	121375	167843,75	715546,88	1252750	3751578,13
Čas odchylka	2201,32	7509,41	10392,84	48162,32	89940,61	202207,58



Graf 7 Čas výpočtu NS v závislosti na počtu tréninkových dat.

Hodnoty průměr a odchylka pro $f_1 = 3*a^3 - 2*a$

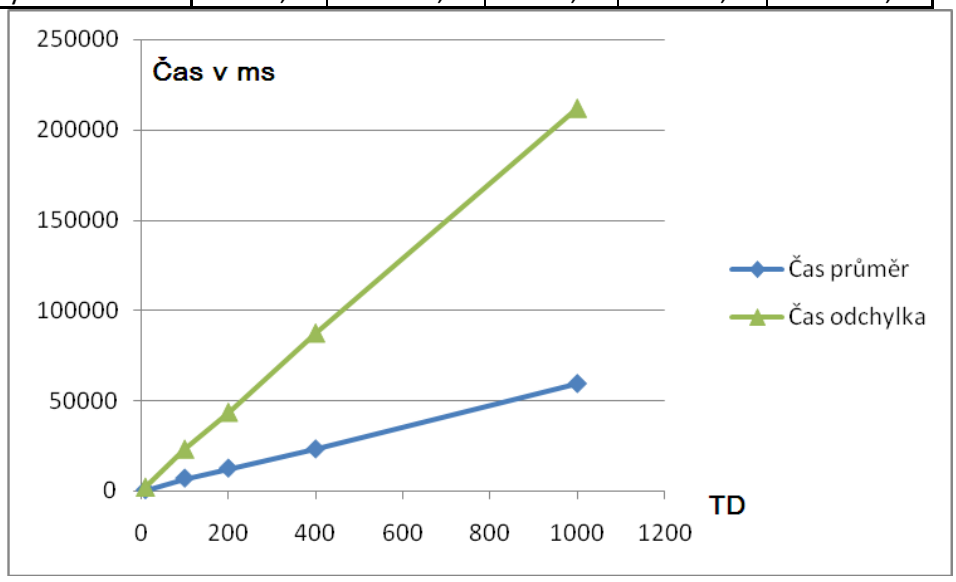


Graf 8 Čas výpočtu NS v závislosti na počtu tréninkových dat.

Vývoj maximální hodnoty pro $f_1 = 3*a^3 - 2*a$

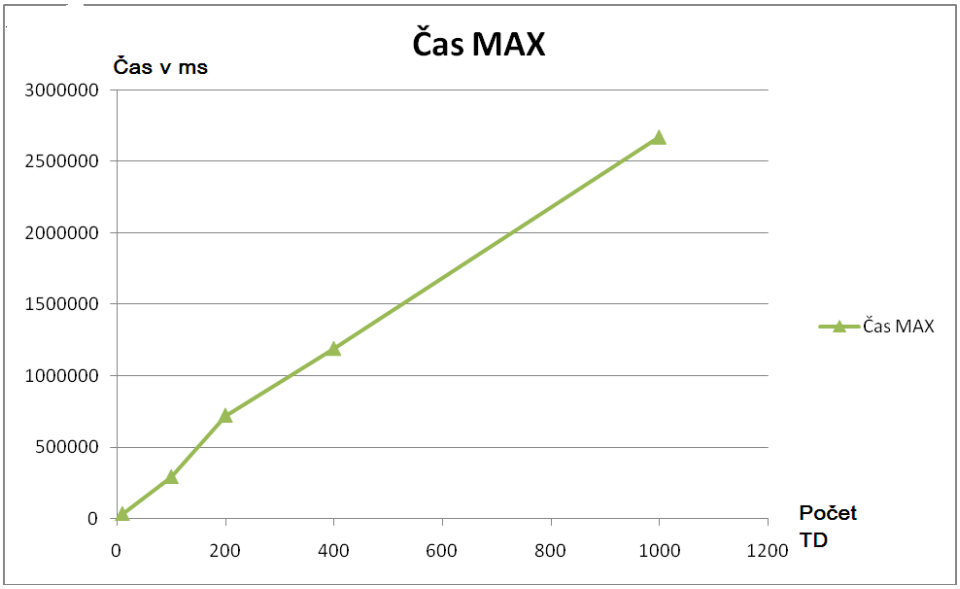
Tabulka 6 Čas výpočtu NS v závislosti na počtu tréninkových dat $f_2 = 2*a^5 - 3*a^3 + a$

Počet tréninkových dat	10	100	200	400	1000
Čas průměr	555,39	7023,86	12579,69	23230,4	59556,29
Čas MAX	36671,88	295484,38	724812,5	1192453,1	2672406,25
Čas odchylka	2124,91	23186,96	43520,46	87414,39	211811,27



Graf 9 Čas výpočtu NS v závislosti na počtu tréninkových dat.

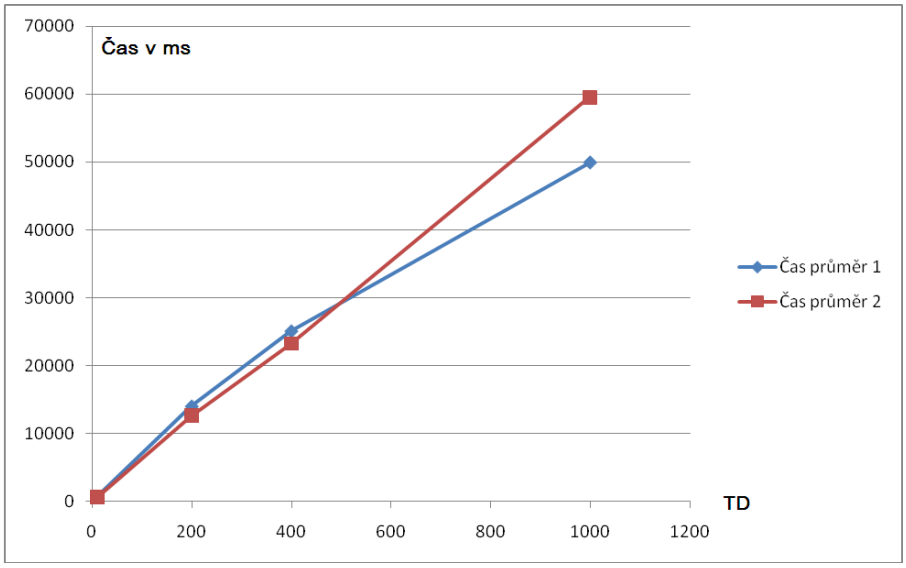
Průměrný čas a odchylka pro $f_2 = 2*a^5 - 3*a^3 + a$.



Graf 10 Čas výpočtu NS v závislosti na počtu tréninkových dat. Vývoj maximální hodnoty času pro $f_2 = 2*a^5 - 3*a^3 + a$.

Tabulka 7 Čas výpočtu NS v závislosti na počtu tréninkových dat. Srovnání průměrného času výpočtu NS pro $f_1 = 3*a^3 - 2*a$ a $f_2 = 2*a^5 - 3*a^3 + a$.

Počet tréninkových dat	10	200	400	1000
Čas průměr 1	649,14	14029,53	25103,33	49964,73
Čas průměr 2	555,39	12579,69	23230,4	59556,29

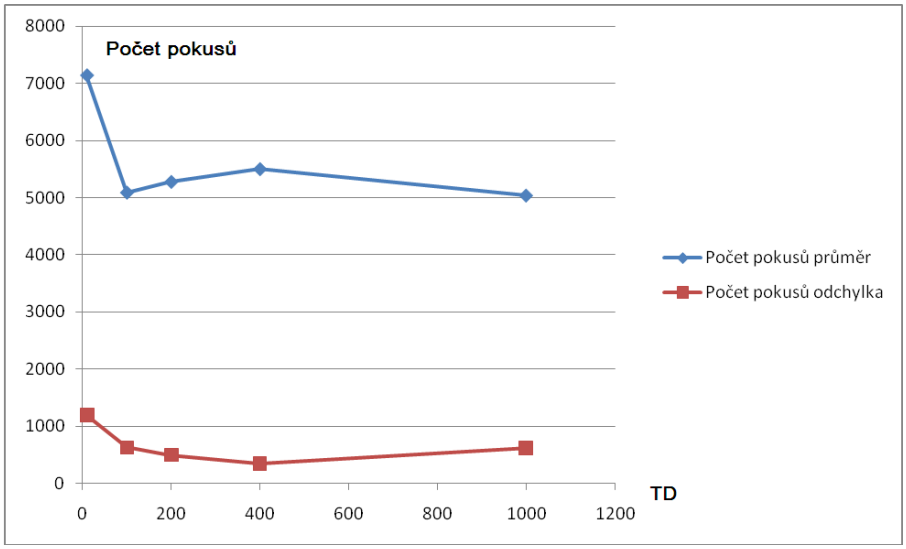


Graf 11 Čas výpočtu NS v závislosti na počtu tréninkových dat Srovnání průměrného času výpočtu NS pro $f_1 = 3*a^3 - 2*a$ a $f_2 = 2*a^5 - 3*a^3 + a$.

5.1.3 Počet pokusů v závislosti na počtu tréninkových dat

Tabulka 8 Počet pokusů v závislosti na počtu tréninkových dat pro $f_2 = 2*a^5 - 3*a^3 + a$

Počet tréninkových dat	10	100	200	400	1000
Počet pokusů průměr	7141,1	5089,7	5278,17	5501	5038,34
Počet pokusů odchylka	1198,978	628,41	498,27	345,67	618,6



Graf 12 Počet pokusů v závislosti na počtu tréninkových dat pro $f_2 = 2*a^5 - 3*a^3 + a$. Hodnoty průměru a odchylky.

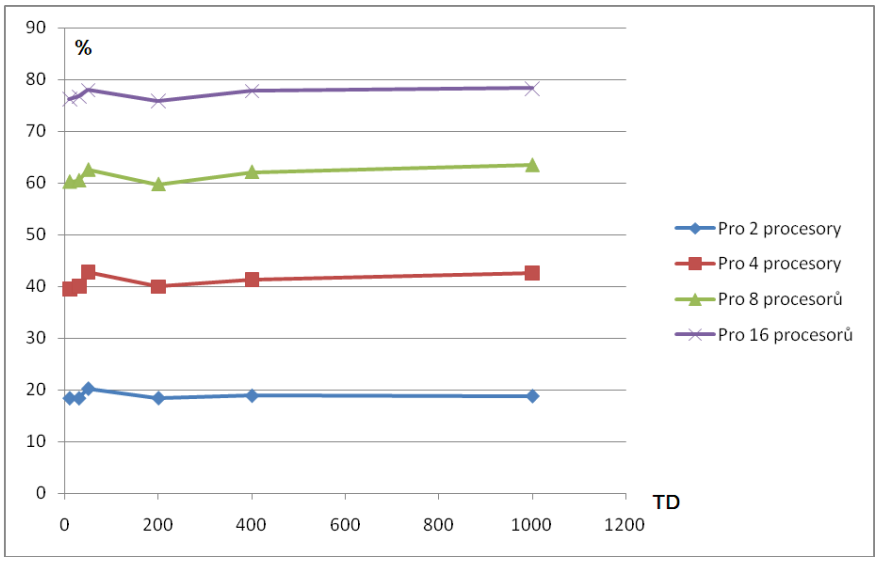
5.1.4 Ušetřený čas pro různý počet procesorů v závislosti na počtu TD

Tato statistika ukazuje, kolik je možné ušetřit času na různém počtu procesorů. Ušetřený čas jak pro f_1 tak pro f_2 je pro každý počet TD přibližně konstantní. Poměrně efektivní se zdá být výpočet na osmi procesorech, protože nárůst 20% oproti čtyřem procesorům je největší. Samozřejmě, že hodnoty jsou pouze hypotetické. Ve skutečnosti běžel výpočet na dvoujádrovém procesoru.

Tabulka 9 Ušetřený čas pro různý počet procesorů - různý počet TD pro $f_1 = 3*a^3 - 2*a$

Počet TD	10	30	50	200	400	1000
Pro 2 procesory	18,43%	18,41%	20,25%	18,45%	18,9%	18,83%
Pro 4 procesory	39,44%	40,01%	42,78%	39,93%	41,34%	42,58%
Pro 8 procesorů	60,22%	60,48%	62,56%	59,75%	62,12%	63,46%

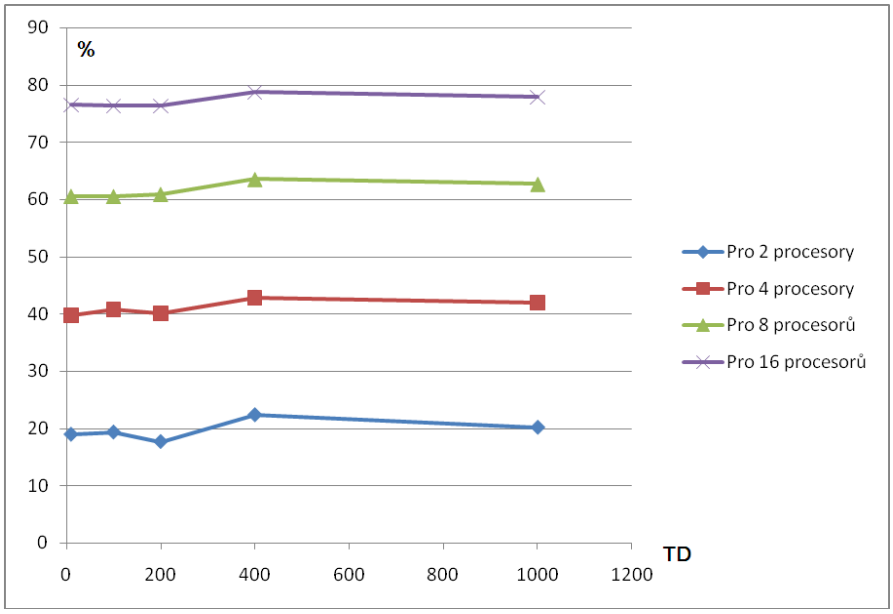
Pro 16 procesorů	76,26%	76,7%	78,02%	75,86%	77,84%	78,32%



Graf 13 Ušetřený čas pro různý počet procesorů - různý počet TD pro $f_1 = 3*a^3 - 2*a$

Tabulka 10 Ušetřený čas pro různý počet procesorů - různý počet TD pro $f_2 = 2*a^5 - 3*a^3 + a$

Počet TD	10	100	200	400	1000
Pro 2 procesory	18,92	19,31	17,65	22,34	20,14
Pro 4 procesory	39,78	40,74	40,09	42,83	41,96
Pro 8 procesorů	60,6	60,56	60,93	63,53	62,69
Pro 16 procesorů	76,45	76,35	76,36	78,7	77,85



Graf 14 Ušetřený čas pro různý počet procesorů - různý počet TD pro $f_2 = 2*a^5 - 3*a^3 + a$

5.2 Závislosti na počtu vstupů syntetizované neuronové sítě

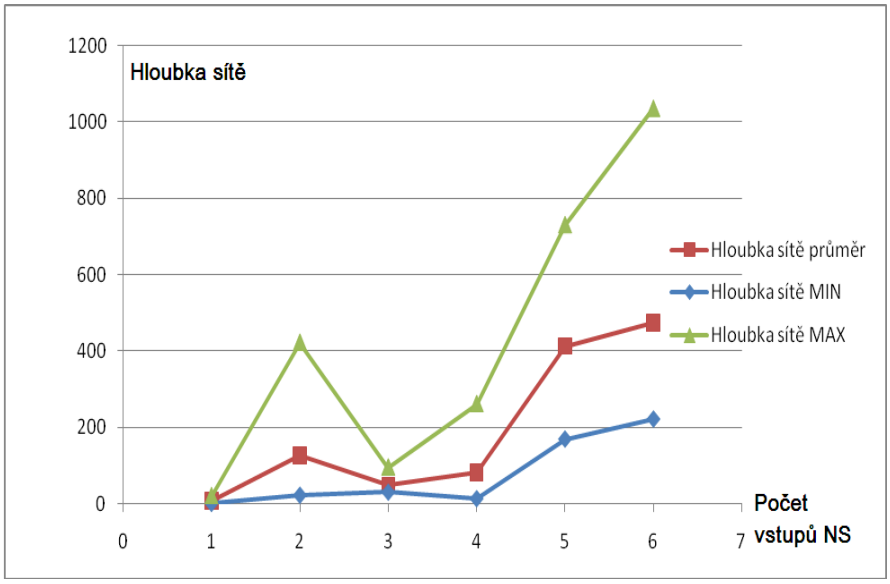
Výsledky těchto experimentů byli získány na čtyřjádrovém procesoru Xeon, nejsou tedy z hlediska dosažených časů přímo porovnatelné s předchozími, protože se jedná o zhruba trojnásobný výkon.

5.2.1 Hloubka sítě v závislosti na počtu vstupů NS

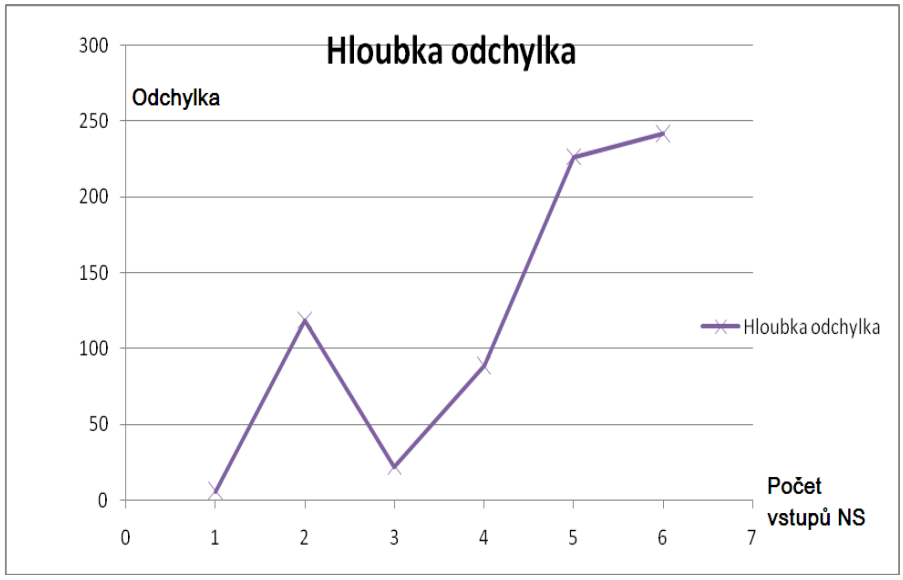
S přibývajícím počtem vstupů NS samozřejmě prudce roste hloubka a tím také složitost struktury výsledné syntetizované NS. Z hodnot odchylky hloubky NS mezi jednotlivými pokusy experimentů je vidět, že rozdíly mezi jednotlivými výslednými strukturami jsou značné. Proto je potřeba pro nalezení dobrého výsledku z hlediska jednoduché struktury NS syntézu opakovat.

Tabulka 11 Hloubka sítě v závislosti na počtu vstupů NS – GFS1

Počet vstupů	1	2	3	4	5	6
Hloubka sítě průměr	7,92	127,5	48,9	82,9	413,25	475,13
Hloubka sítě MIN	2	23	31	14	169	221
Hloubka sítě MAX	21	423	96	262	731	1036
Hloubka odchylka	5,94	118,48	22,29	88,56	226,11	241,5



Graf 15 Hloubka sítě v závislosti na počtu vstupů NS – hodnoty pro MIN, MAX a průměr – GFS1

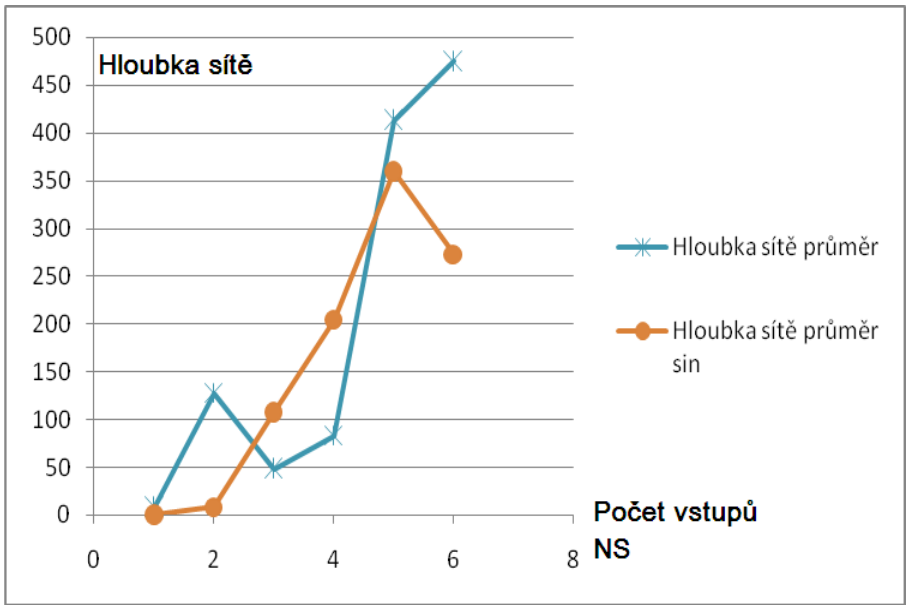


Graf 16 Hloubka sítě v závislosti na počtu vstupů NS – hodnoty odchylky – GFS1

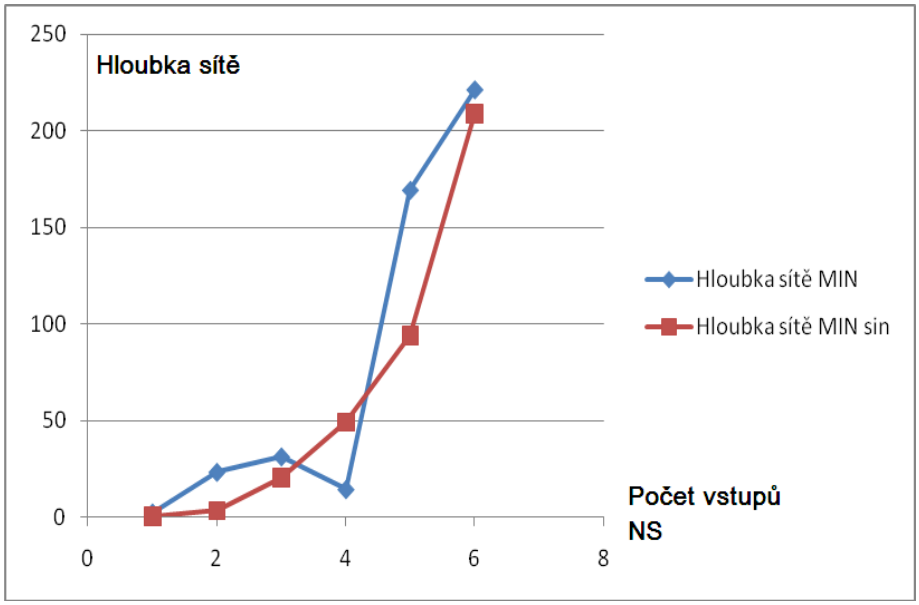
V Tabulka 12 a na ní navazujících grafech je zajímavé srovnání výsledků hloubky NS pro GFS1 a GFS2. Z hlediska průměrné hodnoty se ukazuje jako lepší použití GFS2. A to hlavně pro menší dosažené hodnoty hloubky NS s více vstupy. Velmi dobře se pro GFS2 vyvíjí také hodnota odchylky, pokud má NS více vstupů.

Tabulka 12 Hloubka sítě pro různý počet vstupů. Porovnání GFS1 a GFS2

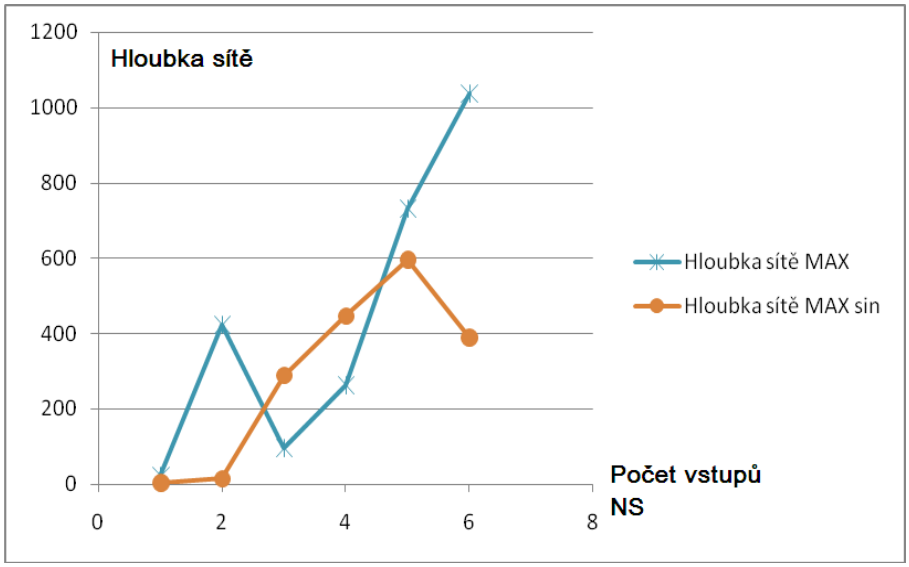
Počet vstupů	1	2	3	4	5	6
Hloubka sítě průměr GFS1	7,92	127,5	48,9	82,9	413,25	475,13
Hloubka sítě průměr GFS2	0,4	8,1	108	204,67	360,14	273
Hloubka sítě MIN GFS1	2	23	31	14	169	221
Hloubka sítě MIN GFS2	0	3	20	49	94	209
Hloubka sítě MAX GFS1	21	423	96	262	731	1036
Hloubka sítě MAX GFS2	2	13	288	446	596	388
Hloubka odchylka GFS1	5,94	118,47	22,29	88,56	226,11	241,5
Hloubka odchylka GFS2	0,66	2,85	87,05	134,44	147,08	70,95



Graf 17 Hloubka sítě pro různý počet vstupů. Porovnání GFS1 a GFS2 z hlediska průměrné hodnoty.

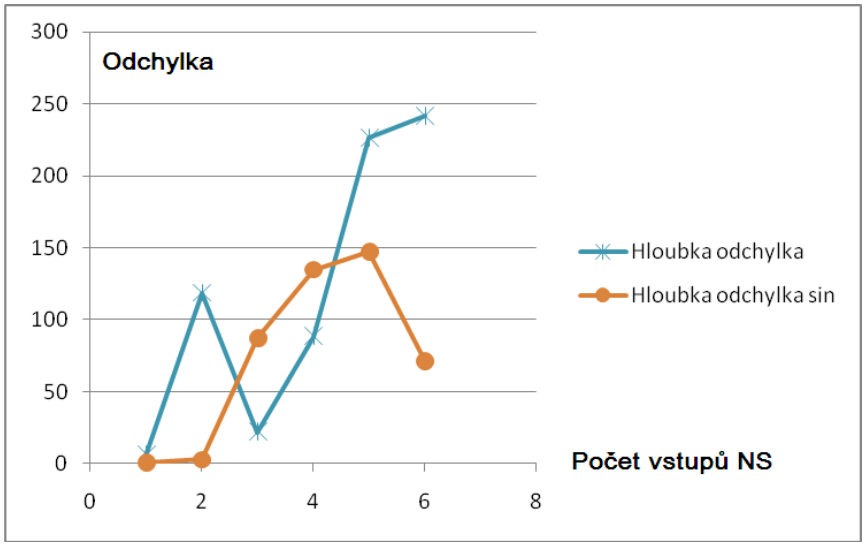


Graf 18 Hloubka sítě pro různý počet vstupů. Porovnání hodnot MIN pro GFS1 a GFS2.



Graf 19 Hloubka sítě pro různý počet vstupů. Porovnání hodnot MIN pro GFS1 a GFS2.

Graf 20 ukazuje jak se vývoj odchylky pro GFS2 odchýlil od hodnot dosažených s GFS1. To by mohlo znamenat potřebu méně opakování syntézy pro nalezení NS úspěšně řešící daný problém s více vstupy a minimální strukturou.

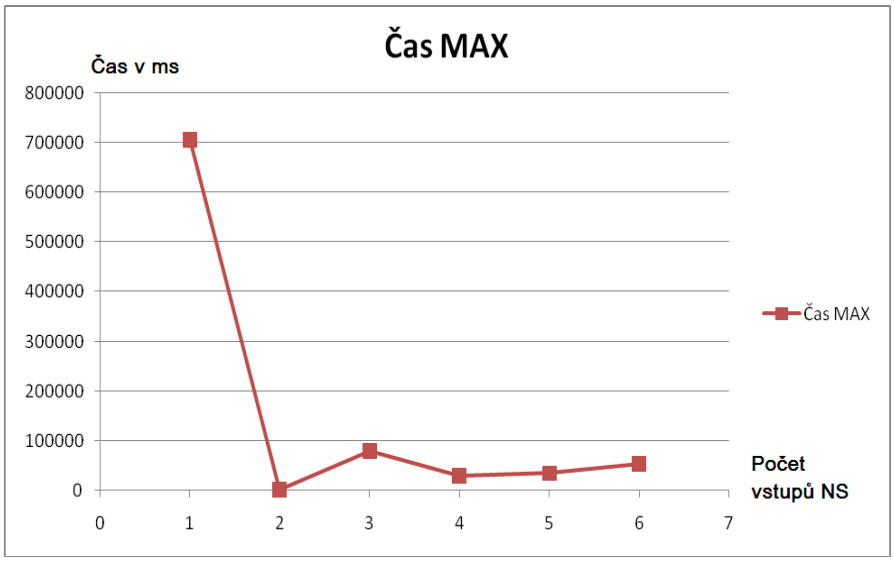


Graf 20 Hloubka sítě pro různý počet vstupů. Srovnání hodnot odchylky pro GFS1 a GFS2.

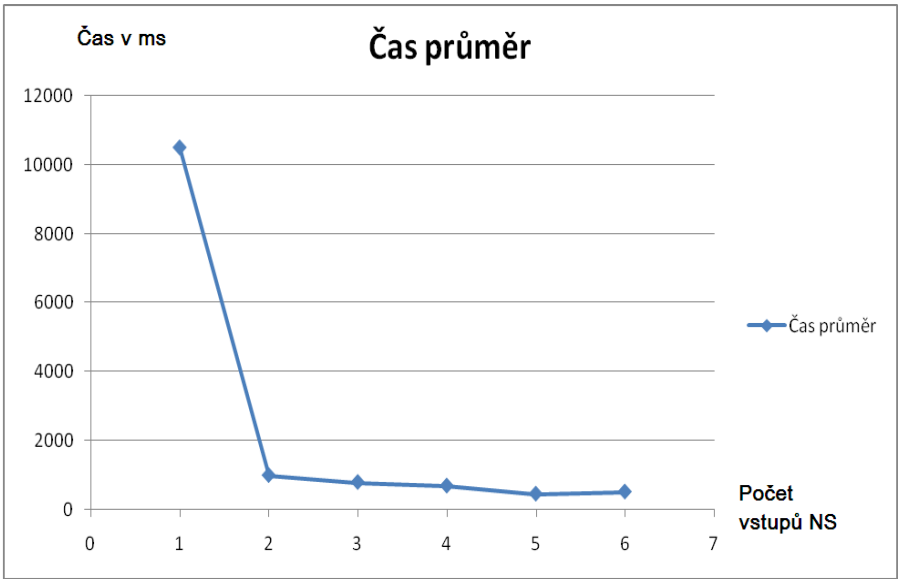
5.2.2 Čas výpočtu NS v závislosti na počtu vstupů NS

Tabulka 13 Čas výpočtu NS v závislosti na počtu vstupů NS s GFS1.

Počet vstupů NS	1	2	3	4	5	6
Čas průměr	10498,34	992,16	791,84	686,68	452,07	515,52
Čas MAX	706234,38	2132,12	80687,5	29890,625	35468,75	54125



Graf 21 Čas výpočtu NS v závislosti na počtu vstupů NS s GFS1 – maximální hodnoty



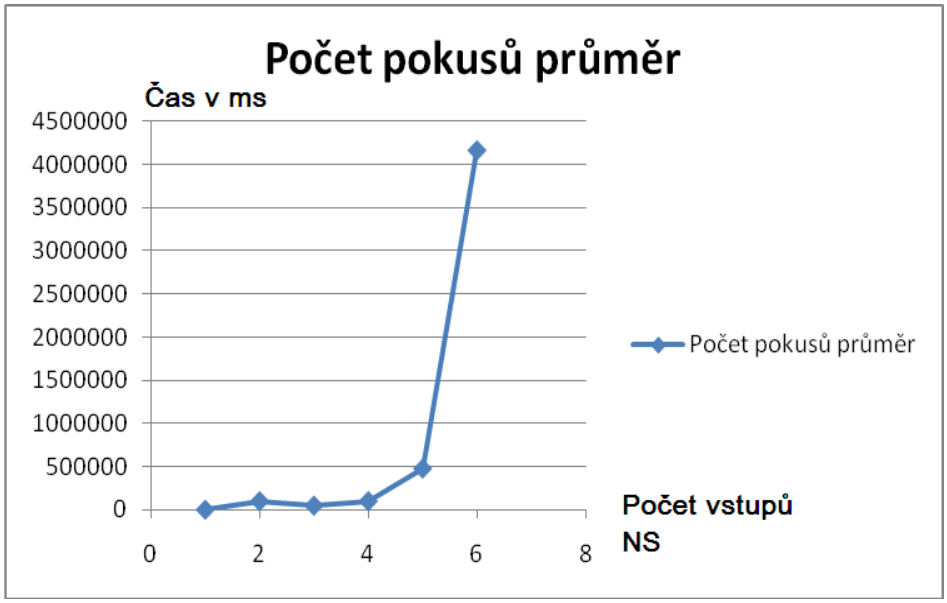
Graf 22 Čas výpočtu NS v závislosti na počtu vstupů NS s GFS1 – průměrné hodnoty

5.2.3 Počet pokusů výpočtu NS v závislosti na počtu vstupů NS

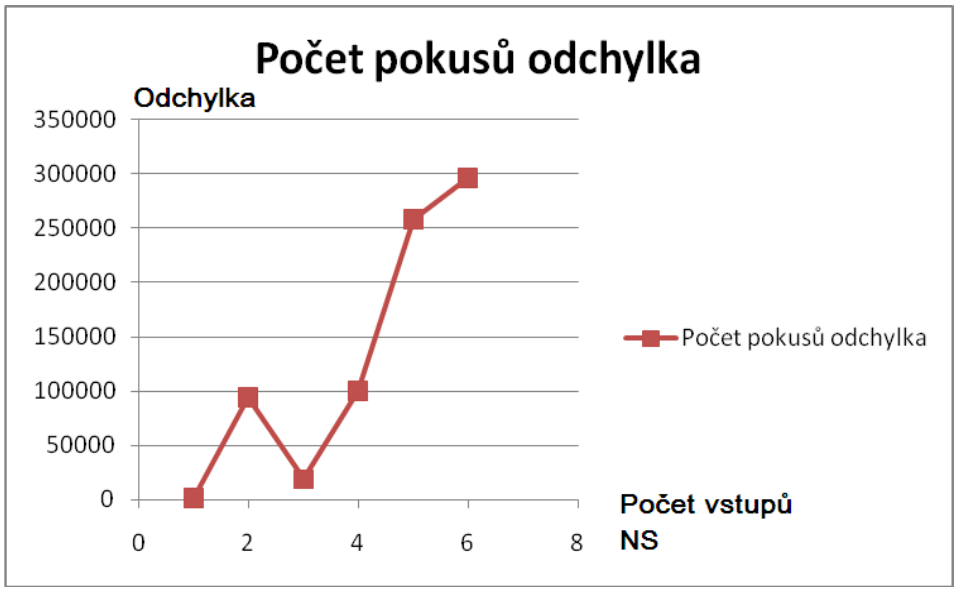
Tabulka 14 a z ní vycházející grafy ukazuje, že pro 6 vstupů s použitím GFS1 nastává dramatický nárůst počtu pokusů potřebných pro syntézu NS. Z tohoto hlediska další navýšení počtu vstupů již prakticky není možné.

Tabulka 14 Počet pokusů výpočtu NS v závislosti na počtu vstupů NS s GFS1.

Počet vstupů	1	2	3	4	5	6
Počet pokusů průměr	6964,25	105316,7	54171,2	105672,5	482349,25	4163710
Počet pokusů odchylka	1198,44	93785,35	18597,67	100176,58	257926,73	295870,63



Graf 23 Počet pokusů výpočtu NS v závislosti na počtu vstupů NS s GFS1 – hodnoty průměru

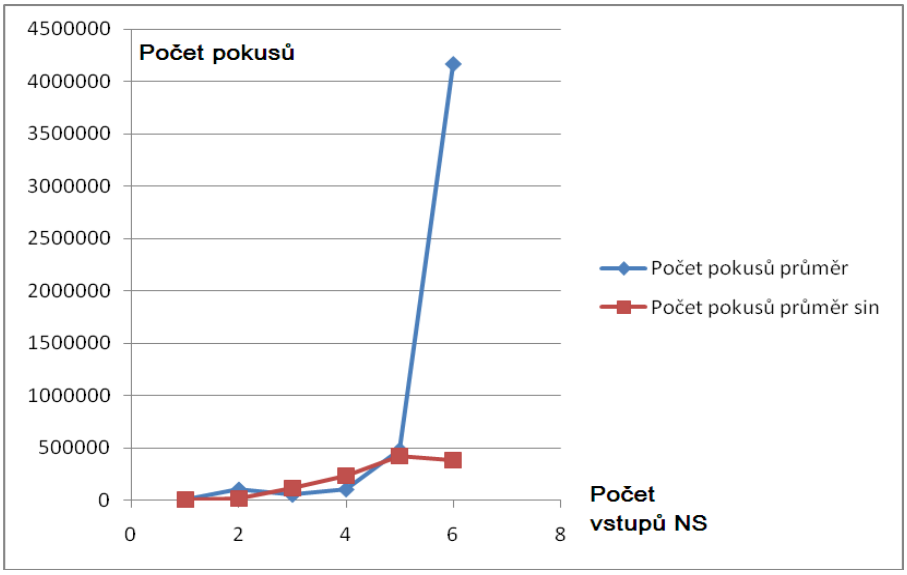


Graf 24 Počet pokusů výpočtu NS v závislosti na počtu vstupů NS s GFS1 – hodnoty odchylky

V Tabulka 15 a navazujícím grafu je srovnání počtu pokusů pro GFS1 a GFS2. Z grafu je patrné, jak velký je rozdíl v nárůstu počtu pokusů při použití GFS2 oproti GFS1. Při použití GFS2 by nejspíš bylo možné počet vstupů ještě navýšit bez dramatického nárůstu počtu pokusů.

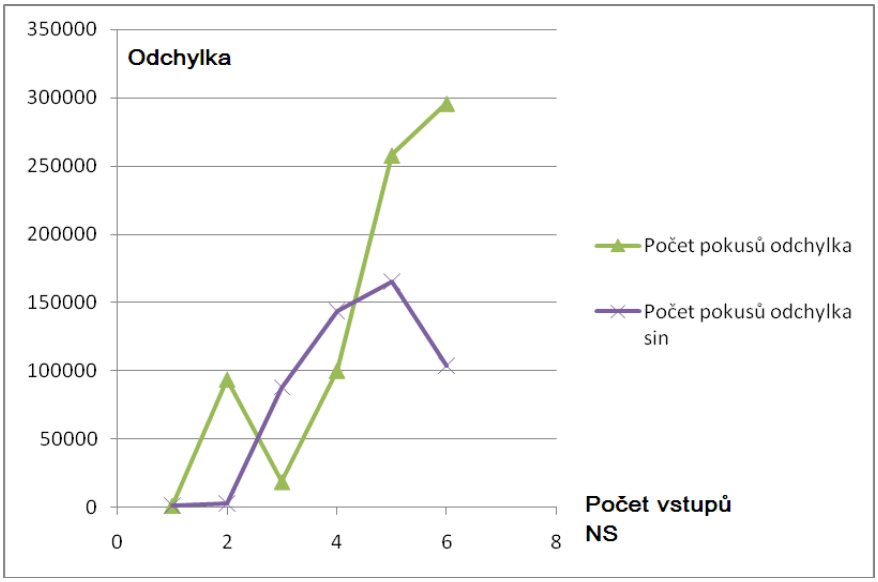
Tabulka 15 Počtu pokusů pro různý počet vstupů. Srovnání GFS1 a GFS2.

Počet vstupů	1	2	3	4	5	6
Počet pokusů průměr	6964,25	105316,7	54171,2	105672,5	482349,25	4163710
Počet pokusů průměr sin	3807,2	14404,2	116786,22	233755,66	423022,85	380707
Počet pokusů odchylka	1198,44	93785,35	18597,67	100176,58	257926,73	295870,63
Počet pokusů odchylka sin	1180,26	2866,06	87853,53	143739,97	165356,55	103870,97



Graf 25 Počtu pokusů pro různý počet vstupů. Srovnání GFS1 a GFS2 – průměrné hodnoty

Jak ukazuje Graf 26 tak také odchylka v počtech pokusů mezi jednotlivými opakováními syntézy dosažená s GFS2 neroste s počtem vstupů tak jako v případě použití GFS1.



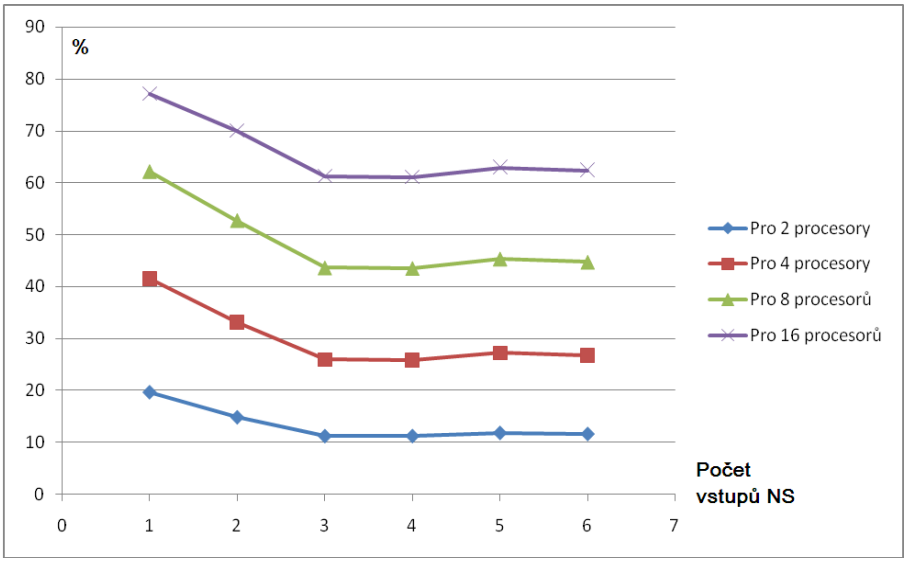
Graf 26 Počtu pokusů pro různý počet vstupů. Srovnání GFS1 a GFS2 – hodnoty odchylky.

5.2.4 Ušetřený čas pro různý počet procesorů v závislosti na počtu vstupů NS

Hodnoty ušetřeného času vzhledem k počtu použitých procesorů potřebného pro syntézu NS v závislosti na počtu vstupů NS ukazuje Tabulka 16 a Graf 27. Jsou to hodnoty dosažené s použitím GFS1. Je vidět, že efektivita více procesorů s přibývajícím množstvím vstupů mírně klesá. S tím, že je zde patrný mírný zlom na hodnotě třech vstupů, pak již je zdá se využití více procesorů konstantní.

Tabulka 16 Ušetřený čas pro různý počet procesorů - různý počet vstupů. Výsledky dosažené s GFS1.

Počet vstupů	1	2	3	4	5	6
Pro 2 procesory	19,63%	14,83%	11,23%	11,24%	11,84%	11,64%
Pro 4 procesory	41,58%	33,09%	25,9%	25,76%	27,19%	26,73%
Pro 8 procesorů	62,09%	52,7%	43,62%	43,52%	45,33%	44,72%
Pro 16 procesorů	77,14%	70%	61,25%	61,09%	62,97%	62,44%

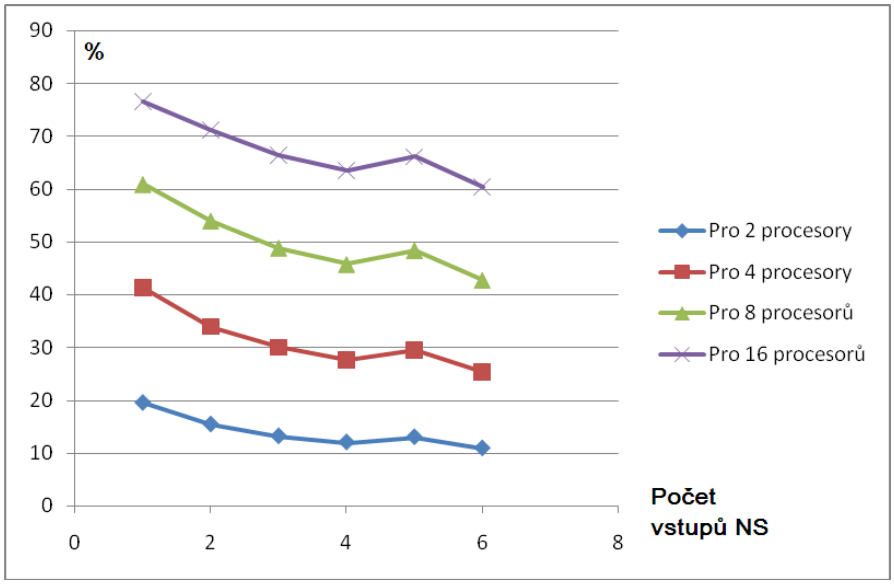


Graf 27 Ušetřený čas pro různý počet procesorů - různý počet vstupů. Výsledky dosažené s GFS1

Při použití GFS2 je situace trochu odlišná. Jak ukazuje Tabulka 17 a Graf 27, tak zde není patrný žádný zlom, ale hodnoty ušetřeného času pro více procesorů mírně klesají pro všechny hodnoty počtu vstupů.

Tabulka 17 Ušetřený čas pro různý počet procesorů - různý počet vstupů. Výsledky dosažené s GFS2.

Počet vstupů	1	2	3	4	5	6
Pro 2 procesory	19,55%	15,46%	13,22%	12,05%	13,04%	10,92%
Pro 4 procesory	41,37%	33,92%	30,01%	27,58%	29,49%	25,26%
Pro 8 procesorů	60,94%	54,02%	48,86%	45,79%	48,42%	42,78%
Pro 16 procesorů	76,54%	71,16%	66,35%	63,42%	66,09%	60,32%



Graf 28 Ušetřený čas pro různý počet procesorů - různý počet vstupů. Výsledky dosažené s GFS2.

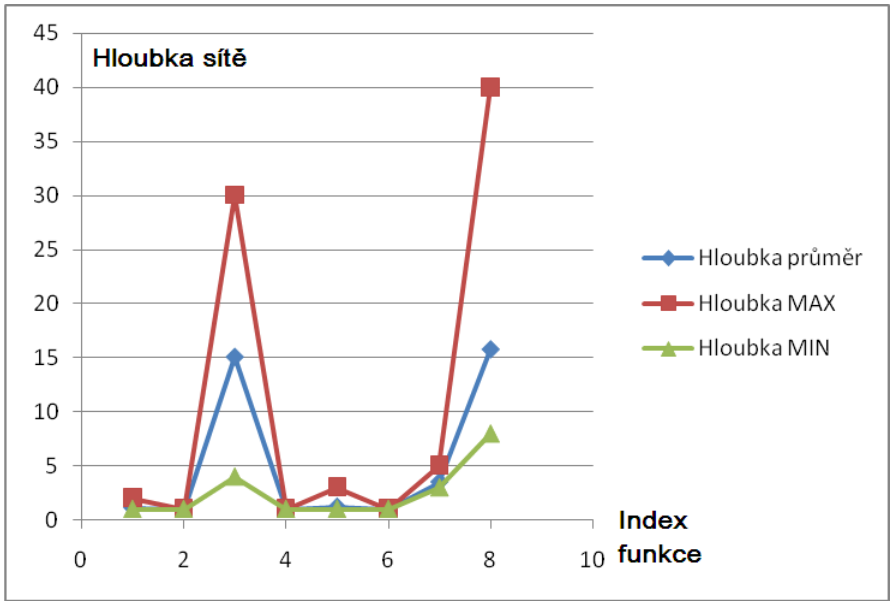
5.3 Závislosti na průběhu aproximované funkce

Experimenty probíhaly pro sadu funkcí o jedné proměnné a pro sadu funkcí o dvou proměnných. Funkce v sadě jsou v pořadí, jak se intuitivně jevily podle složitosti svého průběhu na zkoumaném intervalu. Některé funkce se však ukázaly složitější na syntézu, něž by se dalo podle jejich průběhu očekávat. Všechny experimenty v této kapitole probíhaly s GFS1 a nastaveným zastavovacím kritériem globální chyby na 0,03.

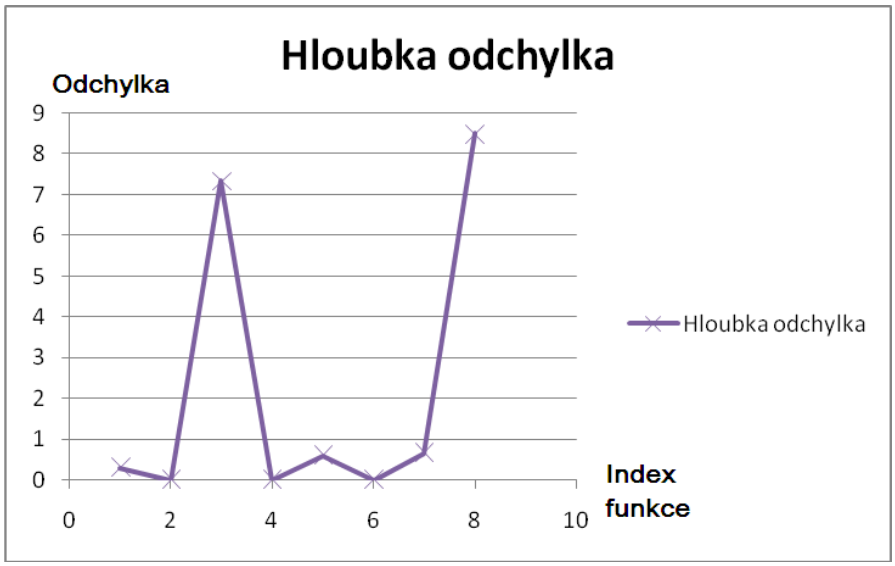
5.3.1 Hloubka NS na průběhu funkce

Tabulka 18 Hloubka NS pro různé průběhy funkcí jedné proměnné

Funkce	f01	f02	f03	f04	f05	f06	f07	f08
Hloubka průměr	1,1	1	15,1	1	1,2	1	3,5	15,8
Hloubka MAX	2	1	30	1	3	1	5	40
Hloubka MIN	1	1	4	1	1	1	3	8
Hloubka odchylka	0,3	0	7,34	0	0,6	0	0,67	8,5



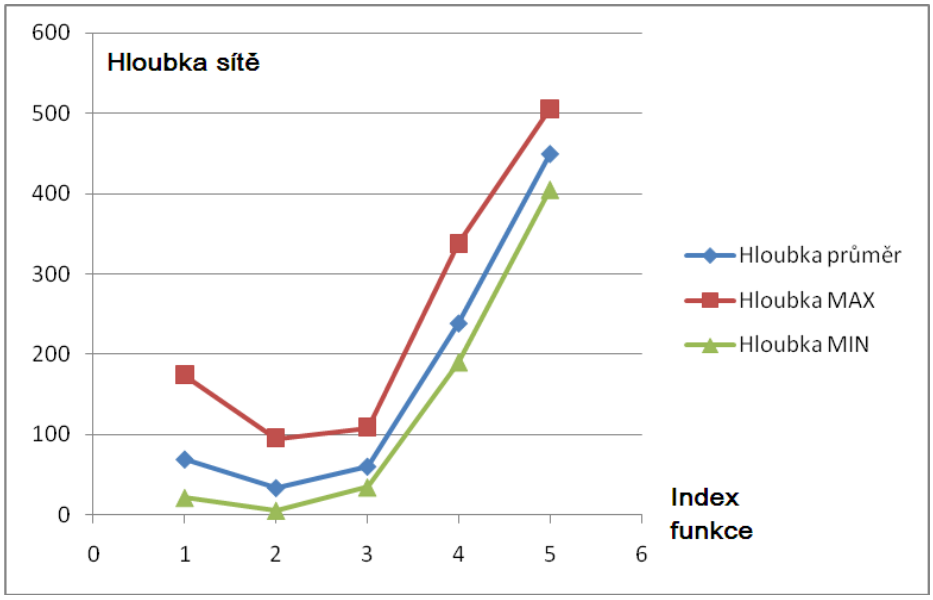
Graf 29 Hloubka NS pro různé průběhy funkcí jedné proměnné – hodnoty MAX, MIN a průměrná hodnota



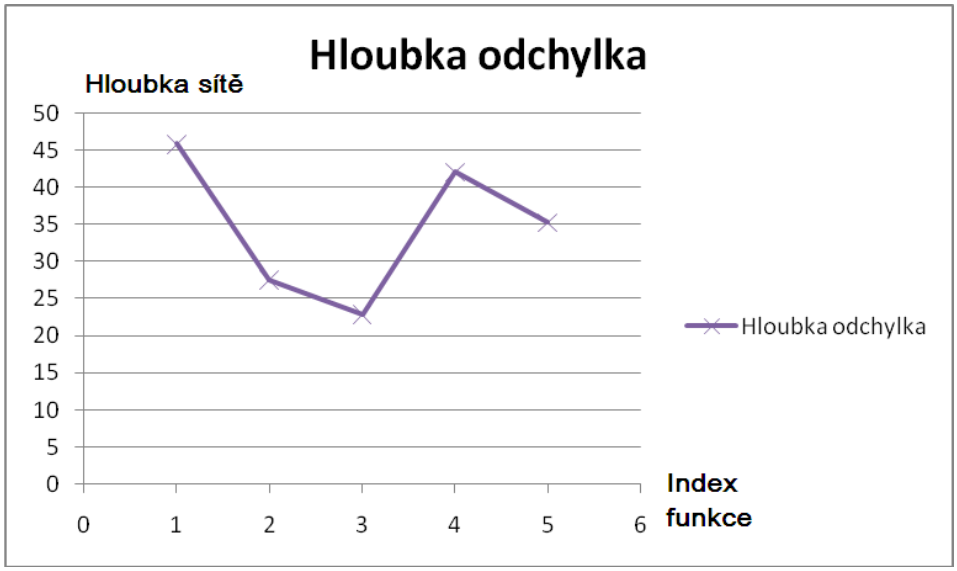
Graf 30 Hloubka NS pro různé průběhy funkcí jedné proměnné hodnoty MAX, MIN a průměrná hodnota

Tabulka 19 Hloubka NS pro různé průběhy funkcí dvou proměnných

Funkce	f01	f02	f03	f04	f05
Hloubka průměr	68,7	33	59,7	238,2	449,4
Hloubka MAX	174	95	109	338	506
Hloubka MIN	21	5	34	190	405
Hloubka odchylka	45,9	27,6	22,9	42,1	35,3



Graf 31 Hloubka NS pro různé průběhy funkcí dvou proměnných – hodnoty MAX, MIN a průměrná hodnota

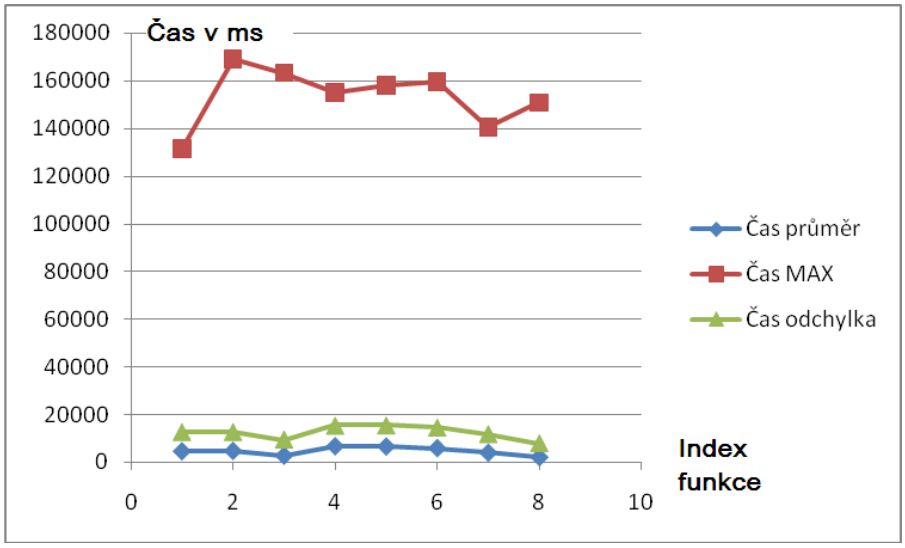


Graf 32 Hloubka NS pro různé průběhy funkcí dvou proměnných hodnoty MAX, MIN a průměrná hodnota

5.3.2 Čas výpočtu NS

Tabulka 20 Čas výpočtu NS pro různé průběhy funkcí jedné proměnné

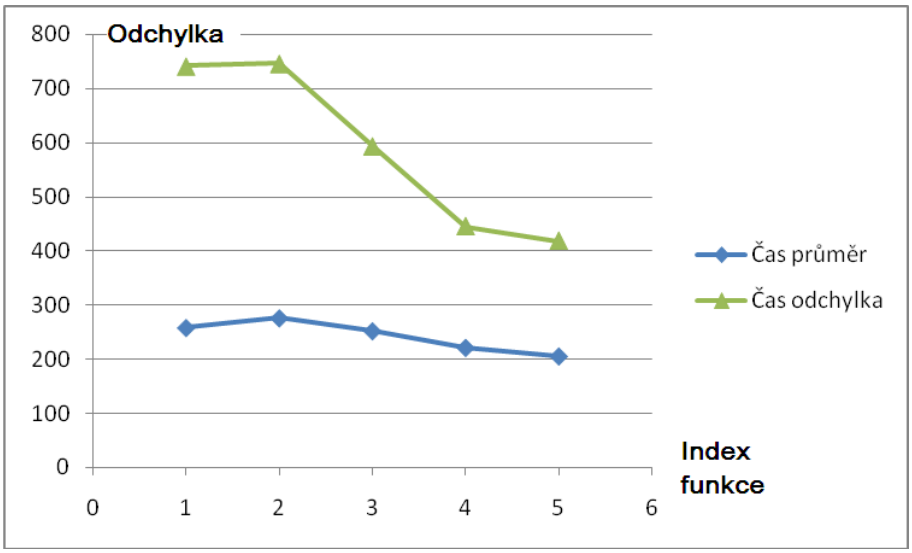
Funkce	f01	f02	f03	f04	f05	f06	f07	f08
Čas průměr	4551	4678	2715	6598	6498	5814	3924	2083
Čas MAX	131234	169125	163187	154984	157890	159453	140281	150750
Čas odchylka	12696	12713	9510	15400	15521	14595	11825	7832



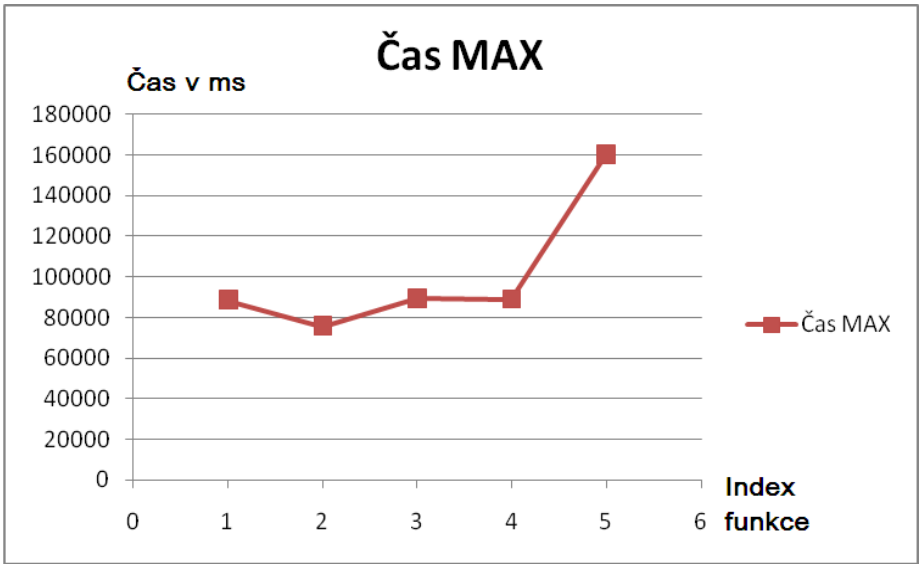
Graf 33 Čas výpočtu NS pro různé průběhy funkcí jedné proměnné – hodnoty MAX, průměru a odchylky

Tabulka 21 Čas výpočtu NS pro různé průběhy funkcí dvou proměnných

Funkce	f001	f002	f003	f004	f005
Čas průměr	258	276	252	221	205
Čas MAX	88625	75562	89312	88890	160203
Čas odchylka	741	746	593	445	418



Graf 34 Čas výpočtu NS pro různé průběhy funkcí dvou proměnných – hodnoty průměru a odchylky

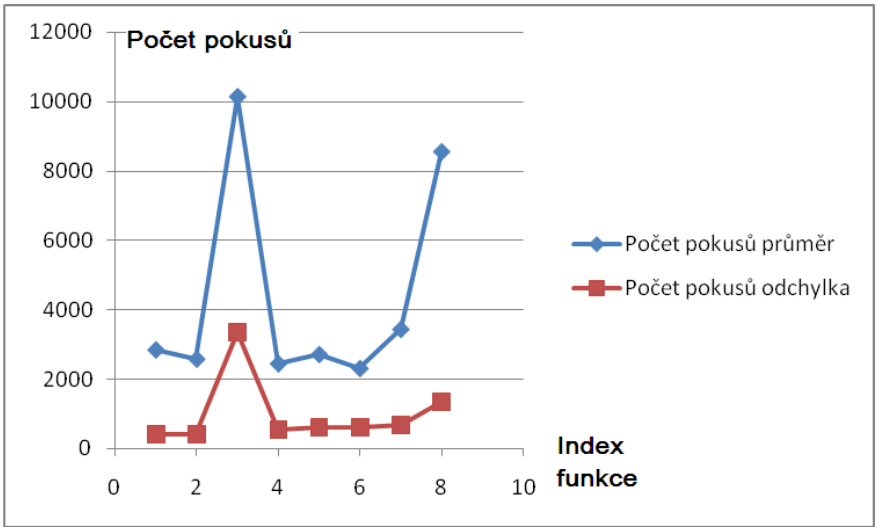


Graf 35 Čas výpočtu NS pro různé průběhy funkcí dvou proměnných – hodnoty MAX

5.3.3 Počet pokusů

Tabulka 22 Počet pokusů pro různé průběhy funkcí jedné proměnné

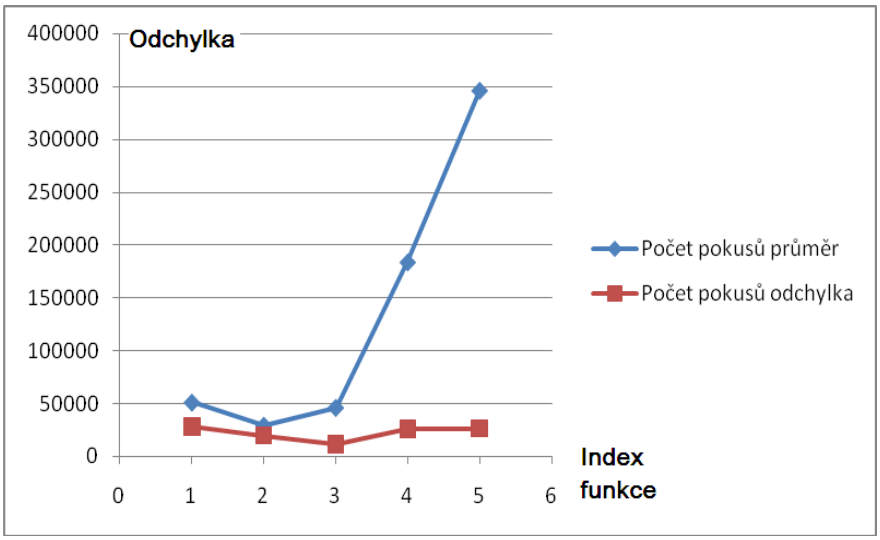
Funkce	f01	f02	f03	f04	f05	f06	f07	f08
Počet pokusů průměr	2860	2593	10137	2459	2727	2325	3448	8554
Počet pokusů odchylka	401	401	3346	534	597	612	668	1348



Graf 36 Počet pokusů pro různé průběhy funkcí jedné proměnné – hodnoty průměru a odchylky

Tabulka 23 Počet pokusů pro různé průběhy funkcí dvou proměnných

Funkce	f01	f02	f03	f04	f05
Počet pokusů průměr	50609	28817	45689	183348	345683
Počet pokusů odchylka	28148	19562	11199	25872	26417



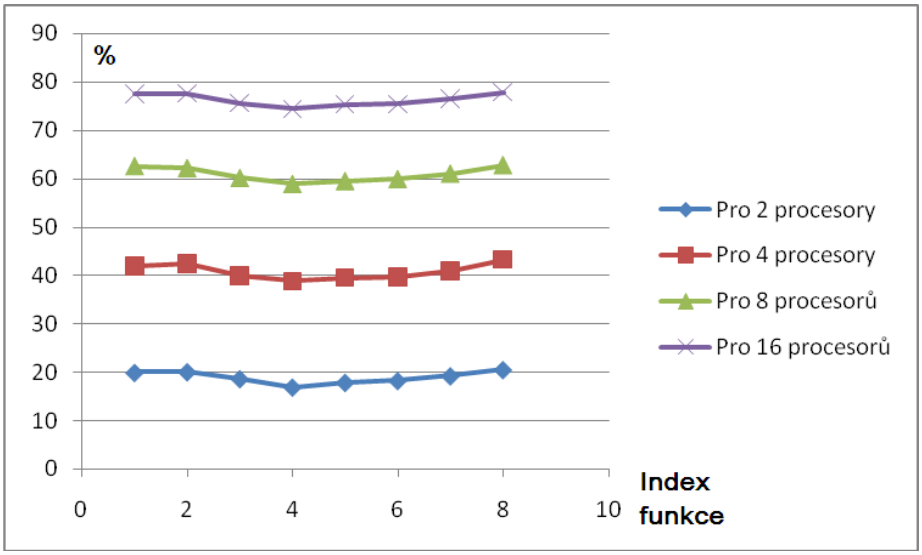
Graf 37 Počet pokusů pro různé průběhy funkcí dvou proměnných – hodnoty průměru a odchylky

5.3.4 Ušetřený čas pro různý počet procesorů

Jak ukazuje Tabulka 24 a Graf 38, tak ušetřený čas na různém počtu procesorů je na různých průbězích funkce jedné proměnné přibližně stejný.

Tabulka 24 Ušetřený čas podle počtu procesorů pro různé průběhy funkcí jedné proměnné

Funkce	f01	f02	f03	f04	f05	f06	f07	f08
Pro 2 procesory	19,95%	20,08%	18,66%	16,92%	17,89%	18,27%	19,25%	20,53%
Pro 4 procesory	41,91%	42,39%	39,86%	38,82%	39,47%	39,68%	40,96%	43,26%
Pro 8 procesorů	62,63%	62,2%	60,15%	58,88%	59,51%	59,93%	61,07%	62,83%
Pro 16 procesorů	77,54%	77,69%	75,58%	74,42%	75,31%	75,36%	76,51%	77,88%

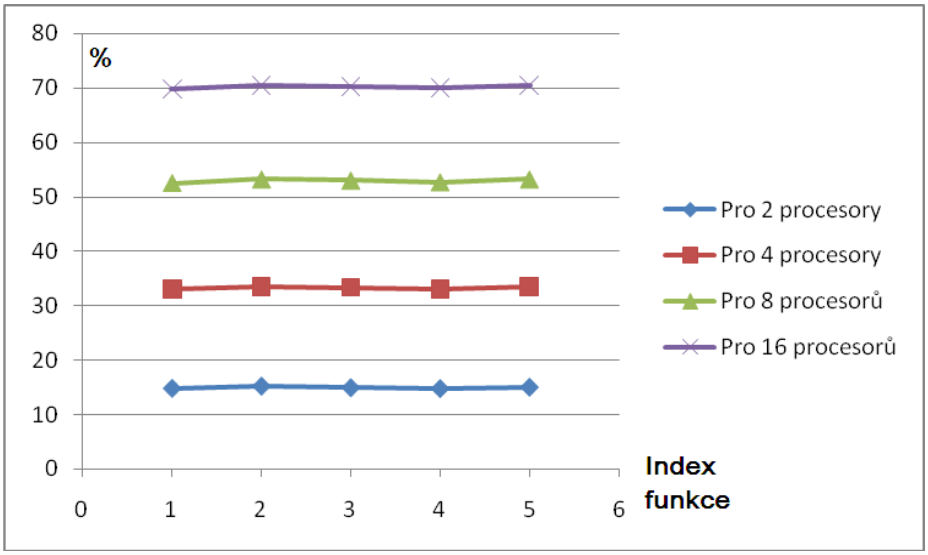


Graf 38 Ušetřený čas podle počtu procesorů pro různé průběhy funkcí jedné proměnné

Tak jako pro funkce jedné proměnné je i pro funkce dvou proměnných ušetřený čas na různém počtu procesorů pro všechny průběhy funkcí přibližně stejný. V tomto případě je shoda dokonce v řádu jednoho procenta. Výsledky ukazuje Tabulka 25 a Graf 39.

Tabulka 25 Ušetřený čas podle počtu procesorů pro různé průběhy funkcí dvou proměnných

Funkce	f001	f002	f003	f004	f005
Pro 2 procesory	14,82%	15,21%	14,93%	14,77%	15,02%
Pro 4 procesory	32,95%	33,49%	33,29%	33,01%	33,47%
Pro 8 procesorů	52,51%	53,2%	52,97%	52,65%	53,19%
Pro 16 procesorů	69,76%	70,44%	70,17%	69,91%	70,41%



Graf 39 Ušetřený čas podle počtu procesorů pro různé průběhy funkcí dvou proměnných

5.4 Zhodnocení výsledků dosažených v experimentech a doporučení z nich vyplívající

5.4.1 Experimenty s různým počtem TD

Z experimentů s různým počtem TD vyplívá, že s použitou implementací algoritmu a použití stolního dvoujádrového počítače je možné pracovat přibližně s 200 až 400 TD. Tento počet TD je dostatečný i pro některé reálné úlohy s testovacího benchmarku Proben1 [8].

Pokud bereme jako měřítko kvality syntetizované NS malou složitost její struktury při dosažení zadaného kritéria globální chyby. Není vhodné pracovat s příliš malým počtem TD, protože potom často není dosaženo minimální struktury v malém počtu pokusů o syntézu. Je potom třeba více opakování syntézy a z nich vybírat vhodnou NS.

Z hlediska času potřebného pro výpočet syntézy na stolním dvoujádrovém počítači se ukazuje počet TD nad 600 až 800 jako příliš velký. Roli zde samozřejmě hraje také složitost řešené úlohy. Pro syntézu NS s jedním vstupem je možné použít až 800 TD. Pro složitější úlohy s více vstupy je třeba zvážit kolik TD lze použít. Čím složitější je řešená úloha, tím méně TD je ještě možné použit. Pro 1000TD a procesor Intel Core 2 duo

1,8GHz trval výpočet funkce $f_1 = 3*a^3 - 2*a$ v průměru 36,8 hodin a pro $f_2 = 2*a^5 - 3*a^3 + a$ to bylo v průměru 41,6 hodin.

5.4.2 Experimenty s různým počtem vstupů syntetizované NS

Tyto experimenty byly zaměřené na to, jak složitou úlohu je z hlediska počtu vstupů NS možné syntetizovat s pomocí použité implementace algoritmu. Z výsledků vyplívá, že maximum vstupů NS vhodných pro použitý algoritmus je 6. Průměrná hodnota hloubky sítě je pro 6 vstupů, 400TD a funkci $f = 3a^3 - 2a + 3b^3 - 2b + 3c^3 - 2c + 3d^3 - 2d + 3e^3 - 2e + 3f^3 - 2f$ s použitím GFS1 475,13 a s použitím GFS2 273. S použitím GFS2 by zřejmě mohlo být použito i 7 vstupů.

Ukazuje se, že pro více vstupů NS prudce roste hloubka sítě a tím také složitost struktury NS. Roste také rozmanitost syntetizovaných řešení. Vzhledem k velké složitosti struktury takto vzniklých NS nabývá na důležitosti potřeba vybrat NS s co nejvíce úspornou strukturou. Je potom potřeba více opakování syntézy, aby taková NS byla nalezena.

Celkový čas potřebný pro výpočet syntézy NS s větším počtem vstupů také dramaticky roste. Pro 6 vstupů NS je již potřeba nejméně čtyřjádrový procesor. Na čtyřjádrovém procesoru Xeon byl průměrný čas výpočtu pro 6 vstupů, 400TD a funkci $f = 3a^3 - 2a + 3b^3 - 2b + 3c^3 - 2c + 3d^3 - 2d + 3e^3 - 2e + 3f^3 - 2f$ s GFS1 19,5 hodin a s GFS2 11,5 hodin.

Ze srovnání použití GFS1 a GFS2 je lepší použití GFS2, protože ve většině ohledů dává v použité implementaci algoritmu větší výkon.

5.4.3 Experimenty s různými průběhy aproximovaných funkcí

Tyto experimenty probíhaly s dvěma sadama funkcí, kdy jedna byla pro funkce jedné proměnné a druhá pro funkce dvou proměnných. Šlo o to, zda lze nějak intuitivně odhadnout na základě průběhu funkce složitost výpočtu syntézy. V experimentu s funkcemi jedné proměnné se jedna funkce ukázala pro syntézu náročnější, než se na pohled podle svého průběhu zdála. V experimentu s funkcemi o dvou proměnných se v podstatě potvrdil počáteční předpoklad o náročnosti výpočtu podle průběhu funkcí.

Pro funkce jedné proměnné a 100TD se průměrná hloubka NS pohybla v rozmezí 1 až 15,8 a pro funkce dvou proměnných v rozmezí 33 až 238,2.

Doba výpočtu byla pro funkce jedné proměnné a 100TD v rozmezí 0,23 až 0,28hod. Pro funkce dvou proměnných a 100TD v rozmezí 0,55 až 4,92 hodin.

Je vidět, že se složitějším průběhem funkce roste jak hloubka výsledné sítě, tak také celkový čas potřebný na syntézu a také odchylky ve struktuře výsledné NS. Je proto vhodné zkusit na základě hodnot TD odhadnout průběh funkce reprezentované naučenou NS. A zvážit jak bude NS náročná na syntézu použitým algoritmem.

5.4.4 Čas ušetřený na různém počtu procesorů

Ve všech experimentech se ukazuje, že s rostoucí složitostí úlohy čas ušetřený na různém počtu procesorů zůstává buď stejný, nebo klesá v řádu několika procent. Maximálně o 15%.

ZÁVĚR

Cílem práce bylo provedení experimentů s implementací algoritmu asynchronního Analytického programování. Podařilo se ověřit použitelnost tohoto algoritmu pro syntézu a naučení neuronových sítí aproximujících funkce více proměnných a to až do dimenze 6. Pro aproximaci těchto funkcí se ukázala jako lepší rozšířená množina GFS2, která poskytuje prokazatelně lepší výsledky jak z hlediska úspornější struktury výsledných neuronových sítí, tak z pohledu rychlejšího výpočtu.

Z hlediska počtu tréninkových dat je algoritmus nejvhodnější pro úlohy s 200 až 400 tréninkovými daty. Přičemž je při výběru úlohy třeba přihlédnout také k počtu vstupů aproximované funkce a použitému procesoru, na kterém výpočet poběží. Na čtyřjádrovém serverovém procesoru Xeon byl algoritmus použitelný i pro aproximaci šestirozměrné funkce s 400 tréninkových daty.

Další věc, ke které je třeba při výběru úlohy vhodné pro tento algoritmus přihlédnout je složitost průběhu apoximavané funkce. Je dobré odhadnout složitost průběhu aproximované funkce na základě konkrétních hodnot tréninkových dat.

Představenou implementaci algoritmu asynchronního Analytického programování je možné doporučit pro úlohy aproximace do šesti vstupů s 200 až 400 tréninkovými daty a přesností globální chyby okolo hodnoty 0,03. Je však třeba dávat pozor na složitost průběhu aproximované funkce. Dále je možné doporučit použití rozšířené množiny GFS2 a alespoň čtyřjádrového procesoru.

ZÁVĚR V ANGLIČTINĚ

Subject of this work was conduct an experiment with implementation algorithm asynchronous analytic programming. Succeed verify usability hereof algorithm for synthesis and precept neural network approximation function of several variables namely until dimension 6. To approximation these function showed like better extensive set GFS2 that the provides demonstrably better results how on the part of saving structure resulting neural network, so from look quicker calculation.

On the part of the number of training dates is algorithm optimal for exercise with 200 as far as 400 training dates. Whereas stand by selection exercise possibly also have a respect to of the number of entrances approximate function and used processor, whereat calculation will run. On four-core server processor Xeon was algorithm usable and for approximation six dimension function with 400 training dates.

Next thing, to which it is necessary at selection exercise suitable for this algorithm consideration is complication course approximation function. Is good estimate complication course approximate function on the grounds concrete values of training dates.

Superior implementation algorithm asynchronous analytic programming it is possible recommend for exercise approximation to the six entry with 200 as far as 400 training data plus exactness global errors round values 0,03. Further it is possible recommend using extensive set GFS2 plus at least four-core processor.

SEZNAM POUŽITÉ LITERATURY

[1] ZELINKA, Ivan, et al. Evoluční výpočetní techniky – principy a aplikace. Vyd. 1. Praha : BEN technická literatura, 2008.

534 s. ISBN 80-7300-218-3.

[2] HYNEK, J. Genetické algoritmy a genetické programování. 1.st ed. 2008. ISBN 978-80-247-2695-3.

[3] MARÍK, V.; ŠTEPÁNKOVÁ, O.; LAŽANSKÝ, J. Umelá inteligence (4). 1.st ed. 2003. ISBN 80-200-1044-0.

[4] ZELINKA, I. Umelá inteligence I. 1st ed. 1998. ISBN 80-214-1163-5.

[5] VAŘACHA, Pavel. Syntéza neuronových sítí metodou symbolické regrese. Zlín, 2006. 83 s. Diplomová práce. UTB.

[6] KASPŘÍKOVÁ, Eva. Analytické programování v C#. Zlín, 2008. 67 s. Diplomová práce. UTB.

[7] OPLATKOVÁ, Zuzana. Analytic programming. Zlín, 2003. 73 s. Diplomová práce. UTB.

[8] MALINKA, Marek. Přehled metod optimalizace struktury a učení neuronových sítí. Zlín, 2009. 87 s. Bakalářská práce. UTB.

[9] Koza J. R., Genetic Programming, MIT Press, 1998, ISBN 0-262-11189-6

[10] ZELINKA, Ivan; OPLATKOVÁ, Zuzana; NOLLE, Lars. Analytic programming. 2004. Symbolic regression by means of arbitrary evolutionary algorithms, s. 44-56.

[11] ZELINKA, Ivan . New optimalization techniques in engineering [online].[cit. 2011-05-15]. SOMA - Self-Organizing Migrating Algorithm, s. . Dostupné z WWW: <http://www.google.com/books?hl=cs&lr=&id=YRsq-Pz0kAAC&oi=fnd&pg=PA167&dq=Analytic+programming+by+Means+of+Soma+Algorithm+&ots=8mtmCReIEn&sig=Tqvuqqwh32Eo1jRk-fH9LpsKpvg#v=onepage&q=Analytic%20programming%20by%20Means%20of%20Soma%20Algorithm&f=false>.

[12] PRECHELT, L. Proben1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules. [online]. Available from www: http://digbib.ubka.uni-karlsruhe.de/volltexte/39794>

[13] BABIS, G., GEORGIOPOULOS, .M. Optimal feed-forward neural network architectures. [online]. Available from www: http://www.cse.unr.edu/~bebis/optimal_arch.pdf

[14] KRIŠKA, J., MAKULA, M. Neuroevolúcia cez rozširovanie topológie. [online]. Available from www: <http://hilbert.chtf.stuba.sk/KUZV/download/kuzv-kriska-makula.pdf>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

NS	Neuronová síť.
TD	Tréninková data.
AP	Analytické programování.
GP	Genetické programování
GE	Gramatická evoluce
GE	Diferenciální evoluce
SOMA	Samo organizující se migrační algoritmus
GFS	general funkcional set
DSH	Discrete Set Handling
CF	Cost Function
DE	Diferenciální evoluce

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM GRAFŮ