VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Řízení a analýza komplexních systémů

Control and Analysis of Complex System

2017

Bc. Matej Sirotiar

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student:	**Bc. Matej Sirotiar**
Studijní program:	N2647 Informační a komunikační technologie
Studijní obor:	2612T025 Informatika a výpočetní technika
Téma:	Řízení a analýza komplexních systémů Control and Analysis of Complex Systems
Jazyk vypracování:	čeština

Zásady pro vypracování:

Projekt je zaměřen na využití nekonvenčních metod k řízení komplexních systémů se zaměřením na řízení komplexních sítí. Toto řízení bude uskutečňováno jak klasickými metodami, tak nekonvenčními algoritmy, jako jsou evoluční algoritmy. Práce na projektu bude programátorského charakteru s tím, že se předpokládá zájem o studium evolučních technik a základních principů komplexních sítí. Evoluční algoritmy je novotvar pro označení algoritmů, které používají Darwinovu teorii evoluce a Mendelovy teorii dědičnosti k simulaci přírodních procesů za účelem řešení složitých problémů optimalizačního charakteru. Projekt se bude zabývat použitím těchto algoritmů včetně grafických vizualizací na řízení komplexních sítí a případně CML systémů. Navržené prostředí by mělo být modulární, a to s podporou připojení budoucích modulů.

Předpokládaná struktura práce je:
1. Seznámení se s problematikou.
2. Volba vhodného programovacího prostředí.
3. Volba vhodných algoritmů z oblasti evolučních technik.
4. Programová realizace těchto algoritmů v jednotném GUI.
5. Vizualizace všech realizovaných algoritmů.
6. Tvorba uživatelského manuálu.

Seznam doporučené odborné literatury:

[1] J. Nathan Kutz, Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big, Oxford University Press; 1 edition (September 15, 2013)
[2] Nino Boccara, Modeling Complex Systems (Graduate Texts in Physics), Springer; 2nd ed. 2010 edition (September 21, 2010)
[3] Dragoslav D. Silja, Decentralized Control of Complex Systems (Dover Books on Electrical Engineering), Dover Publications (January 17, 2012)
[4] Sean Meyn, Control Techniques for Complex Networks, Cambridge University Press; 1 edition (December 10, 2007)
[5] Zelinka I., Oplatková Z., Šeda M., Ošmera P., Včelař F., Evolutionary techniques – principles and applications, BEN, Prague, 2008, 598 p.
[6] Koza J.R. 1998, Genetic Programming, MIT Press, ISBN 0-262-11189-6, 1998
[7] Kvasnička V., Pospíchal J., Tiňo P. 2000, Evolučné algoritmy, STU Bratislava, ISBN 85-246-2000, 2000

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **prof. Ing. Ivan Zelinka, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 14.07.2017

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prehlasujem, že som tuto diplomovú prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave dňa 18. júla 2017 Sirotiak .....

Rád by som poďakoval vedúcemu menom prof. Ing. Ivan Zelinka, Ph.D. Predovšetkým za jeho odborné rady počas tvorby aplikácie a písania tejto práce. Taktiež by som poďakoval mojím rodičom, ktorý ma počas štúdia usmerňovali a podporovali.

Abstrakt

Táto práca poukazuje na odlišný spôsob riadenia a analýzy komplexných sietí. Tento spôsob je založený na prevode komplexnej siete na CML, ktorý predstavuje model deterministického chaosu. Model je založený na nelineárne spojitých rovniciach využívajúci chaotické mapy na generovanie chaosu. Riadenie je realizované evolučnými technikami. Súčasťou práce je ukázať schopnosť evolučných algoritmov riadiť deterministický chaos. Pre riadenie boli vybrané dva evolučné algoritmy: Diferenciálna Evolúcia a SOMA (Self-Organizing Migrating Algorithm). Výsledkom je aplikácia, ktorá bude vizualizovať komplexnú sieť reprezentovanú ako CML model Samotný model bude možno riadiť pomocou vybraných evolučných technik v statickom a real-time režime.

Kľúčové slová: CML, Evolučný algoritmus, Chaos, DE, SOMA, Komplexné systémy

Abstract

This thesis points to different method of control and analysis complex networks. This method is based on transfer complex network to CML, which represent deterministic chaos. CML is model based on non-linear continous equations using a chaotic maps to generate chaos. The managment control is realized by evolutionary techniques. Part of the thesis is show that evolutionary techniques are capable of control of deterministic chaos. Two evolutionary algorithms are used for chaos control: Differential Evolution and SOMA (Self-Organizing Migrating Algorithm). The result is an application to visualize a complex network represent as CML model. The model itself will be controlled using selected evolitionary techniques in either static or real-time mode.

Key Words: CML, Evolutionary algorithms, Chaos, DE, SOMA, Complex systems

Obsah

Zoznam použitých skratiek a symbolov

CML	–	Coupled Map Lattice
EA	–	Evolučný algoritmus
DE	–	Differential evolution
SOMA	–	Self-Organizing Migrating Algorithm
OGY	–	Ott E, Greboki C, Yorke J

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov zdrojového kódu

1 Úvod

Komplexné siete sú v súčastnosti pomerne veľmi skúmaná oblasť. Riadenie a analýza takýchto sietí je čoraz populárnejšia záležitosť. Exstuje viacero algoritmov a technik ako riadiť komplexné siete. Táto práca má za účel poukázať na možnosť riadenia komplexných sietí za pomocou CML systémov. Tento systém predstavuje model deterministického chaosu, ktorý je založený na nelineárne spojitých rovniciach využívajúcu chaotické mapy na generovanie chaosu.

Deterministický chaos objavený Lorenzom [1] je pomerne rozsiahla oblasť výskumu za posledné storočie. Lorenzov chaotický systém produkuje jeden z dobre známych kanonických chaotických atraktorov v trojdimenzionálnom priestore. Ďalším dobre známym chaotickým systémom je logistická rovnica založená na modele dravec-korisť. Pri štúdii chaosu je ako základný model používaný buď jednodimenzionálny (1D) alebo dvojdimenzionálny (2D) coupled map lattice (CML) [2]. Odkedy vytvárajú nelineárne systémy chaotické chovanie stávajú sa pozorovanými a skúmanými. Chaotické systémy sa stávajú životne dôležitou súčasťou vedy a techniky, pričom v poslednej dobe sa štúdia chaosu nezaoberá len pochopením a analýzou ale predovšetkým riadením a optimalizáciou. Pojem „riadenie chaosu“ predstavuje proces v ktorom je kontrola rozdelená a použitá tak, že pôvodné chaotické správanie môže byť stabilizované na konštantnej úrovni hodnôt alebo n-periodických cykloch. Od prvého experimentu riadenia chaosu vzniklo niekoľko metód, ktoré sú prevažne založené na OGY [3]. Veľa metód bolo prispôsobených pre časopriestorový chaos reprezentovaný ako CML, Medzi úspešné metódy riadenia patria aj evolučné algoritmy.

Spočiatku sa práca zaoberá stručným náhľadom na deterministický chaos a predstavením osobností, ktorý významne prispeli do teorie chaosu. Súčasťou je aj popis základného CML systému a vybraných evolučných technik. V ďalšej časti sa práca zaoberá vysvetlením komplexných sietí a ich prevodom na CML. Praktická časť sa zaoberá riadením CML systému pomocou evolučných technik. Obsahuje popis samotnej aplikácie a jej možnosti. Nasledovne sú vykonané experimenty, ktoré dokazujú schopnosť evolučných technik riadiť CML.

2 Teoria chaosu

Na počiatku 20. storočia sa zrodila nová veda, teórie chaosu. Ako prvý pozoroval chaotický pohyb francúzsky vedec Henry Poincaré, ktorý študoval problém troch telies. Tento problém spočíval v tom, že sa telesa navzájom gravitačne ovplyvňujú a Poincaré sa snažil vypočítať, a teda aj predpovedať ich pohyb. Pri štúdiu odhalil jav známy ako *citlivosť na počiatočné podmienky.*

Po ňom sa štúdiom dynamických systémov zabývalo niekoľko ďalších vedcov. Trvalo však dlhšiu dobu, než vedci prestali svet skúmať len pomocou lineárnych systémov, ktoré majú ustálený priebeh. Rovnako tak trvalo veľa rokov, než vedci prestali náš svet chápať len pomocou Euklidovskej geometrie. Ta dokáže definovať priamky, kružnice či obdĺžniky, ale v prírode nenájdeme také pravidelné obrazce. Iba pomocou fraktálnej geometrie môžeme napodobiť prírodu. Hlavným katalyzátorom vývoja teórie chaosu bol elektrický počítač. Väčšina matematických teórií chaosu zahrňuje jednoduché opakované iterácie, ale ich vývoj je nepraktické skúšať ručne. Počítače tento výskum veľmi zjednodušili [4].

2.1 Osobnosti

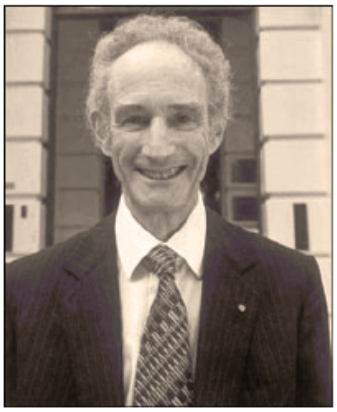
2.1.1 Edward Norton Lorenz

Za oficiálneho objaviteľa teorie chaosu bol americký matematik a meteorológ Edward Norton Lorenz (1917 - 2008). Študoval matematiku na strednej škole v New Hampshiru a neskôr na Harwarde v Cambridge. Počas druhej svetovej vojny pôsobil ako meteorológ pre vojenské letectvo a po vojne sa rozhodol venovať meteorológii. Zaviedol pojem **„podivný atraktor“**, ktorý znázornoval grafické výsledky jeho výpočtov. Jeden z jeho objavov, ktorý sa stal základným kameňom teórie chaosu je takzvaný **„motýlí efekt“**. Jeho atraktor spolu s Lyapunovým exponentom je symbolom chaosu. Všetky svoje výpočty vykonával na svojom počítači BLGP-30, ktorý dokázal previesť „neuveriteľných“ 17 operácií za sekundu [5].

2.1.2 Lord Robert May

Robert McCredie May alebo Baron May of Oxford sa narodil 8. Januára roku 1938 v Sydney. Chodil na chlapčenskú strednú školu a neskôr na univerzitu v Sydney. Na univerzite sa venoval chemickému inžinierstvu a teoretickej fyzike v ktorej roku 1959 získal doktorát. Hneď od začiatku svojej kariéry sa začal venovať populačnej dynamike zvierat a vzťahu medzi zložitosťou a stabilitou v prirodzenom spoločenstve. Aj keď nebol biológom spravil významné kroky v oblasti populačnej biológii prostredníctvom aplikácie matematických metód. Bol prednášajúcim aplikovanej matematiky na Harvardskej univerzite ale začiatkom 70 rokov odišiel späť do Sydney, kde sa začal zaoberať logistickou mapu ,ktorú publikoval roku 1970 [4].

a) Edward Norton Lorenz

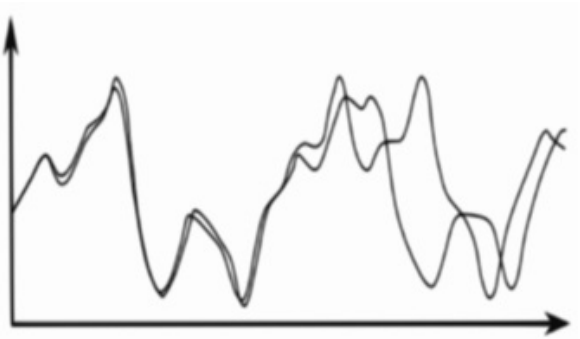


b) Lord Robert May

Obr. 1: Osobnosti chaosu

2.2 Motýlí efekt

Na prelome päťdesiatych a šesťdesiatych rokov sa veľa vedcov venovalo predpovediam počasia. Lorenz síce ešte nebol meteorológ ale počasie ho veľmi zaujímalo. Bol jeden z mála vedcov, ktorý mali k svojej práci počítač. V tej dobe boli i výpočty jednoduchých rovníc pomalé preto sa jedného dňa sa Lorenz rozhodol skrátiť čas výpočtom iba polovice grafu. Chcel preskúmať detailnejšiu časť grafu tak zadal počiatočné hodnoty, ktoré už predtým počítač spracoval. Keď bol graf hotový zistil že grafy sa podobajú len na začiatku ale od určitého bodu sa chovajú chaoticky (Obr. 2). Spočiatku si myslel, že ide o chybu počítača ale i po opätovnom výpočte bol výsledok rovnaký. Po nejakej dobe zistil že zadával hodnotu do počítača s trom desatinnými miestami, zatiaľ čo počítač počítal so šiestimi. Vďaka tomuto pochopil, že dlhodobá predpoveď počasia je úplne nemožná. Týmto spôsobom vznikol pojem Motýlí efekt. V počasí sa tento proces premieta do toho, čo je napoly vážne a napoly žartom-pojem vyjadrujúci to že ak dnes motýľ zvíri vzduch v Pekingu, budúci mesiac to môže zmeniť systém búrok v New Yorku. Mávnutie krídel motýli tu prestavuje malú zmenu v počiatočných podmienkach systému, ktorá spôsobí reťazec udalostí vedúcu k rozsiahlym javom ako je búrka či tornádo [4].



Obr. 2: Motýlí efekt v Lorenzovom atraktore

[1] Zdroj : [4]

[2] Zdroj: http://www.hungry-lord.wz.cz/data/osobnosti/lorenz$_o$*dchylka.gif*

2.3 Deterministický chaos a jeho výskyt

Determinizmus je vlastne presvedčenie, že každá udalosť alebo stav vecí, vrátane ľudského rozhodnutia je dôsledkom predchádzajúcich udalostí. Z výrazu „deterministický chaos“ plynie fakt, že systémy, ktoré ho produkujú možno modelovať. V týchto modeloch sa nevyskytuje žiadna náhodnosť avšak pre bežného pozorovateľa sa javia ako úplne náhodný systém. Rôzne náhle a nepredvídateľné zmeny v systémoch dokážu spôsobiť vznik chaotického správania. Dôležitou úlohu v týchto systémoch hrá nelinearita systému.

Nelineárny systém je taký systém kde neplatí princíp super pozície. V skutočnosti existuje malá množina izolovaných bodov pre ktoré platí princíp super pozície, ktoré sa nazývajú fixné body. No pre tie, ktoré to neplatí je nutné vypočítať zmenu stavu systému pomocou diferenciálnych rovníc, ale niekedy je to veľmi náročné a nie je zaručené že sa podarí predpovedať stav systému aj v budúcnosti [6].
Aby sme mohli klasifikovať chovanie systému ako chaotické, musí systém vykazovať nasledujúce vlastnosti:

1. Veľká citlivosť počiatočných podmienkach
2. Hustá množina periodických bodov
3. Topologická tranzitivita

Deterministický chaos stretávame v bežnom živote pomerne veľmi často a ani si to neuvedomujeme. Každý systém, ktorý je citlivý na počiatočné podmienky môžme považovať za deterministický chaos. Typickým príkladom je tlkot srdca alebo predpoveď počasia spojená s javom *Motýlí efekt*.

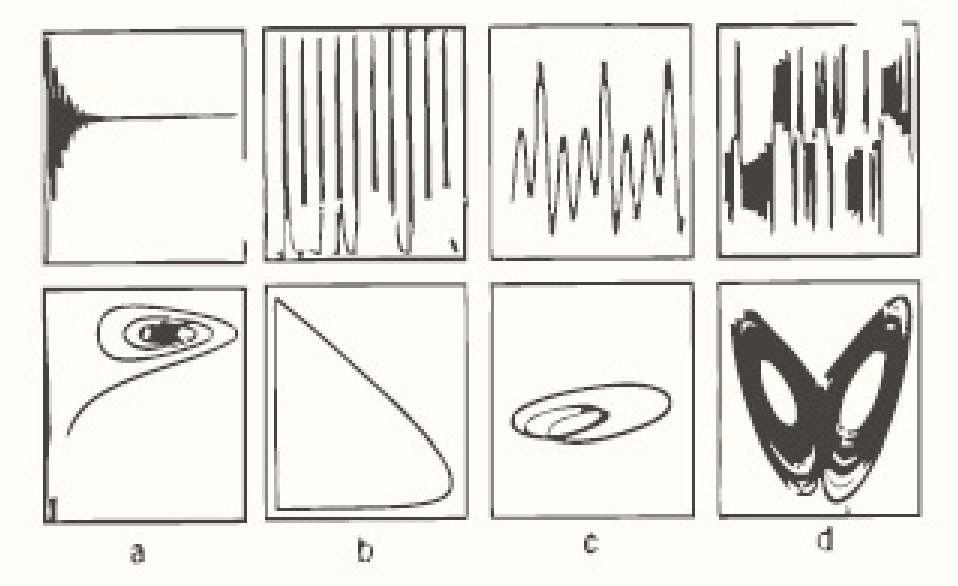
V oblasti biologie môžme chaos pozorovať vo vývoji populácie medzi viacerými druhmi živošíchov. Populačný vývoj jednotlivcov sa javí ako náhodný ale pravdou je že sa pohybuje deterministicky. Taktiež ho je možné pozorovať pri pohybe ryb alebo vtákov v hejne. Každý jednotlivec sa riadi malým počtom jednoduchých pravidiel, ale v kolektivnom chovaní je tento systém komplikovaný a chaotický.

Medzi dalšie známe oblasti patrí ekonómia, lekárstvo či chémia. Chaotické správanie môžme pozorovať aj vo fyzikálnych javoch ako je tektonika zemských dosiek, turbulencia tekutín alebo trajektória vesmírnych telies.Výskyt chaosu môžeme očakávať u systémov, ktoré obsahujú vhodnú nelinearitu, alebo pokiaľ medzi spolupracujúcimi systémami existuje nelineárna väzba. Samotný výskyt chaosu ešte nemusí znamenať, že sa v danom systému deje niečo zlého. Veľakrát to iba znamená, že je príslušný systém „na ceste“ ku kvalitnejšiemu usporiadaniu [7].

2.4 Chaotické atraktory

Atraktor, vlastne konečný stav systému, alebo inak povedané stav do ktorého systém v čase smeruje (je do neho priťahovaný). Bežným príkladom je pripojenie tlmiča ku kyvadlu. Bez ohľadu na počiatočnú pozíciu a rýchlosť sa bude blížiť ku kľudnému stavu (k jeho limite). Tento bod v strede, vlastne stav kedy je kyvadlo v kľude sa nazýva „atraktor“.

- Trajektória môže smerovať do jedného bodu, napríklad mechanické kyvadlo, na ktoré pôsobí trenie a časom sa zastaví. Tento prípad sa nazýva množina bodov a je to najjednoduchší príklad atraktora (Obr. 3a).
- Teleso, môže ustáli tak, že osciluje medzi niekoľkými hodnotami. Tento stav sa nazýva periodické body (spočítateľné), alebo kvaziperiodické body (nespočítateľné). Príkladom môže byť cudzie teleso, ktoré sa objavilo do našej sústavy a je priťahované slnkom. No po čase sa ustáli (Obr. 3b).
- Ďalším príkladom sú chaotické atraktory. Tieto atraktory sú veľmi citlivé na počiatočné podmienky a preto je dosť zložité dopredu predvídať ich chovanie (Obr. 3c).
- Zvláštnou skupinou sú podivné atraktory (Obr. 3d). Vykazujú veľkú zložitosť a taktiež fraktálnu štruktúru. Fraktál je objekt, ktorý môžme pozorovať v akejkoľvek mierke a vždy uvidíme rovnaký tvar.



Obr. 3: Príklady Atraktorov

[3] Zdroj: [4]

2.5 Chaotické mapy

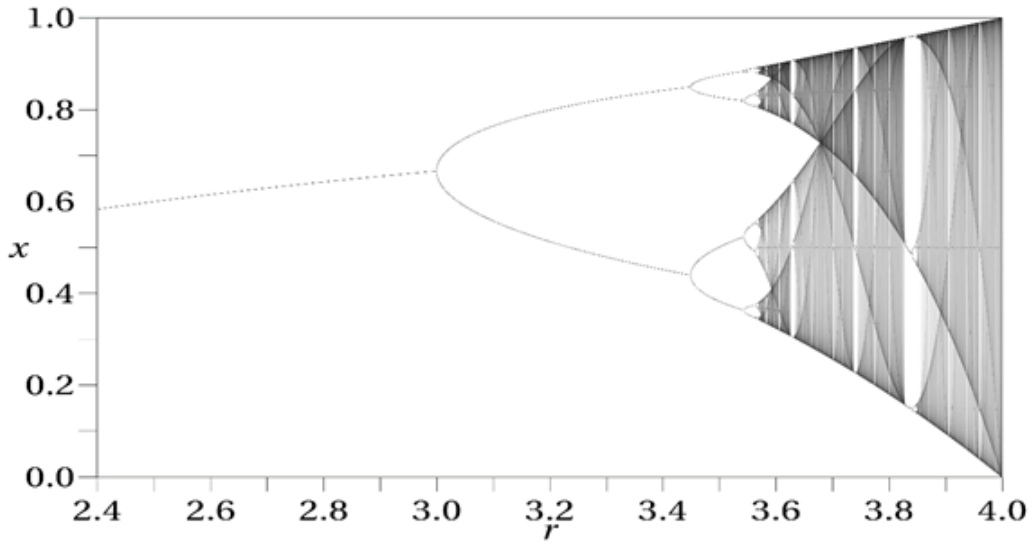
2.5.1 Logistická rovnica

Logistická rovnica je jedna z najjednoduchších modelov deterministického chaosu, publikovaná širokej verejnosti vďaka práci Roberta Maye (viz. Obr. 2). Bežne sa avšak používajú mapy, ktoré sú modifikované logistickou rovnicou. Príčinou vzniku rovnice bola simulácia biologických procesov ako je správanie rôznych živočíšnych druhov v ich prostredí. Jednoduchým príkladom môže byť spolužitie dvoch druhov v uzavretom prostredí ako je napríklad rybník, kde jeden druh predstavuje potravu pre druhého. Nevyvrátiteľným dôsledkom bude to, že požieraného druhu bude stále ubúdať, ale akonáhle bude požieraného druhu nedostatok, začne zomierať hladom dominantný druh, čo zase spôsobí zvýšenie jedincov požieraného druhu. Z tohto popisu je teda jasné že populácia druhov bude periodicky oscilovať alebo sa ustali na konštantnej hodnote [4, 20].

Logistická rovnica znie:

$$x_{n+1} = r * x_n(1 - x_n) \tag{1}$$

Parameter *xn* predstavuje pomer aktuálneho stavu populácie k maximálnej možnej populácii, teda stav v n-tom kroku a $x_n + 1$ v kroku *n+1* a parametr r predstavuje tempo rastu populácie. Časť rovnice 1-x predstavuje riešenie pre tu časť vývoja populácie kedy jeden druh nemá dostatok jedla a začnú umierať hladom. Túto rovnicu poprvý krát predstavil Perre Francois Verhulst.



Obr. 4: Bifurkačný diagram pre logistickú mapu

[4] $Zdroj : https://en.wikipedia.org/wiki/Logistic\_map\#/media/File : Logistic\_Bifurcation\_Resolution.png$

Závislosť chovania dynamického systému na jeho parametroch vyjadruje bifurkačný diagram. Chovanie logistickej mapy není výnimkou a veľmi často sa stretávame práve s týmto typom reprezentácia. Na obr. 4 možno vidieť bifurkačný diagram pre logistickú mapu, pričom na horizonálnej osi vidíme riadiaci parameter r a na vertikálnej ose hodnotu logistickej rovnice. Z obrázku je taktiež vidieť, že pri určitej hodnote riadiaceho parametru r dôjde k zdvojení periody a pri ďalšom náraste sa zdvojí už zdvojená perioda atď. Pri určité hodnote je už týchto zdvojení tolko, že systém vykazuje chaotické chovanie.

Ďalšou zaujímavosťou je fakt, že bifurkačný diagram vykazuje fraktálne chovanie a to tak že zdvojovanie period sa opakuje v presnom meradle.

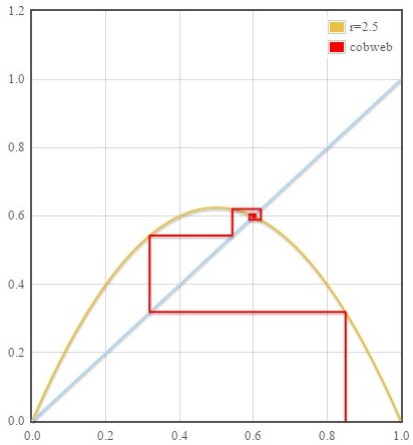
Správna voľba riadiaceho parametru ovplyvní správanie celého systému. Existuje niekoľko variant nastavenia parametru. Zmenu rozptylu hodnôt môžme sledovať na bifurkačnom diagrame, zobrazenom na obr.4. Pre túto rovnicu platí, že čím vačšia hodnota riadiacieho parametru r tým vačší chaos.

- **Hodnota riadiaceho parametra r $\in$ (0;1)**

  Pri riadiacom parametre menšom ako 1 bude systém konverguje k 0, čo znamená, že v modelu rastu populácie všetci zomrú.

- **Hodnota riadiaceho parametra r $\in$ (1;3)**

  Pri hodnote riadiaceho parametra v rozmedzí 1 až 3 sa vytvorí atraktor, ktorý smeruje k bodu *1 - 1/r*. Napríklad ak hodnotu r nastavíme na 2.5 tak systém bude konvergovať k bodu 0.6, pričom nezáleží na počiatočnej hodnote x_0.

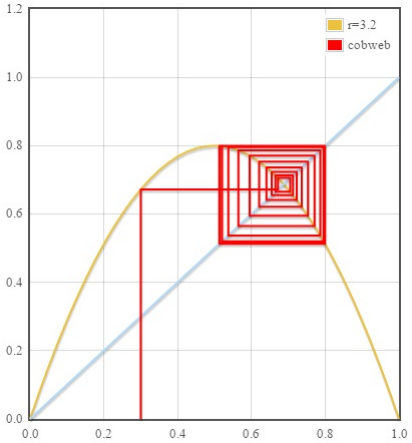


Obr. 5: CobWeb diagram pre x_0=0.85 a r=2.5

[5,6,7] Zdroj: http://sites.saintmarys.edu/ sbroad/example-logistic-cobweb.html

- **Hodnota riadiaceho parametra r $\in$ (3 ; 1+$\sqrt{6}$)**

  V prípade nastavenia riadiaceho parametru v tomto intervale vzniknú dva nestabilné body, medzi ktorými bude systém periodicky oscilovať. Napríklad ak zvolíme riadiaci parameter $r = 3.2$ a $x_0 = 0.4$ potom sa systém ustáli a bude oscilovať medzi 2 hodnotami 0.79945, 0.513044.



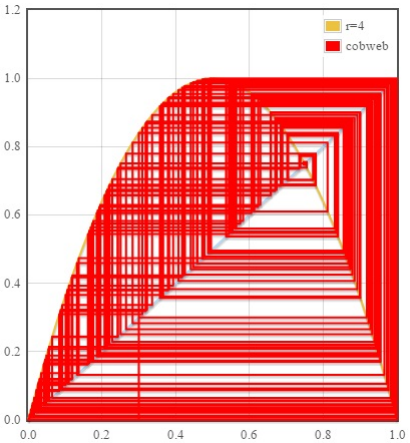
Obr. 6: CobWeb diagram pre x_0=0.4 a r=3.2

- **Hodnota riadiaceho parametra r $\in$ (1+$\sqrt{6}$;3.54409)**

  Postupným zvyšovaním riadiaceho parametru r bude nastávať tzn. jav *zdvojovanie periódy*. Systém bude postupne oscilovať medzi 4 hodnotami, potom medzi 8, 16 atď.

- **Hodnota riadiaceho parametra r$\in$ (3.54409;4)**

  Hodnota 3.54409 predstavuje hranicu od ktorej nemôžeme pozorovať žiadne oscilácie a nastáva *chaos*. Tento jav trvá až do hodnoty 4. Keby riadiaci parameter presiahol 4 tak sa celý systém by bol pritiahovaný k ∞.



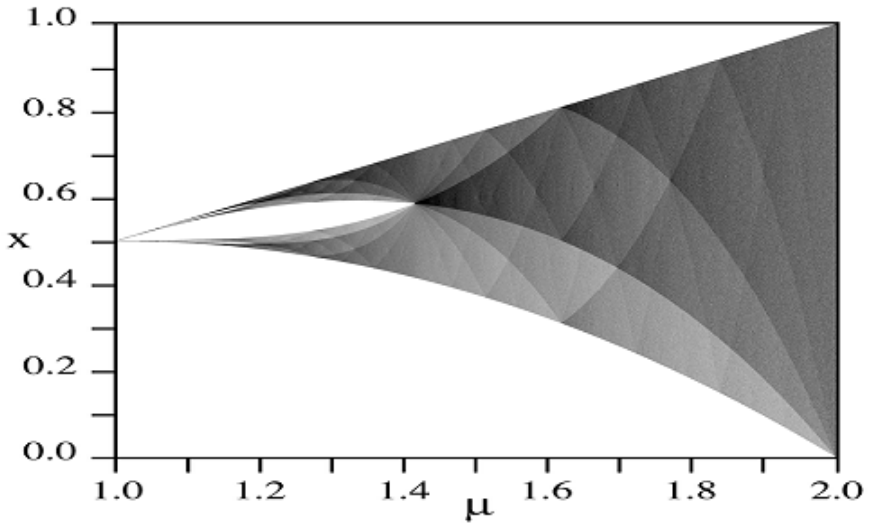
Obr. 7: CobWeb diagram pre x_0=0.3 a r=4

2.5.2 Tent mapa

Táto mapa získala svoje meno vďaka svojmu grafu, ktorý vyzerá ako stan (tent) [19, 20]. Porovnaním s logstickou mapou sú topologicky konjugatívne. Ich správanie je identické v zmysle správania sa pod iteráciou. Jej rovnica vypadá nasledovne:

$$x_n = \begin{cases} \mu x_n & \text{za podmienky } x_n < \frac{1}{2} \\ \mu(1 - x_n) & \text{v ostatných prípadoch} \end{cases} \tag{2}$$

Podobne ako u logistickej mapy, paramater x_n reprezentuje aktuálny stav a μ je riadiaci parameter ovplyvňujúci správanie systému. Toto správanie je zobrazené opäť pomocou bifurkačného diagramu (viz obr. 8). Z diagramu možno vidieť, že riadiaci parameter má hodnotu z intervalu (1;2).



Obr. 8: Bifurkačný diagram pre Tent mapu

- **Hodnota riadiaceho parametra $\mu\in$ (0;1)**

  Pri riadiacom parametre menšom v rozmedzí honôt 0 a 1 bude systém konvergovať k 0. Jedná sa o atraktor pevného bodu.

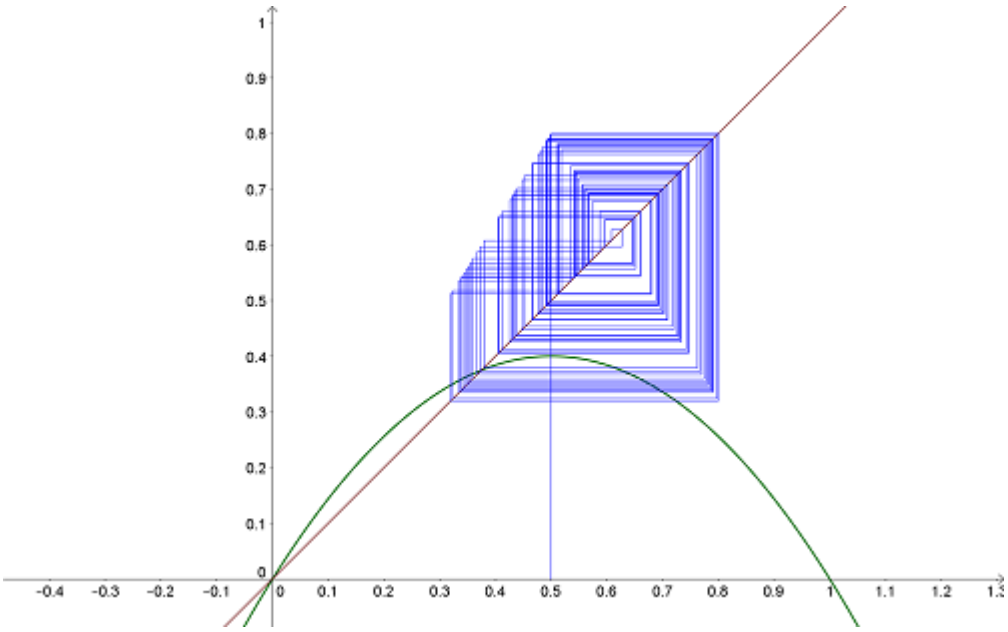
- **Hodnota riadiaceho parametra $\mu = 1$**

  Pri riadiacom parametre rovno 1 dosiahneme toho, že systém bude konvergovať k prvej hodnote menšej ako 0.5. Jedná sa opäť o atraktor pevného bodu.

[8] *Zdroj : https : //en.wikipedia.org/wiki/Tent_map/media/File : TentMap_BifurcationDiagram.png*

- **Hodnota riadiaceho parametra $\mu \in$ (1;2)**

  V tomto prípade docielime toho, že získané sekvencie budú kmitať v určitom intervale. Tento interval je $(\mu - \mu^2/2; \mu/2)$. Napríklad nastavením hodnôt x = 0.5 a μ =1.6 získame interval (0,32 ; 0,8) a sekvenciu hodnôt 0,5 -> 0,8 -> 0,312 -> 0,52-> atď.



Obr. 9: CobWeb diagram pre Tent mapu x_0=0.5 a μ =1.6

- **Hodnota riadiaceho parametra $\mu > 2$**

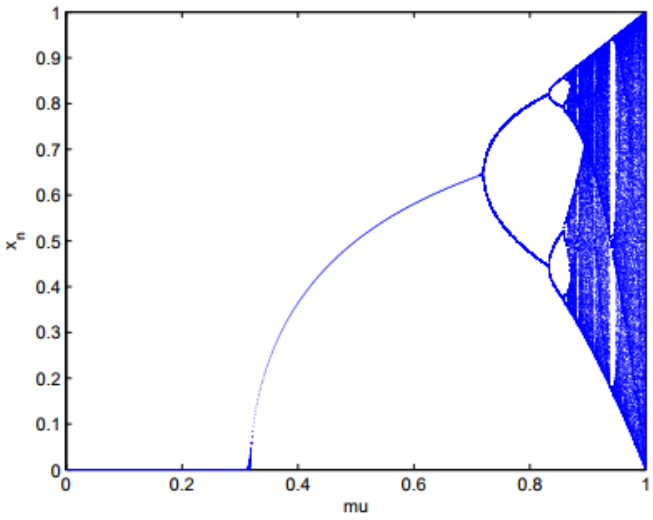
  Keby riadiaci parameter presiahol hodnotu 2, celý systém stratí svoje chaotické správanie a začne konvergovať k ∞.

2.5.3 Sínová mapa

Sínová mapa je veľmi podobná logsitickej a má skoro identický priebeh, avšak sa riadi inými vzťahmi [21]. Ako je možno vidieť na obr. 11 tak oproti logistickej mape dochádza k bifurkácii oveľa skôr, avšak pri porovnaním s bifurkačným diagramom logistickej mapy (viz obr.4) sú mapy z pohľadu bežného pozorovateľa takmer identické. Okrem faktu, že lyapunov exponent je o pol percenta menší a medzery medzi periodickými okrami sú menšie patrí rýchlosť bifurkácie medzí najvýznamnejší rozdiel sínovej a logistickej mapy. Jej rovnica znie nasledovne:

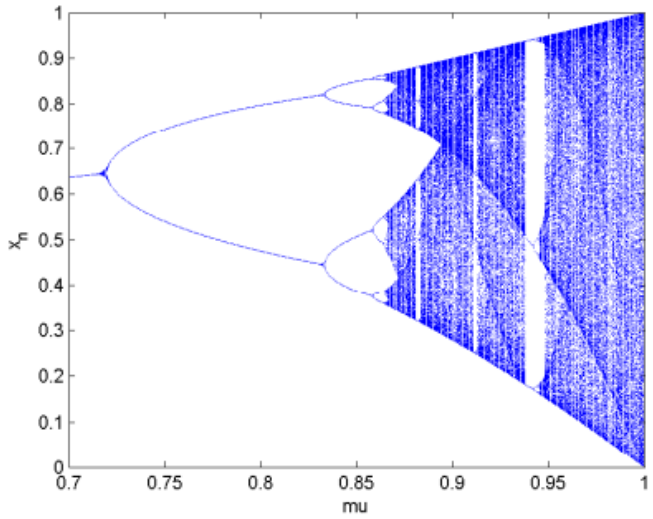
$$x_{n+1} = \mu * sin(\pi * x_n) \tag{3}$$

[9] Diagram vykreslení pomocu programu GeoGebra



Obr. 10: Bifurkačný diagram pre Sine mapu

Chovanie systému je možné vidieť pomocou bifurkačného diagramu na obr. 10. Podobne ako pri vyššie zmienených mapách je aktuálny stav systému vyjadrený premennou x_n a riadiaci parameter je μ, ktorého hodnota sa pohybuje v intervale [0;1]. Ako bolo spomínané sinová mapa má podobné vlastnosti medzi, ktoré patrí aj zdvojovanie periódy. Táto vlastnosť závisí na riadiacom parametre μ. Hodnoty menšia ako 0.345 majú za následok to, že systém bude konvergovať k 0. K tomu aby systém dosahoval chaotického správania ba sa mali použiť hodnoty blízke 1. Najčastejšie je používaná hodnota 0,999.



Obr. 11: Bifurkačný diagram pre Sine mapu

[10,11] Zdroj: [21]

3 Základný CML

CML (Coupled map lattice) je dynamický systém, ktorý modeluje správanie nelineárnych systémov a to predovšetkým parciálnych diferenciálnych rovníc. Prevažne sú používané k štúdiu chaotickej dynamiky priestorovo rozšírených systémov [8]. Systém bol predstavený roku 1980 radou niekoľkých publikácií [9]. Niektorý používali CML ako model pre chemické priestorové javy, iný sa usilovali aplikovať CML do elektrických obvodov. Existuje niekoľko modifikácií pre výpočet no najviac je používaný model predstavený Kanekom roku 1983. Tento model bol zavedený nasledovne:

$$x_i(n+1) = (1-\epsilon)f(x_i(n)) + \frac{\epsilon}{2}(f(x_{i-1}(n)) + f(x_{i+1}(n))) \tag{4}$$

Premenná x predstavuje aktuálne vybranú rovnicu a parameter n znázornuje priebeh v čase. Doležitá je funkcia generujúca chaos *f*. Pre základný model je použitá logistická mapa popísaná v kapitole 2.5.1. *Epsilon* predstavuje hodnotu vzájomného prepojenia rovníc , ktorá je nastavená na konstantnú hodnotu 0.8.

CML systém obsahuje X vstupov, kde každý vstup je v určitom rozmedzí. V tomto prípade je medzi hodnotami 0 až 1, pretože na generovanie CML systému používa logisticku mapu. Pre generovanie chaosu môžme použiť akúkoľvek inú chaotickú mapu ako je napríklad Tent alebo Sine. Každý jeden vstup si môžme predstaviť ako jednu rovnicu, ktorá bude aktualizovať svoju hodnotu podľa vzorca 2. Tento matematický zápis jednoducho vraví, že nasledovná hodnota rovnice je ovplyvnená z určitej časti aktuálnou hodnotou a taktiež hodnotami susedných rovníc. Pri správnych podmienkach si môžme po opakovaných iteráciách všimnúť ako CML generuje chaotický systém, ktorý sa môže zdať z pohľadu bežného pozorovateľa náhodný ale v skutočnosti sa jedná o deterministický chaos (viz obr. 16).

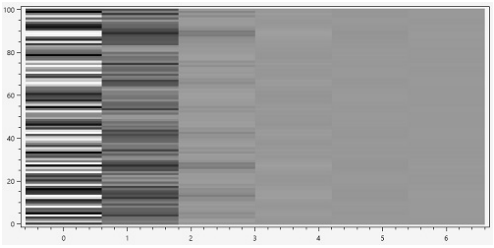
V jednoduchosti je model vlastne matica o veľkosti $X*N$, kde X (Equations) predstavuje množinu vzájomné prepojených rovníc a N (Iteration) predstavuje počet krokov v časovej línii. Každá bunka v matici obsahuje číslo v rozmedzí 0 - 1 čomu príslušní následná farba (0 – čierna 1 – biela). Spočiatku sa vygeneruje náhodný stav (0 - 1) pre každú rovnicu z množiny rovníc X, ktorý nachádza v čase (n_0). Pre každý časový krok (n+1) sa bunka aktualizuje podľa vyššie uvedené vzorca (2). Problém nastáva pri krajných rovniciach, pretože tieto rovnice majú iba jedného suseda. Konkrétne sa jedná o rovnice x_0 a x_{max}. Jednou z možnosťou je prepojenie týchto krajných rovníc. Tým docielime aby sa matica predstavovala valec.

3.1 Dôležitosť riadiaceho parametru

V predchádzajúcej kapile bolo spomenuté, že CML dokáže pri správnych podmienkach generovať pomocou logistickej rovnice chaos. Hlavnou príčinou je riadiaci parameter r. V nasledujúcich príkladoch bude možné vidiet ako veľmi ovplyvňuje riadiaci parameter r správanie CML systému.

- **Hodnota riadiaceho parametra *r=2.5***

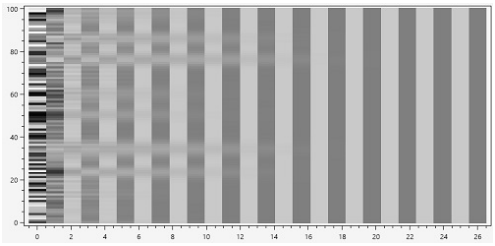
  Pri hodnote riadiaceho parametra 2.5 sa systém v priebehu niekoľkých iterácií ustáli na konstantnej hodnote a stagnuje.



Obr. 12: CML s riadiacim parametrom 2.5

- **Hodnota riadiaceho parametra *r=3.2***

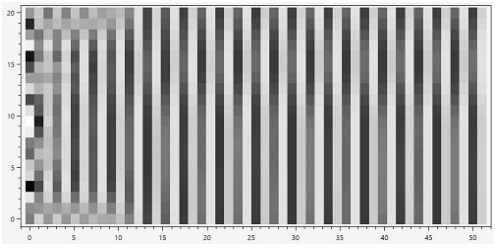
  Zvyšením parametra na 3.2 už systém prestal stagnovať na konštantnej hodnote. Namiesto jednej konstatnej hodnoty teraz periodicky osciluje medzi 2 hodnotami.



Obr. 13: CML s riadiacim parametrom 3.2

- **Hodnota riadiaceho parametra *r=3.5699***

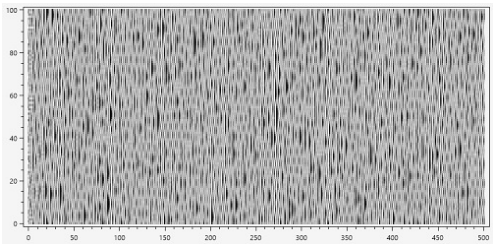
  Pri hodnote 3.5699 už systém periodicky osciluje medzi 4 hodnotami.



Obr. 14: CML s riadiacim parametrom 3.5699

- **Hodnota riadiaceho parametra r=4**

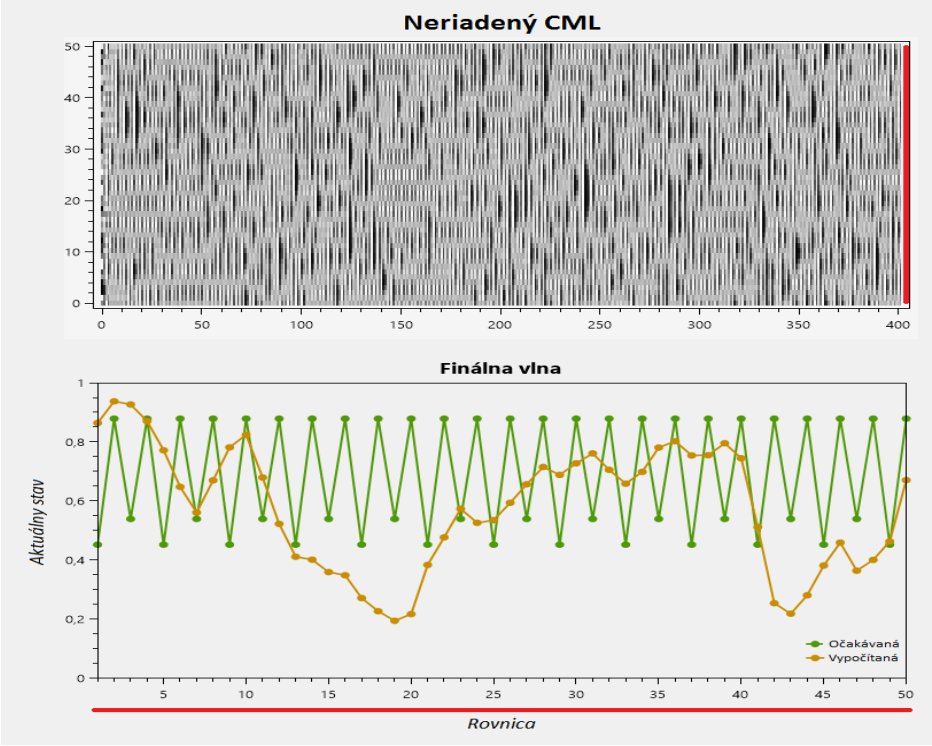
  Pri nastavení hodnoty riadiacieho parametru na 4 je možné vidieť, že sa systém správa chaoticky a začínajú sa miestami vytvárať chaotické štruktúry.



Obr. 15: CML s riadiacim parametrom 4

Na predchádzajúcich príkladoch bolo možné vidieť dôležitosť a hlavne závislosť riadiacieho parametru r pri generovaní CML. Porovnaním vygenerovaných CML príkladov s bifurkačným digramom logistickej mapy (Obr. 4) a CobWeb diagramov (Obr. 5-7) je možné vidieť rovnaké vzory vzhľadom na nastavenie riadiacieho parametru r, či už sa jedná o ustálenú konstantnú hodnotu, periodu niekoľkých hodnôt alebo chaosu.

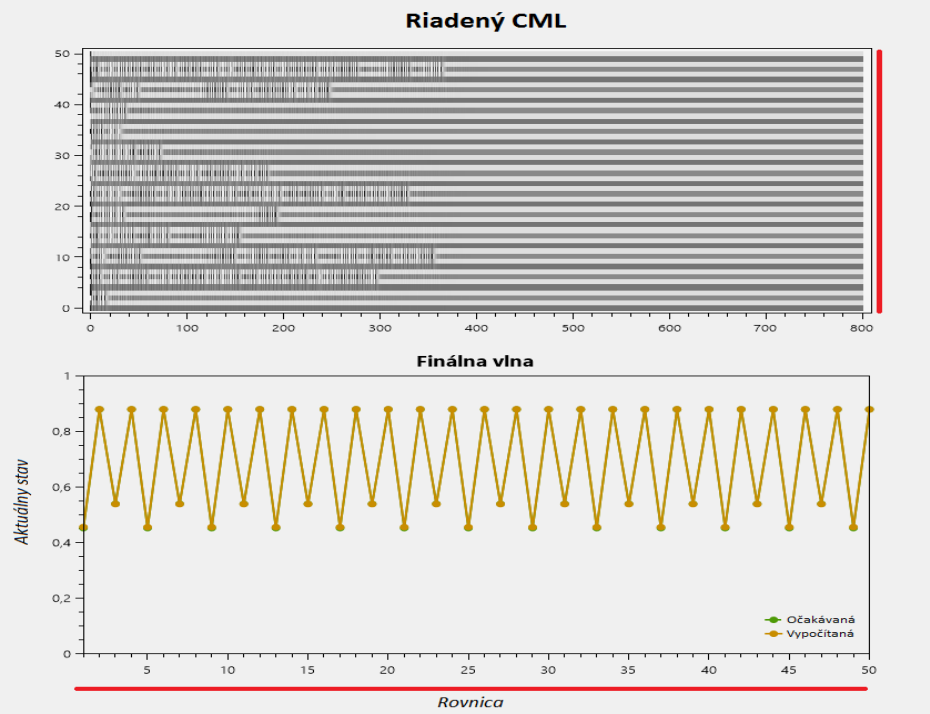
3.2 Neriadený CML



Obr. 16: Neriadený CML

Na obrázku 16 je zobrazený neriadený CML s počtom rovníc 50 a počtom iterácií 400. Vytvorený model sa generoval za pomocou logistickej rovnice s parametrom r o hodnote 4 a ako očakávaný stav bol použitý základný periodický stav. Červená priamka na konci CML zobrazuje finálnu vlnu, ktorá je premietnutá na grafe pod ním. Jednoducho povedané finálna vlna je to konečný stav systému po 400 iteráciách. Horizontálna os X (rovnica) predstavuje jednotlivé rovnice z množiny vstupných rovníc a vertikálna os Y ich aktuálny stav v rozmedzí 0 až 1. V tom prípade je aj konečný. Zelená krivka predstavuje očakávaný stav do ktorého sa systém snažíme konvergovať. Oproti tomu krivka oranžová predstavuj aktuálny stav. Ako môžeme vidieť hodnoty sa dosť líšia čo je úplne normálne vzhľadom nato že CML je neriadený a nevykonávame do systému žiadne zásahy.

3.3 Riadený CML



Obr. 17: Riadený CML

Obr. 17 zobrazuje rovnaký CML systém aj aký je predvedený na obrázku predchádzajúcom avšak s jednou zmenou. Tento systém bol rozšírený o riadenie. Riadenie spočíva v aktivácii PiningSites pre aktuálny stav systému. Ako môžeme vidieť systém už nepôsobí chaoticky ale postupne sa stabilizuje k očakávaného stavu. Už po 300 iteráciách je aktuálny stav systému takmer totožný z očakávaným stavom a pri 400 je už prakticky totožný. Pri opakovanom spúšťaní modelu pričom vždy s inými počiatočnými podmienkami (zmena počtu rovníc, zmena počiatočného stavu) sa systém pri základnom periodickom stave vždy vy stabilizoval pri maximálnom počte iterácií 500.

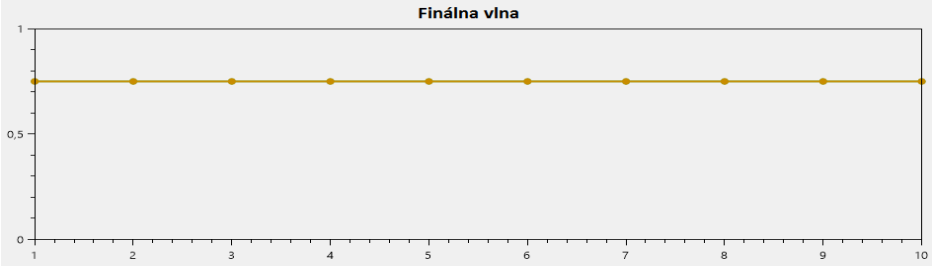
3.3.1 PiningSites

PiningSites reprezentuje veličinu, ktorá je počítaná na základne susedných väzieb daného stavu v CML systéme a je pripočítaná k aktuálnej rovnici. PiningSites potrebuje pre svoj výpočet nasledovné množiny pomenované ako **PiningControl** a **Pining**. Všetky spomínané množiny majú rovnakú veľkosť ako množiny X a to o počet vstupných rovníc.

- **PiningControl:** predstavuje hodnotu nabudenia pre odpovedajúcu rovnicu zvyčajne v rozmedzí 0 - 5. Jednoducho povedané akou silou bude aktuálna rovnica ovplyvňovať ostatné.
- **Pining:** je množina, ktorá obsahuje Bool hodnoty 0 a 1, ktoré predstavujú informáciu či bude daná rovnica rozšírená

3.3.2 Základný konštantný vzor

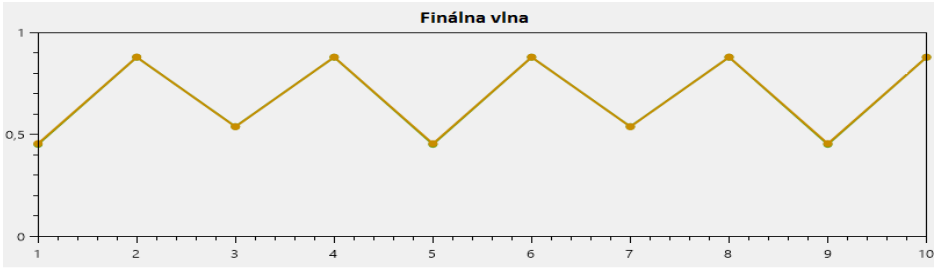
Základný konštantný stav predstavuje PiningControl nastavený na konštantnej hodnotou 3.1 pre všetky prvky z množiny. Pining je nastavená na každú druhú hodnotu. Stabilizácia CML systému na konštantný stav je pomerne veľmi rýchla. Už po 50 až 100 iteráciách je CML model kompletne stabilný. Obr. 18 zobrazuje finálnu vlnu konštantného stavu, ktorej hodnota je na konštante 0.75. [2].



Obr. 18: Konstantný stav

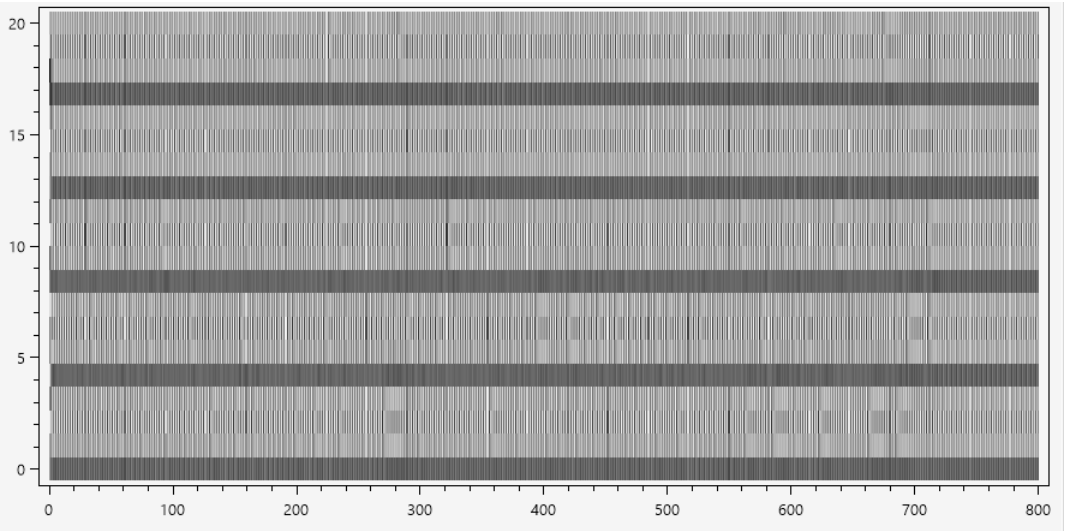
3.3.3 Základný periodický vzor

Základný periodický stav predstavuje PiningControl nastavený na konštantnú hodnotu 2.1 pre všetky prvky z množiny. Množina Pining je nastavená na každú 4 hodnotu. Na stabilizáciu CML systému zvyčajne postačí 600 iterácií. Obr. 19 zobrazuje finálnu vlnu periodického modelu. Táto vlna osciluje medzi 4 hodnotami : 0.45203913, 0,87896794, 0,5391973, 0,87896794.[2].



Obr. 19: Periodický stav

Predstavený konstantný a periodický stav sa viaže na CML, ktoré je generované logistickou mapou. Pri použitý iných chaotických map je veľmi nepravdepodobné aby sa model vystabilizoval do týcho stavov. Príkladom je CML založený na Tent mape pri použitý periodického vzoru (viz obr. 18). Ako je možné vidieť model sa snaží dosiahnuť výsledný periodický stav avšak to nedokáže a správa sa chaoticky. Rovnaké výsledky by sme dosiahli aj pri použití iných chaotických map. Problém je v tom, že stabilné stavy periodického vzoru odpovedajú len fixným bodom logistickej rovnice. Každá chaotická mapa má odlišné fixné body, to znamená, že pre každá mapa má aj svoje vlastné stabilné stavy. V tejto práci a nasledovných experimentoch bude CML model používať len logistickú mapu.



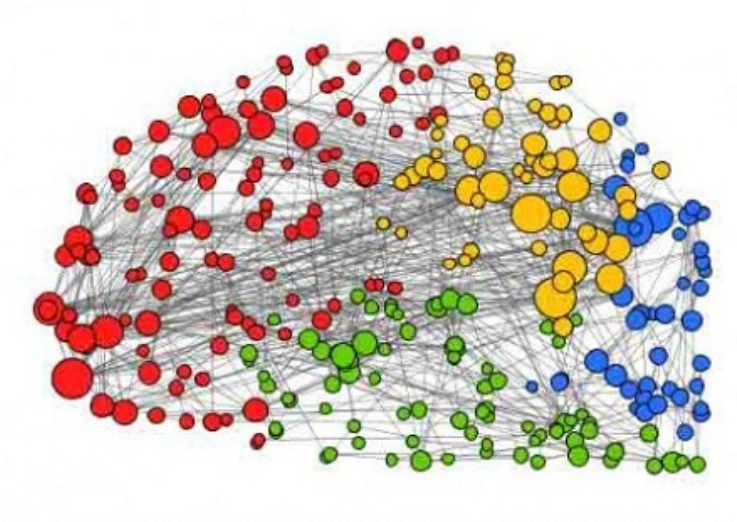
Obr. 20: CML s použitím Tent mapy

Obrázky použité v tejto kapitole tj. obr. 12 až obr. 20 sú výstupom mojej aplikácie.

4 Komplexné siete

Sieť je definovaná ako kolekcia entit, ktoré sú prepojené vazbami. Siete, ktorých štruktúra je veľmi zložitá a rozsiahla nazývame *komplexné*. Výskumy komplexných sietí zahŕňajú rozdielne vedecké oblasti. Poznatky a informácie, ktoré prinášajú nám dávajú lepšiu predstavu či už v oblasti sociálnych vazieb, prírodných zákonov, alebo iných nepreskúmaných oblastí. Spojením týchto poznatkov s viacerými obormi sa nám otvára viac možností pochopenia samotných zákonov vesmíru.

Základným typom komplexných sietí je **Sociálna sieť**. Príkladom takej siete môže byť Facebook, kde jednotlivé uzly predstavujú uživatelia a ich hrany predstavujú priateľstvo. V prípade ohodnotenej siete môžu hrany predstavovať počet odoslaných správ medzi uživateľmi. Dalším typom sú **Informačné siete**, ktoré majú za účel združovať informácie. Príkladom je citačná sieť, kde rôzne diplomové práce, vedecé publikácie, články a texty majú uvedené ako zdroje iné práce, publikácie a podobne. Ak medzi uzlami existuje hrana, znamená to že jedna práca odkazuje na druhú. Často sa jedná o hrany orientované aby bolo dobre rozoznateľné, ktorá práca čerpá z inej. **Technologické** siete sú veľmi rozsiahle ako napríklad Internet. Jeho uzly sú servery uložené po celom svete, pričom ich prepojenie napríklad káblom predstavujú hrany. Váha hrany môže predstavovať vzdialenosť alebo rýchlosť prenosu informácií. Do tejto skupiny patria aj transportné siete tj. cestné, železničné alebo energetické siete. Medzi poslednú základnú skupinu patria siete **Biologické.** Tieto siete prevažne zobrazujú potravinové reťazce, neurónové siete, interakciu proteionov alebo iné biologické procesy. Príkladom môže byť potravinový retazec, kde uzol predstavuje živočícha a orientovaná hrana znamená, že jeden požiera druhého.



Obr. 21: Ukážka komplexnej siete

[21] Zdroj: https://newmedialab.cuny.edu/wp-content/uploads/2014/01/attweb.gif

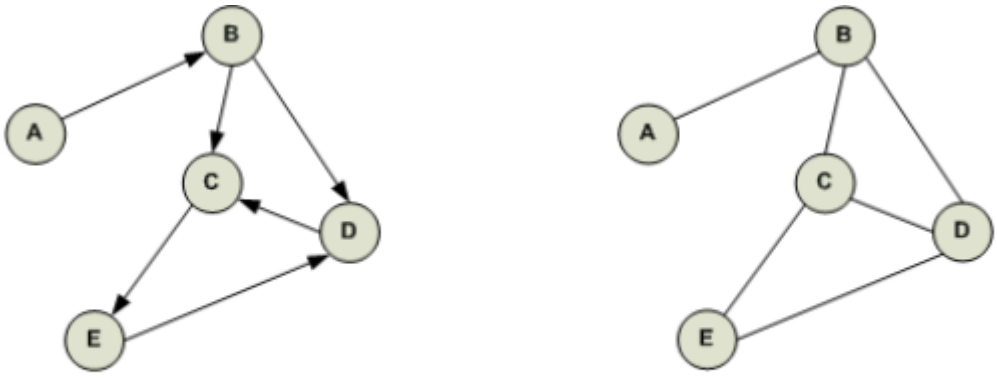
4.1 Teoria grafov

Teória grafu je teoretickým základom komplexných sietí. Pojem graf bol zavedení Leonhardem Eulerem v roku 1736. Jedná sa o model, ktorý reprezentuje objekty a vzťahy medzi nimi. V tejto kapitole si najskor nadefinujeme pojem graf a ukážeme si rôzne typy grafov. Potom nasledujú základné pojmy z teorie grafu a ukážka reprezentácií grafu.

4.1.1 Graf a jeho typy

Formálne je v [10] graf **G** definovaný ako usporiadaná dvojica **(V,E)**, kde **V** je neprázdna množina vrcholov a **E** je množina hran - množina dvojprvkových podmnožin množiny V.

Takto definovaný graf sa nazýva **neorientovaný**, pretože sa u neho nerozlišujú hrany *uv* a *vu*. Zavedením smeru potom dokážeme rozlížiť ich poradie. V grafe sa sa tento smer znázorňuje šipkou a jedná sa o **orientovaný** graf. Príklad pre orientovaný a neorientovaný graf je na nasledujúcom obrázku.



a) Orientovaný graf b) Neorientovaný graf

Obr. 22: Typy grafov

4.1.2 Základné pojmy teorie grafu

- **Stupeň vrcholu** označujeme ako *deg(u)*. Jedná sa o hodnotu, ktorá predstavuje počet hran zasahujúcich(spojených) do daného vrcholu. Ak sa jedná o orientovaný graf tak rozlišujeme či sa jedná o vstupný alebo výstupný stupeň vrcholu. Tieto hodnoty sa značia $deg_{in}(u)$ a $deg_{out}(u)$.
- **Cesta** v grafu je postupnosť vrcholov, pre ktorú platí, že v grafe existuje hrana z daného vrcholu do jeho nasledovníka. Žiadne dve hrany a vrcholy sa pritom neopakujú. Cesta, ktorá má najmenší počet prechádzajúcich hran sa nazáva **najkratšia**.

- **Shlukovací koeficient** udáva aká je pravdepodobnosť, že dva ľubovoľné uzly, ktoré majú aspon jedného spoločného suseda sú tiež prepojený hranou. Tento koeficient popisuje do akej miery sa vrcholy vytvárajú skupiny.

4.1.3 Grafové Centrality

Centralita vrcholov je jedna z metrik zachytávajúca jednotlivé vlastnosti topologie siete. Určuje ako veľmi je daný vrchol v sieti dôležitý. Nasledovne si popíšeme centrylity degree, closeness a betweeness. Samozrejme existuje oveľa viac centralit, avšak v tejto práci nám stačia len tieto základné.

- **Degree centralita** patrí medzi nejjednoduchšiu mieru centrality. Určuje počet hran alebo súčet vah hran incidentných s vrcholom. Táto centralita nám vraví, že čím viac bude mať vrchol susedov tým bude dôležitejší v sieti.

$$C_D(u) = \frac{d(u)}{N-1} \tag{5}$$

  ,kde $C_D(u)$ je hodnota degree centrality vrcholu u, *d(u)* je stupeň vrcholu u a N je počet všetkých vrcholov v sieti.

- **Closeness centralita** nám udáva informáciu o tom ako dlho bude trvať, než sa určitá informácia rozšíri z daného vrcholu do všetkých ostatných vrcholov. Matematicky je možno túto hodnotu vypočítať ako prevrátenú hodnotu súčtu najratších ciest ku všetkýchm ostatným vrcholom

$$C_C(u) = \sum_{v \in V} \frac{1}{d(u,v)} \tag{6}$$

  ,kde $C_C(u)$ je hodnota closeness centrality vrcholu u, V je množina všetkých vrcholov v grafe a *d(u,v)* je dĺžka najkratšej cesty z vrcholu v do vrholu v.

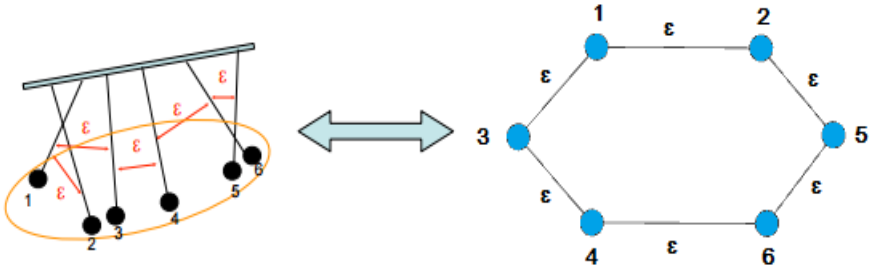
- **Betweenness centralita** určuje koľko najkratších ciest medzi dvojicami ostatných vrcholov prechádzajú daným vrcholom. Obecne čím viac najkratších ciest prechádza cez daný vrchol, tým je dôležitejší

$$C_B(u) = \sum_{i,j \in V, j,k \neq u} \frac{p(i,u,j)}{p(i,j)} \tag{7}$$

  ,kde $C_B(u)$ je hodnota betweeness centrality vrcholu u, *p(i,u,j)* je počet najkratších ciest medzi vrcholami i a j cez vrchol u, *p(i)* je počet najkratších ciest medzi vrcholami j a j

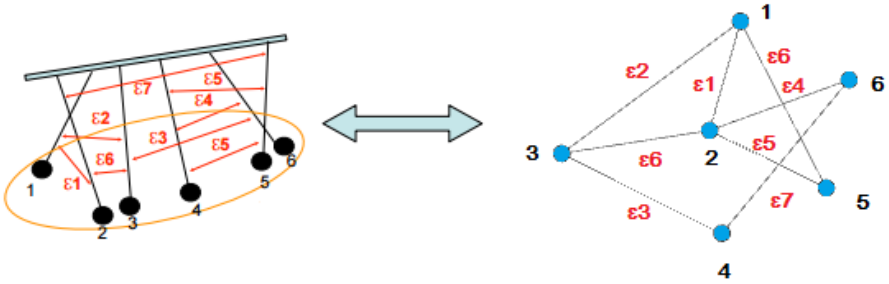
4.2 Komplexné siete reprezentované ako CML

Ukážku komplexných siete možno vidieť na obrázkoch 21, 22. V kapitole 3 sme si ukázali a vysvetlili najjednoduchšiu verziu CML. Táto verzia je zobrazená ako rada vzájomne prepojených rovníc, kde každá rovnica je prepojená so svojim najbližším susedom. Standartný CML môže byť pochopený ako rada kyvadiel, ktoré sú medzi sebou prepojené. Takúto vizualizáciu základnej verzie je možno vidieť na ľavej strane Obr. 23. Na pravej strane sa nachádza jeho ekvivalentná kompletxná sieť. Idea ekvivalencie medzi CML a komplexnou sieťou je celkom jedhoduchá. V oblasti komplexných sietí si je možné každú rovnicu v CML predstaviť ako vrchol a vazbu epsilon ako hranu.



Obr. 23: Základná verzia CML komplexná sieť

Pri porovnaní pokročilejšej verzie so základnou (Obrázok 23-24) je možné vidieť, že hrany v pokročilejšej vezrii CML niesú spojené s najbližším susedom ale sú spojené na základe štruktúry komplexnej siete. Tieto vazby už nie sú spojené systematicky a medzi jednotlivými uzlami je náhodný vzor spojenia. Týmto spôsobom môžme previesť ľubovoľnú komplexnú sieť na CML a naopak. Avšak to aj znamená čím komplexnejšia sieť tak komplikovanejší CML model a tým je horšie a zložitejšie nájsť správne riadenie.

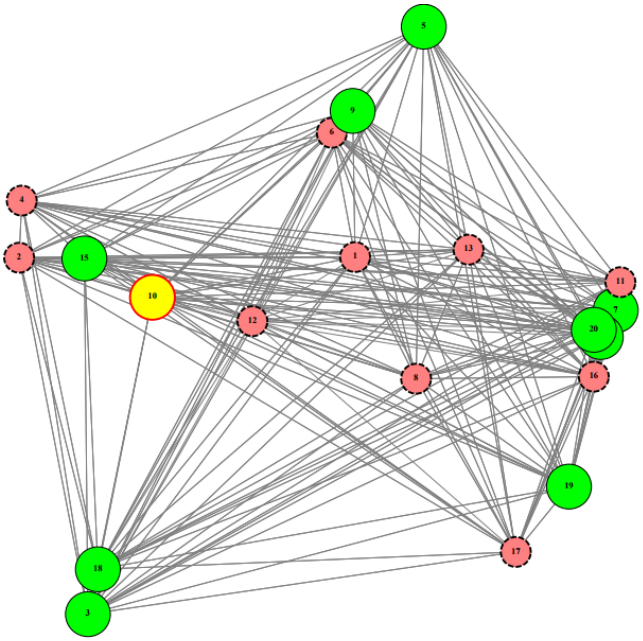


Obr. 24: Pokročilá verzia CML komplexná sieť

[25,26,27] Zdroj: [22]

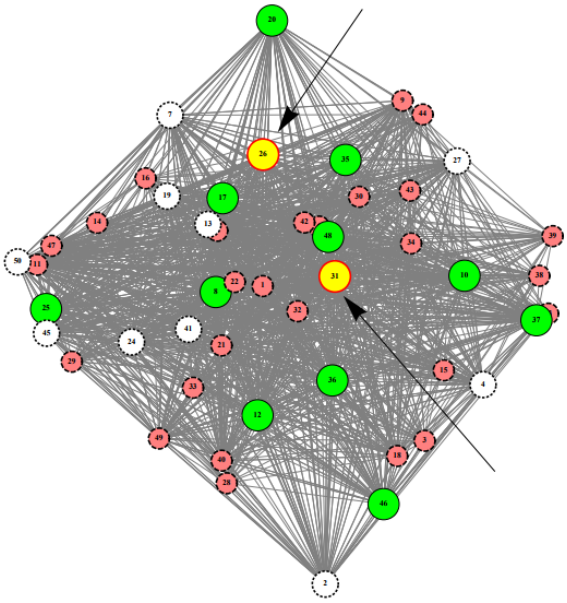
4.3 Populačná dynamika ako komplexná sieť

Jedna z problémov je získanie dat pre komplexnú sieť. V tejto časti je opísané ako sa dá previesť populačná dynamika na komplexnú sieť. Pre ukážku budú použité zmienené algoritmy DE (viz 5.1) a SOMA (viz 5.2). Každý algoritmus je založený na odlišnom princípe. Hlavná myšlienka je, že každý jednotlivec je reprezentovaný ako vrchol a vazby budú predstavovať dynamiku v populácii teda interakciu medzi jednotlivcami. SOMA algoritmus sa skladá z lídra, ktorý láka celú populáciu k sebe, pričom v každom migračné kole je zvolený nový leader. V DE algoritme je vynechána fylozofia, že horší rodič je nahradený lepším potomkom ale je prijatá fylozofia interperácie, že jednotlivec (horší rodič) je posunutý k lepšiemu riešeniu. Z tohto pohľadu to znamená, že vrchol nemôže byť zničení ani nahradený. Ak napríklad v DE/rand/1 je rodič nahradený potomkom tak potom je považovaný ako aktivácia vrcholu (horšieho rodiča) z troch náhodne vybraných vrcholov [16].



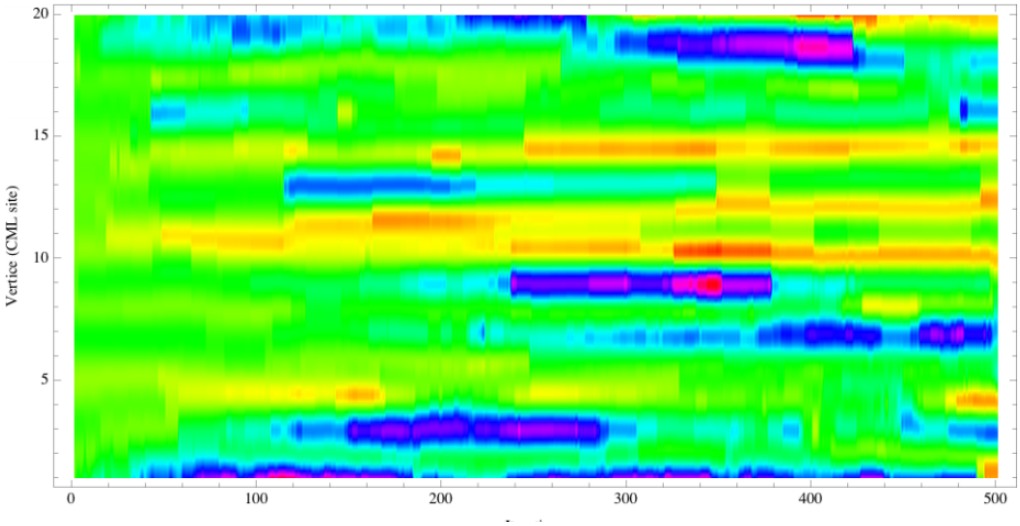
Obr. 25: SOMA - komplexná sieť

Obrázky 25 a 26 ukazujú, že interakcia medzi jednotlivcami v populácii vytvára štruktúru, ktorá pripomína komplexnú sieť. Hodnota vrcholov na zmienených obrázkoch je daná ako pomer prichádzajúcich a odchádzajúcich vrcholov. Zelené vrcholy predstavujú jednotlivcov, ktorý majú viac prichádzajúcich hran a opačne ružové vrcholy zase jednotlivcov obsahujúcich viac hran odchádzajúcich. V evolučnej problematike to znamená, že zelený vrchol je jednotlivec, ktorý bol viac krát úspešne vylepšený svojim potomkom. Opačne ružový vrchol znamená, že jednotlivec sa viac krát podielaľ na vylepšení iných potomkov. Oranžový vrchol označuje najlepšieho jednotlivca tj. najlepšie riešenie.



Obr. 26: DE - komplexná sieť

Znázornené komplexné sieť sú znázornené ako orientované hrany avšak šipky sú vynechané pre lepšiu viditelnosť. Obr. 25 znázorňuje dynamiku SOMA, kde jednotlivec 10 (žltý vrchol) je najlepšie riešenie. V prípade DE na obr. 26 sú dva najlepšie riešenia a to jednotlivci 26 a 31. V akomkoľvek prípade môže byť CML komplexná sieť počítaná a vizualizovaná v rozdielnych metrikách. Rozdielne úrovne excitácie vrcholov (jednotlivcov) môžu byť zobrazené rozdielnimy farbami (viz obr. 27). To môže byť reprezentované pomocou degree, closeness, betweenness centrality prípadne PagRank alebo iných metrik [17] .



Obr. 27: Populačná dynamika

5 Evolučné algoritmy

Pri optimalizácii problému je potrebné nájsť najlepšie riešenie v čo najkratšom čase, pričom toto riešenie nemusí byť najlepšie. Existuje iba jediný spôsob ako nájsť najlepšie riešenie a to výpočtom všetkých možných riešení, čo je pri väčšine problémov výpočtovo náročné až nereálne. Namiesto prechádzania všetkých možných riešení hľadajú je potrebné hľadať len časť prípustných riešení. Tímto prehľadávaním sa zaoberajú optimalizačné algoritmy, medzi ktoré patria aj evolučné algoritmy. Ich základný princíp je v použití modelov, ktoré využívajú niektoré známe princípy evolúcie, teda prírodného výberu, mutácie a rozmnožovania. Asi najväčšia výhoda evolučných algoritmov v porovnaní so štandardnými metódami optimalizácie je to, že je pomerne jednoduché ich pochopiť, implementovať a používať. Poskytujú riešenie pre zložité nelineárne problémy alebo dokonca neriešiteľné inými metódami a to v prijateľnom čase. Evolučné algoritmy pracujú s populáciou jedincov pričom každý jedinec má svoju **„fitnes“** hodnotu, ktorá určuje kvalitu jedinca.[11]

K dosiahnutiu úspešnej stabilizácie CML systému boli vybrané nasledovné evolučné algoritmy:

- Diferenciálna evolúcia[12]
- Self-Organizing Migration Algorithm (SOMA) [13]

5.1 Diferenciálna evolúcia

Práca publikovaná vedcami „Ken Price“ a „Rainer Storm“ roku 1994 predstavila pomerne nový spôsob riešenia optimalizačných problémov [12]. Hlavným cieľom tohto algoritmu je nájdenie globálneho optima v konečnom priestore riešení. Algoritmus vytvorí populáciu potomkov, ktorý medzi sebou súperia o pozíciu v novej populáciu až kým sa nenaplnia.

Činnosť a kvalita algoritmu je ovplyvnená nasledovnými parametrami :

- **CR -** parameter, ktorý sa nazýva prah kríženia a môže nadobúdať hodnoty z intervalu [0,1]. V prípade nastavenia prahu kríženia na 0 nastane situácia, že mutácia sa nedostane do skúšobného jedinca, čo bude maž za následok, že skúšovný jedinec bude kópiou svojho rodiča a evolúcia sa zastaví. Opačne ak parameter CR nastavíme na 1 tak potom bude skúšošný jedinec vytvorený iba z náhodne vybratých rodičov. Potom by sa algoritmus podobal skôr na náhodné prehladávanie ako na evolučný algoritmus. Vzhľadom na tento fakt by parameter CR nikdy nemal nadobúdať hodnoty 0 a 1.
- **D -** parameter predstavujúci dimenziu problému, ktorý predstavuje počet argumentov účelovej funkcie. Tento parameter je daný problémom, ktorý je momentálne riešený a ak by sme ho chceli pozmeniť tak by bolo nutné preformulovať daný problém.

- **NP -** parameter predstavujúci veľkosť populácie (počet jedincov). Minimálny veľkosť populácie by nemala klesnúť pod 4 avšak aby sa zabránilo stagnácii sa hodnota pohybuje v rozmedzí [10D,100D].
- **F -** parameter predstavujúci mutačnú konstantu z intervalu [0,2]. Táto konstanta by nikdy nemala nadobúdať hodnoty 0 alebo 1. Keby bol parameter nastavený na 1 tak ba sa jedinec "zmutoval"na samého seba a to by nemalo žiadny význam. V prípade nastavenia na 0 by jedinec stratil všetky informácie a uviazol v hodnote 0.
- **G -** parameter predstavujú počet evolučných cyklov počas ktorých sa evolúcia vyvíja

Mutácia tvorí časť tvorby nových jedincov. V prípade diferenciálnej evolúcie sa mutácia vykonáva na základe jednej z možných stratégií (viz tabulka). Podľa zvolenej stratégie sa vyberie **n** rôznych jedincov a vypočíta sa *šumový* vektor. Tento vektor nie nič iné len kombinácia **n** jedincov prenásobený mutačnou konstantou F.

Kríženie tvorí druhú časť tvorby nových jedincov, ktorá sa vykonáva až po procese mutácie. Kríženie funguje tak že sa so *šumového* vektora a náhodne vybraného jedinca vytvorí *skúšobný* vektor. Tento vektor je tvorený pomocou parametra CR. Pre každú dimeziu sa porovnáva náhodne vygenerované číslo a CR. Ak je toto náhodné číslo menšie tak sa do *skúšobného* vektoru zapíše hodnota zo *šumového* a naopak. Výsledkom je nový vektor zložení ako kombinácia 2 jedincov.

Stagnácia je jav v diferenciálnej evolúcii, ktorý zapríčiňuje zastavenie vývoja hodnoty účelovej funkcie smerom nadol ešte predtým ako bol dosiahnutý globálny extrém. Jedným z možných dôvodov je malé množstvo populácie. V prípade máme populáciu o 5 jednotlivcoch máme omezený počet možností ako vytvoriť *skúšovný* vektor. Avšak ani jeden zo všetkých možných *skúšobných* vektorov nie je lepší ako jedinci v aktuálnej populácii. Práve táto situácia evolúcia bude stagnovať. Tento prípad môže nastať aj pri vyššom počte jednotlivcov v populácii, avšak naopak príliš veľké množstvo jednotlivcov môže tiež spôsobiť stagnáciu.

Tabuľka 1: Stratégie Diferenciálnej evolúcie

Typ stratégie	**Výpočet**
DE/rand/1	NV= Xr1 + F* (Xr2 -Xr3)
DE/best/1	NV= Xbest+ F* (Xr2 -Xr3)
DE/rand to best/1	NV= Xr1+ F1* (Xr2 -Xr3)+F2*(Xbest-Xr1)
DE/rand/2	NV= Xr1 + F (Xr2-Xr3+Xr4)
DE/best/2	NV= X_{best} + F (Xr2-Xr3+Xr4)

Samostatný algoritmus je rozdelený do nasledujúcich krokov:

1. **Inicializácia** - Vygenerovanie počiatočnej populácie o veľkosti NP a výpočet fitnes hodnôt
2. **Mutácia** - Vytvorenie *šumového* vektoru NV na základe individuálnych jednotlivcov z aktuálnej populácie a konštanty F. Stratégie výberu jedincov je zobrazená v tabuľke 1
3. **Rekombinácia** - Vytvorenie skúšobného vektoru TV zmiešaným náhodného jednotlivca predchádzajúcej generácie a NV na základne konštanty CR.
4. **Výber** - Do novej generácie sa zapíše lepšie riešenie a to buď riešenie z predchádzajúcej generácie alebo novo vytvorený skúšobný vektor TV
5. **Konvergencia** - Algoritmus sa opakuje od bodu 2 až kým neprebehnú všetky generácie.

```
Input: Population_size, Problem_size, Weighting_factor, Crossover_rate
Output: S_best
Population ← InitializePopulation(Population_size, Problem_size)
EvaluatePopulation(Population)
S_best ← GetBestSolution(Population)
While (¬ StopCondition())
  NewPopulation ← ∅
  For (P_i ∈ Population)
    S_i ← NewSample(P_i, Population, Problem_size, Weighting_factor, Crossover_rate)
    If (Cost(S_i) ≤ Cost(P_i))
      NewPopulation ← S_i
    Else
      NewPopulation ← P_i
    End
  End
  Population ← NewPopulation
  EvaluatePopulation(Population)
  S_best ← GetBestSolution(Population)
End
Return (S_best)
```

Obr. 28: DE - Pseudocode

5.2 SOMA - Self-organizing Migrating Algorithm

SOMA [13] je stochastický optimalizačný algoritmus, ktorý je modelovaný na základe sociálneho správania a spolupráce jedincov (2004). Narozdiel od iných evolučných technik sa noví jedinci a potomkovia nevytvárajú pomocou kríženia rodičov, ale ich tvorba je založená na spoločnom prehľadávaní možných riešení v priestore. To znamená, že sa počiatočná populácia iba posúva (migruje) v priestore. Výsledkom jedného kola teda nie je Generácia ale „Migrácia“. Ako príklad si môžme predstaviť skupinu zlatokopov, ktorý hľadajú zlato. Migrujú po priestorovej oblasti, pričom sa nazvájom ovplyvňujú. Ak zistia, že niektorý má z nich má bohatšie ložisko ako oni sami začnú migrovať k nemu za účelom nájdenia väčšieho ložiska. SOMA delí parametre na riadiace a ukončovacie. Riadiace ovplyvňujú kvalitu ohodnotenia účelovej funkcie a ukončovacie podľa nastavených podmienok zastavia algorimus.

- **PathLength -** parameter určujúci maximálnu vzdialenosť pri ktorej sa jedinec zastaví od vedúceho jedinca inak povedané lídra. Zvyčajne nadubúda hodnoty z intervalu [1,5]. Hodnota by nemala byť menšia ako 1. Znamená to, že sa jedinec zastaví ešte pred lídrom a tým dôjde k degenerácii migračného procesu. Najlepšou variantou je hodnota okolo 3 aby jedinec prestretil lídra dostatočne daľeko.

- **Step -** tento parameter určuje veľkosť skoku na vzialenosti PathLength. Táto hodnota by mala byť oveľa menšia ako PathLength a zároveň by nemala byť násobkom vzdialenosti medzi aktívnym jedincom a lídrom. V prípade nedodržania týchto podmienok je veľmi pravdepodobné, že by migračný proces uviazne v lokálnom extréme.

- **PRT -** perturbačný vektor je jeden z najdôležitejších parametrov a zároveň aj najcitlivejších. Môže nadobúdať hodnoty z intervalu [0,1]. Jeho hlavnou úlohou je určenie smeru skoku aktívneho jedinca k lídrovy. V prípade nastavenia na 1 stráca algoritmus stochastické vlastnosti a začne sa správať deterministicky.

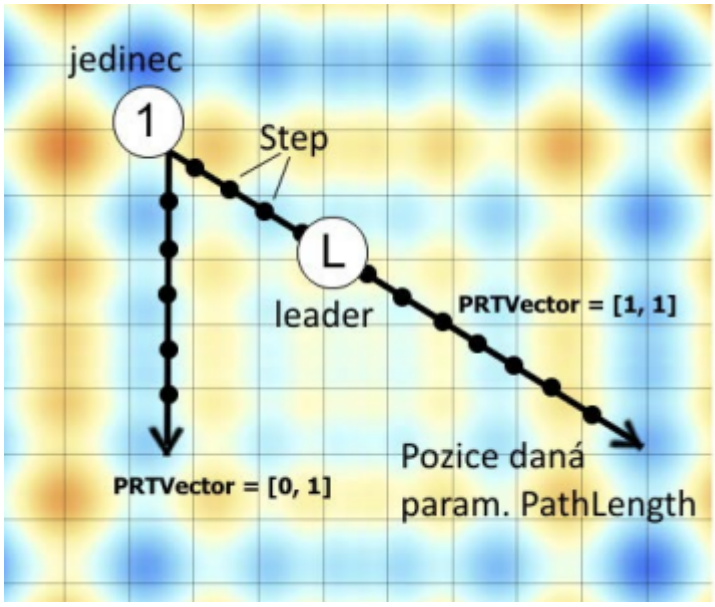
- **D -** podobne ako u Diferenciálnej evolúcie tento parameter údáva dimenziu problému, teda počet argumentov účelovej funkcie.

- **PopSize -** predstavuje počet jedincov v počiatočnej populácia. Odporúčaný minimálne počet je 10 jedincov a pri vyššej dimenzii problému na 0.2 až 0.5 D

- **Migrácia -** predstavuje počet evolučných cyklov počas, ktorých sa populácia preorganizuje. Jedná sa o ukončovací parameter. Najlepšou variantov ukončenia algoritmu jeho kombinácia s parametrom MinDEv.

- **MinDev -** predstavuje druhý ukončovací parameter. Jeho hodnotou je možné nastaviť maximálny rozdiel medzi najlepší a najhorším jedincom v aktuálnom stave populácie. Jednoducho povedané ak je tento rozdiel menší hodnota tohto parametru tak sa algoritmus ukončí. Ak je nastavený na veľmi nízku hodnotu tak je možné, že sa algoritmus nikdy neskončí. V prípade nastavenia vysokej hodnoty sa algoritmus ukončí skôr ako nájde globálny extrém.

Na začiatku každého evolučného cyklu (migrácie) sa zo všetkých riešení vyberie najlepšie a stáva sa lídrom pričom ostatný jednotlivci migrujú smerom k lídrovi. Na základe parametru **PRT** sa pre každého jedinca náhodne vygeneruje binárny perturbačný vektor, ktorý určuje pohyb jednotlivca. Vzdialenosť jednotlivca k lídrovi sa nazýva **PathLength**, pričom sa posúva po určitých skokoch **Step**. V prípade nastavenia vzdialenosti PathLength viac ako 1 jednotlivec preskočí leadra. Všetci jednotlivci, ktorý "skáču"smerom k lídrovi si po ukončení potrebných skokov vyberú to najlepšie a nasledne sa tam presunú. Týmto skončilo jedno migračné kolo a nasleduje dalšie. Obr. 29 znároňuje grafické vysvetlenie jednotlivých pojmov.



Obr. 29: SOMA - PRTVector

V súčasnosti existuje niekoľko variant základného algoritmu SOMA, pre ktoré sa často používa termín „stratégia“ a to hlavne pre zdôraznenie spolupráce jednotlivcov v populácii pri presune v priestore.

1. **AllToOne** - Patrí medzi základnú stratégiu. Na začiatku každého migračného kola sa každý jedinec ohodnotí pomocou účelovej funkcie a ten, ktorý bude mať najlepšiu sa stane lídrom nasledujúceho kola. V tomto kole sa lídrova pozícia nezmení ale ostatný začnú k nemu pohybovať.

2. **AllToAll** - Táto stratégia nemá Leadra. Každý jednotlivec migruje ku všetkých ostatným z populácie a to vždy zo svojej počiatočnej pozície na začiatku migračného kola. Tento typ stratégie je veľmi náročná na výpočet avšak je veľká pravdepodobnosť nájdenia globálneho extrému.

3. **AllToAllAdaptive** -na rozdiel od AllToAll stratégie, kde sa jednotlivec posúva na novú pozíciu až po ukončení všetkých migráčných kôl táto stratégia posúva jednotlivca každe migračné kolo. V prípade ak by našiel jedinec po ukončení jedného migračného kola lepšiu pozícia tak sa tam presunie.

4. **AllToOneRand** - podobná stratégia ako AllToOne až na fakt že každý jednotlivec si náhodne zvolí svojho leadra z aktuálnej populácie.

[29] Zdroj: [13]

5.3 Účelová funkcia CML

Účelová funkcia predstavuje číselné ohodnotenie kvality jednotlivca v danej populácii. V tomto prípade je počítaná ako rozdiel hodnôt očakávaného stavu s aktuálnym stavom CML podľa [9]. Minimálna hodnota fitnes funkcie zaručuje že ide o najlepšie . Cieľom všetkých simulácií je nájsť čo najlepšie riešenie to je teda také ktoré vracia najnižšiu možnú hodnotu.

Fitnes hodnota je daná nasledujúcim vzorcom:

$$f_{cost} = \sum_{i=0}^{X} \sum_{j=m-n}^{m} |TS_{ij} - CML_{ij}|^2 \tag{8}$$

Parameter X predstavuje počet rovníc v CML systéme. TS obsahuje hodnoty očakávaného stavu a CML aktulneho stavu. m označuje celkový počet iterácií a j jeho aktuálnu iteráciu. Podobne n predstavuje počet spätne kontrolovaných iterácií primálne nastavených na 10. Parameter i znázorňuje aktuálnu rovnicu.

Druhá alternatívna fukncia vznikla miernou modifikáciou prvej:

$$f_{cost} = p1 + (p2 * \sum_{i=0}^{X} \sum_{j=m-n}^{m} |TS_{ij} - CML_{ij}|^2) \tag{9}$$

Parametre p1 a p2 sú heuristicky nastavené hodnoty. Hodnota p1 reprezentuje počet PiningSites v CML a hodnota p2 predstavuje hodnotu na „upútanie pozornosti“ v evolučnom procese. Primárne je nastavená na 1000. Optimálne riešenie by malo vrátiť počet použitých PiningSites čo je hodnota p1 (prvá časť funkcie 4). Na prvý pohľad sa možno môže zdať že fitnes funkcia (4) je navrhnutá zle pretože z matematického hľadiska je najlepšie a zároveň najnižšie riešenie hodnota 0. To je síce pravda ale hodnota 0 by znamenala, že nebol použitý žiadny PiningSites a CML by sa nachádzal v chaotickom stave, čo by znamenalo že druhá časť funkcie 4 by nemohla byť nulová. Na druhej strane ak by hodnota druhej časti funkcie 4 je nulová tak niektoré PiningSites museli byť použité a teda je vrátená hodnota p1. Z toho vyplýva že pri použitý funkcie 4 nemožno dosiahnuť hodnotu 0.

6 Implementácia

Program je napísaný v objektovo orientovanom jazyku C# s frameworkom .NET 4.6.2. Vývoj prebiehal v prostredí Visual Studio 2015. K vytvoreniu uživateľského rozhrania som bola použita technologia Windows Forms. Jedná sa o prostredie klasickej desktopovej aplikácie. Webová alebo mobilná aplikácia nie je v tomto prípade nutná.

6.1 Reprezentácia CML v jazyku C#

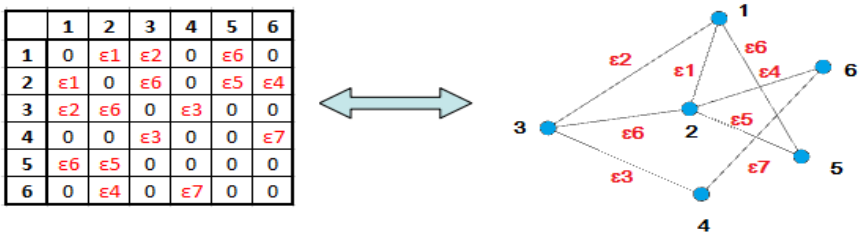
Jedna z dôležitých úloh správny vhodný výber reprezentcie CML. Výber a implementácia vhodnej štruktúry nie je triviálnou záležitosťou, pretože treba uvažovať s viacerými faktorami, ktoré náš budú ovplyvňovať ako je napríklad rozsah dat. Za najlepšiu alternatívu som považoval rozdelenie samostatnej štrúktúry CML na dve časti.

- **Matica vizualizácie:** Táto matica je o veľkosti X*N, kde X je počet rovníc a N je počet iterácií. Nakoľko chceme vizualizovať daný priebeh systému je nutné uložiť všetky vypočítané historické data. Popis počítania matice je v kapitole 3. Výpis 1 znázornuje výpočet aktúálneho stavu pre danú rovnicu CML, kde je vstupom index aktuálnej rovnice a index iterácie v matici vizualizácie. Výpis 2 predstavuje výpočet metodu na výpočet Pining. Čo je nutné poznamenať je fakt že všetky hodnoty v matici sú v rozsahu 0 až 1.

```
public double CalculateState(int node, int iteration, double[,] lattice)
{
 double result=0, counter=0;
 for (int i = 0; i < EpsilonLattice.GetLength(0); i++)
     {
       if (i != node && EpsilonLattice[i, node] != 0)
       {
          result += EpsilonLattice[i, node] *
          lattice[i, iteration].logistic();
          counter++;
       }
     }
 result /= counter;
 result += (1 - EpsilonLattice[node, node]) *
           lattice[node, iteration].logistic();
 return result;
}
```

Výpis 1: Výpočet CML v C#

- **Matica susednosti:** V základnom CML sú väzby len medzi susednými rovnicami a to s konstantnou hodnotou epsilon 0.8 pre celý CML. Avšak tieto väzby a hodnoty epsilon môžu byť odlišné vzhľadom na vstupnú komplexnú sieť. S predpokladom, že sa informácie o epsilon hodnotách spolu s väzbami nahrajú na začiatku a postupne sa už budú len čítať, je najlepšou možnosťou uloženia matica. Najväčšou výhodou matice je jej časová zložitosť čítania O(1). Ukážku uloženia epsilon väzieb je možno vidieť na Obr. 30.



Obr. 30: Matica susednosti v C#

```
private double CalculatePining(int node, int index, double[,] lattice)
{
   if (PiningSites[node] == 0)
      return 0;
   double result = 0, counter = 0;

   for (int i = 0; i < EpsilonLattice.GetLength(0); i++)
   {
      if (i != node && EpsilonLattice[i, node] != 0)
      {
          result += EpsilonLattice[i, node] *(lattice[i, index] *
                    PiningValue[node] * (lattice[i, index] - x0[node]));
          counter++;
      }
   }
   result/= counter;
   result += (1 - EpsilonLattice[node, node]) * (lattice[node, index] *
             PiningValue[node]) * (lattice[node, index] - x0[node]);
   return result;
}
```

Výpis 2: Výpočet PiningSites v C#

6.2 Knižnica Oxyplot

Jedným z problémov bolo samotné vykreslenie CML. Spočiatku som použil naivný spôsob a vykresľovanie som si implementoval sám. Neskôr sa ale vyskytli problémy pri vizualizácii CML a to hlavne pri Real-time režime. Preto som sa rozhodol preskúmať už existujúce knižnice. Po preskúmaní niekoľkých variánt som nakoniec použil knižnicu **Oxyplot**. Jedná sa o *opensource* knižnicu, ktorú je možno použiť na rôznych platformách. Ovládacie prvky sú implementované pre WPF, Windows Phone, Silverlight, Xamarin, Windows Foms,... Výhodou je možnosť prepísania alebo vytvorenia vlastnej funkcionality. Ukážka vykreslenia CML je na výpise kódu 3.

PlotModel tvorí základ pre vykresľovanie. K modelu je možné pridať uživateľom požadované nastavenia osy, vlastné pozadia, názov a mnoho iných nastavení. Súčasťou modelu sú taktiež série. V aplikácii som použil 2 druhy sérií. Pre vykreslenie CML je použitá *HeatMapSeries* , ktorej požadovaný vstup je matica hôdnôt. Táto matica je vyššie zmienená **Matica vizualizácie**, avšak je nutné ju pretransformovať na požadovaný rozsah (OxyPalet-255). Hodnoty z rozsahu 0-1 sa zmenia do rozsahu 0-255. *LineSeries* je použitá pri vykreslení očakávanej a vypočítanej finálnej vlny CML (viz. Obr.15). Tento typ série požaduje ako vstup vektor hodnôt.

```
PlotModel model = new PlotModel { Title = "Static" };
model.Axes.Add(new LinearColorAxis { Palette = OxyPalettes.Gray(255) });
var heatMapSeries = new HeatMapSeries
{
X0 = 0,
X1 = data.GetLength(0),
Y0 = 0,
Y1 = data.GetLength(1),
RenderMethod = HeatMapRenderMethod.Rectangles,
Data = data
};
model.Series.Add(heatMapSeries);
Static.Invoke(new MethodInvoker(delegate { CML.Model = model; }));
```

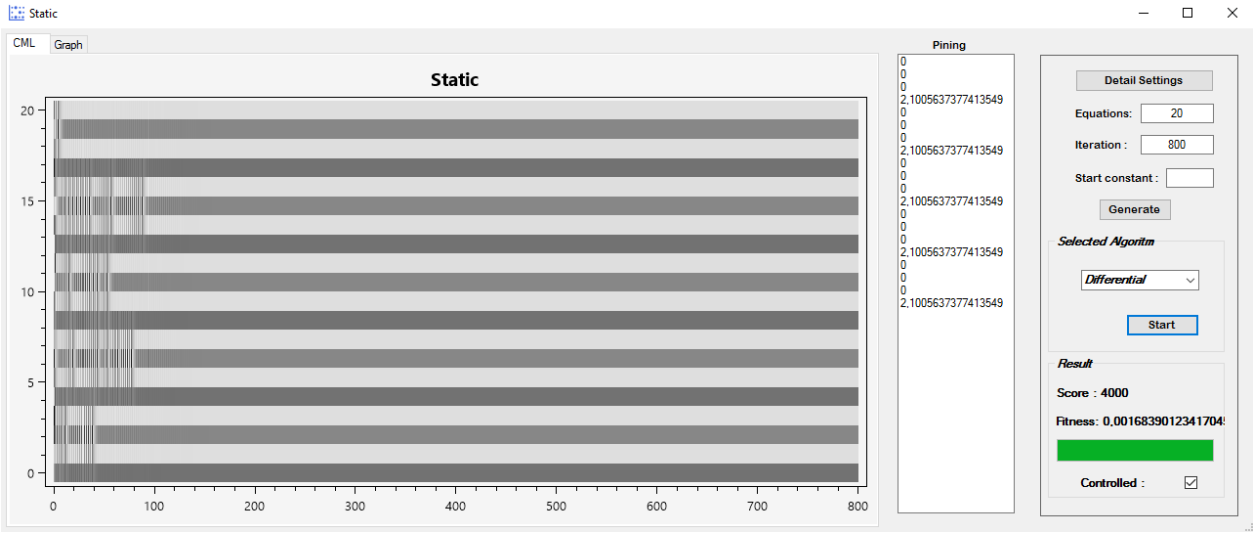
Výpis 3: CML Oxyplot

Jedinou doposial zistenou nevýhodou tejto knižnice je dlhá doba čakania pri vykreslení *HeatMapSeries* o väčšej matici. Pri vykreslení statického CML, kde je očakávaná matica zvyčajne o veľkosti 50*800 nemá knižnica s vykreslením ani interakciou s uživateľom (zvačšenie-zmenšenie okna) žiadny problém. Avšak pri Real-time režime, kde je matica zvyčajne o veľkosti 20*40 000 a vyššie začína vykresľovanie sekať. Tento problém som sa snažil vyriešiť zmenou rendrovacej metody z *Rectangles* na *Bitmap*. Táto zmena sice pomohla pri vykresľovaní ale heatmapa už ne-

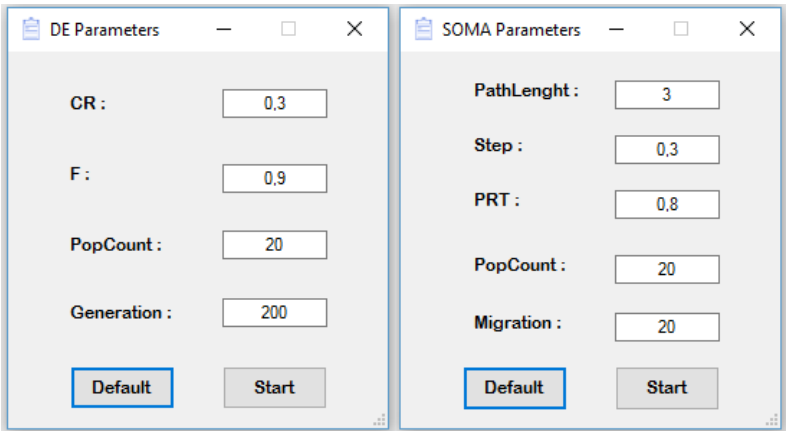
bola tak detailne vykreslená. Z tohto dôvodu som ostal pri verzii *Rectangles* aj keď pri vysokých hodnotách je problém s vykresľovaním.

6.3 Popis Uživateľského rozhrania

Čo sa týka uživateľského rozhrania je pomerne veľmi jednoduché. Po zapnutí aplikácie si užívateľ zvolí v akom režime bude chcieť pracovať (statický alebo real-time). Po vybratí ľubovoľného režimu je rozhranie takmer totožné ako na obr 31. Na ľavej strane v záložke CML je možné vidieť aktuálny stav CML modelu či už riadeného alebo nie. Túto vlastnosť je možno prepínať za pomocí vlastnosti v pravom dolnom rohu s názvom **„Controlled“**. Záložka **Graph** znázorňuje graf, ktorý zobrazuje finálnu vlnu CML zobrazenú na Obr. 16. Tabuľka **Pining** znázorňuje aktuálne hodnoty PiningControl a Pining, kde jednotlivé indexy určujú akou silou sa odpovedajúca rovnica nabudí. Najdôležitejšou časťou je pravý panel. Ako prvé je potrebné vygenerovať neriadené CML a to pomocou **„Generate“**. Vstupom je počet rovníc **„Equations“**, počet iterácií **„Iteration“** a štartovná konštanta **„Start constant“**. Štartovná konštanta predstavuje počiatočný stav CML v rozmedzí 0 až 1. Pre vygenerovanie náhodných počiatočných hodnôt ostáva hodnota prázdna. Po vygenerovaní neriadeného CML je možné aplikovať algoritmus na riadenie systému. Uživateľ má možnosť výberu medzi algoritmami DE a SOMA, pričom po vybratí a kliknutí na **„Start“** sa uživateľovi zobrazí jednu s nasledujúcich možností na obr. 32, kde je možnosť nastavenia primárnych parametrov pre vybraný algoritmus. Po zapnutí vybraného algoritmu môžeme priebeh sledovať v oblasti **„Result“**. **„Score“**, čo predstavuje aktuálny počet ohodnotení a **“Fitnes“** zobrazuje fitnes hodnotu najlepšieho jedinca.



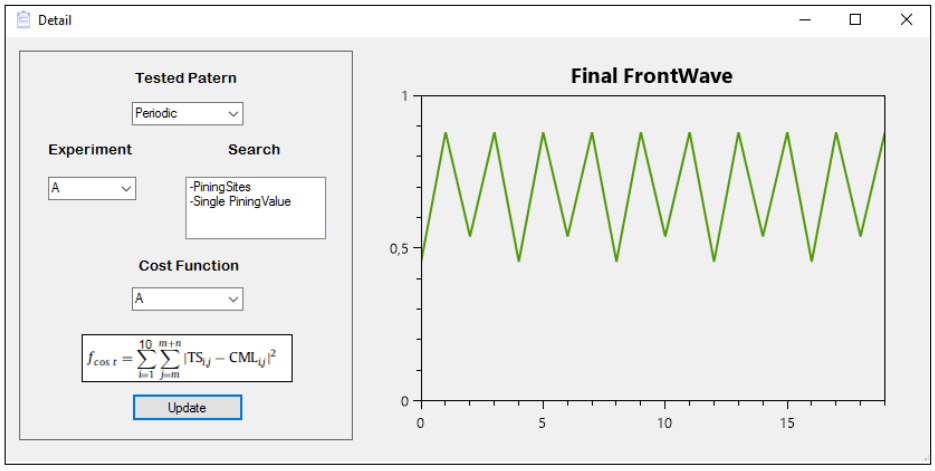
Obr. 31: Real-time režim



Obr. 32: Nastavenie Parametrov

Najdôležitejšou súčasťou je **„Detail Settings“**, ktoré obsahuje základné nastavenia CML systému. Tieto nastavenia je možné vidieť na obr. 33. V tomto nastavení sa na pravej strane nachádza očakávaná finálna vlna vybraného vzoru. Ľavá strana zobrazuje nasledujúce základné parametre potrebné k správnemu riadeniu CML:

- **Testovacie vzory:** aplikácia má naimplementované 2 testovacie vzory. Jedná sa o základný konstantný a periodický vzor, ktoré sú popísané v kapitole 3.3.
- **Typy Experimentov:** nastavenie pri ktorom sa určí či sa jedná o hľadanie PiningControle a jedinej Pining, alebo Pining pre každú rovnicu.
- **Účelová funkcia:** možnosť výberu účelovej funkcie pri hľadaní globálneho minima počas evolúcie. Implementované sú dve funkcie popísané v kapitole 5.3.



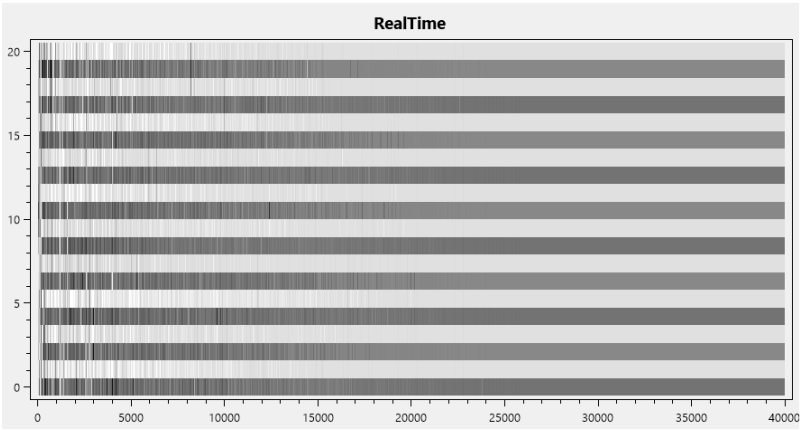
Obr. 33: Detailné nastavenia

6.4 Statický a Real-time režim

Samotné riadenie prebieha pomocou vybraných evolučných algoritmov avšak v dvoch rozdielnych režimoch. Princíp výpočtu ostáva pre obidva režimy. Hlavný rozdiel je pri nastavení počiatočných hodnôt pri evolúcii.

- **Statický režim** znamená, že pri hodnotení sa každý jednotlivec z populácie spustí na CML, ktorého počiatočný stav je stále rovnaký, nemeniaci sa. Príkladom môže byť CML, ktorý má počiatočný stav nastavený na konštantu, napríklad 0.52 pre všetky rovnice a počet iterácií na 800. Pri hodnotení populácie bude každý jednotlivec začínať z rovnakého stavu a to z 0.52 po dobu 800 iterácií, avšak každý s inými hodnotami PiningSites a PiningValue. Ukážku statického riadenia možno vidieť na obr. 16.

- **Real-time režim:** je oproti statickému založený na filozofii z reálneho života - nemožno sa vrátiť do stavu predchádzajúceho alebo počiatočných podmienok. To znamená, že CML musí vychádzať z aktuálnych výpočtov a pokračovať ďalej v čase. Využitie EA procesu t.j. vývojom jednotlivcov v populácii je simulované v CML tak, že určitý počet iterácií slúži na ohodnotenie každého jednotlivca v populácii. Toto číslo odráža časovú medzeru reálneho systému potrebného na spracovanie a ohodnotenie daného vstupu pred prijatým ďalšieho. Znamená to, že ak ku príkladu má populácia 20 jednotlivcov a počet CML iterácií medzi jednotlivými jedincami je 10, potom v prípade 50 generácií (predpoklad nájdenia vhodného riešenia) je celkový počet CML iterácii 10 000 (20*10*50). Z pohľadu jednotlivcov to znamená, že 1 jednotlivec 1 generácie začína na 1 iterácii CML a končí na 1*10 iterácii. Druhý jednotlivec začína na 1*10+1 iterácii a končí na 2*10 iterácii atď. V prípade druhej generácie začína prvý jednotlivec na (20*10+1) t.j. 201 iterácii a končí na (20*10+1*10) t.j. 210 CML iterácii. Týmto dokáže simulovať real-time správanie, avšak treba brať do úvahy, že nepracujeme so žiadnym skutočným real-time modelom ale len so simuláciou.



Obr. 34: Real-time režim

7 Experimentálna časť

Všetky experimenty sa vykonali najskôr v statickom a nasledovne v real-time režime. Na testovanie bol použitý základný CML model popísaný v 3 kapitole. Počet vstupných rovníc CML bolo nastavených na 20. Značenie Experimentu sa skladá z 3 písmen reprezentujúce použitý režim, typ experimentu a použitá účelová funkcia. Tabuľky 2 a 3 zobrazujú základné informácie o jednotlivých experimentoch ako je napríklad použitá účelová funkcia alebo použitý počet iterácií a podobne. Popis jednotlivých písem je nasledovný:

- **S:** Statický režim **R:** Real-time režim
- **A:** Hľadanie PiningControl a jedinej Pining

  B: Hľadanie PiningControl a Pining
- **A:** účelová funkcia (vzorec 6) **B:** účelová funkcia (vzorec 7)

Pri experimentoch bolo dôležité rozhodnúť od akej miery alebo kedy je vlastne test **„považovaný“** za úspešný. Ako rozhodovaciu podmienku som určil nasledovné kritérium. Experiment je považovaní za úspešný v prípade ak PiningControl je v rozmedzí 2.05 - 2.15 a počet Pining je 5 alebo 6.

Tabuľka 2: Experimenty pre Statický režim

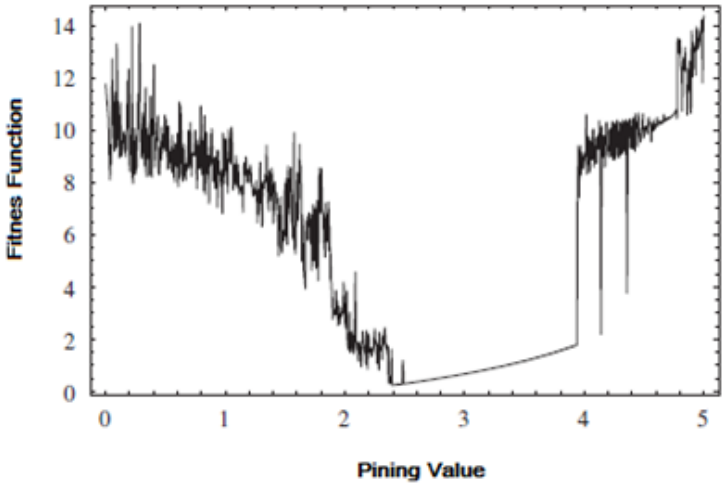
Statický režim			
Experiment	**Iterácie**	**Popis Hľadania**	**Účelová funkcia**
SAA	800	PiningControls a jedna Pining	vzorec (6)
SAB	800	PiningControls a jedna Pining	vzorec (7)
SBA	800	PiningControls a Pinings	vzorec (6)
SBB	800	PiningControls a Pinings	vzorec (7)

Tabuľka 3: Experimenty pre Real-time režim

Real-time režim			
Experiment	**Iterácie**	**Popis Hľadania**	**Účelová funkcia**
RAA	30	PiningControls a jedna Pining	vzorec (6)
RAB	30	PiningControls a jedna Pining	vzorec (7)
RBA	30	PiningControls a Pinings	vzorec (6)
RBB	30	PiningControls a Pinings	vzorec (7)

7.1 Nastavenie parametrov

Hlavným dôvodom prečo sú pre tieto experimenty vybrané EA ukazuje obr.35 zobrazujúci plochu fitnes hodnoty. Vzhľadom na predchádzajúce experimenty [14] je vidieť že plocha pre PiningSites je vysoko nelineárna až nevyspytateľná a preto nie je možné použiť deterministicky založené metódy. Taký typ povrchu je vhodný pre triedu EA. Avšak dôležitým faktorom pre správne fungovanie EA je vhodné nastavenie vstupných parametrov. Existuje niekoľko metód na získanie optimálnych parametrov ale v tomto prípade som sa riadil podľa [15]. Pre obidva testované algoritmy DE a SOMA sú vytvorené dve verzie 1 a 2, ktorých jednotlivé nastavenia parametrov sú zobrazené v tabuľkách 4 a 5. Je treba upozorniť že pre experimenty typu „Hľadanie PiningControl a Pining“ je pri EA použitý vyšší počet generácií či migrácií. Hlavným dôvodom je to, že tieto typy experimentu majú dvojnásobnú dimenziu hľadania ako ostatné. Avšak pri väčšine prípadov nikdy nedošlo k maximálnemu počtu ohodnotení.



Obr. 35: Plocha účelovej funkcie

Tabuľka 4: DE - parametre

Verzia	DE1	DE2
Populácia	20	20
F	0,9	0,5
CR	0,3	0,1
Generácie	200/400#	200/400
Max	4000/8000	4000/8000

Tabuľka 5: SOMA - parametre

Verzia	SOMA1	SOMA2
Populácia	20	20
PRT	0,8	0,2
PathLength	3	3
Step	0,3	0,15
Migrácie	20/40	10/20
Max	4000/8000	4000/8000

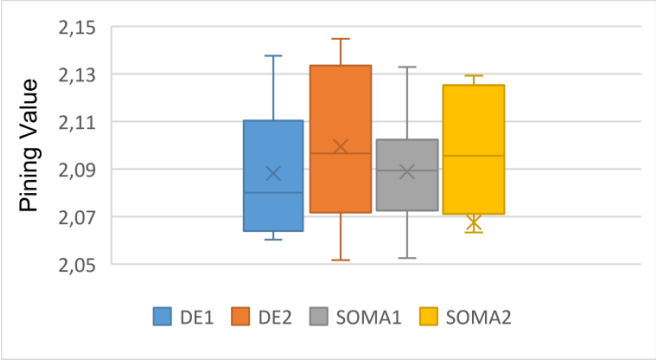
[35] http://ars.els-cdn.com/content/image/1-s2.0-S0952197608001310-gr7.jpg

7.2 Výsledky Experimentov

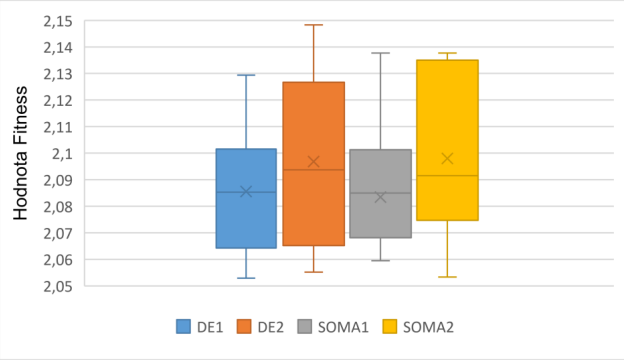
Pre obidva režimy sú vybraté experimenty štyroch typov a každý sa bude testovať na 2 verziách DE a SOMA čo dokopy predstavuje 32 (2*4*2*2) experimentov. Samozrejme každý jeden experiment sa vykonával opakovane a to 20 krát. Všetky experimenty sú zamerané na potvrdenie schopnosti EA riadiť CML. Celkovo sa vykonalo 640 (20*32) testov, pričom každý jeden test začínal v náhodne generovanom stave.

7.2.1 Statický režim

Väčšina experimentov SAA a SAB skončilo takmer úspešne pre všetky prevedené testy. Primárnym účelom obidvoch experimentov bolo hľadanie PiningControl a jedinej Pining, ale z rozdielne použitými účelovými funkciami. Experiment SAA používal vzorec (6) a Experiment SAB vzorec (7). Obrázky Experiment SAA a Experiment SAB zobrazujú škatuľkový graf vypočítaných hodnôt PiningControl pre testované verzie EA. Ako je možné vidieť tieto experimenty dokázali, že všetky 4 verzie sú schopné nájsť správne riešenie, bezohľadnú na použitú účelovú funkciu. Je dôležité podotknúť že Obr. 36 a 37 zobrazujú len „úspešné“ testy pretože neúspešné boli považované za odľahlé pozorovania a nasledovne odstránené.

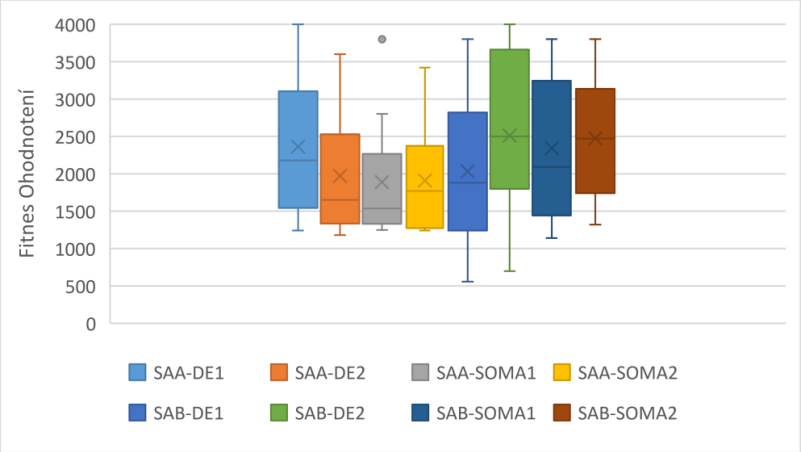


Obr. 36: Experiment SAA

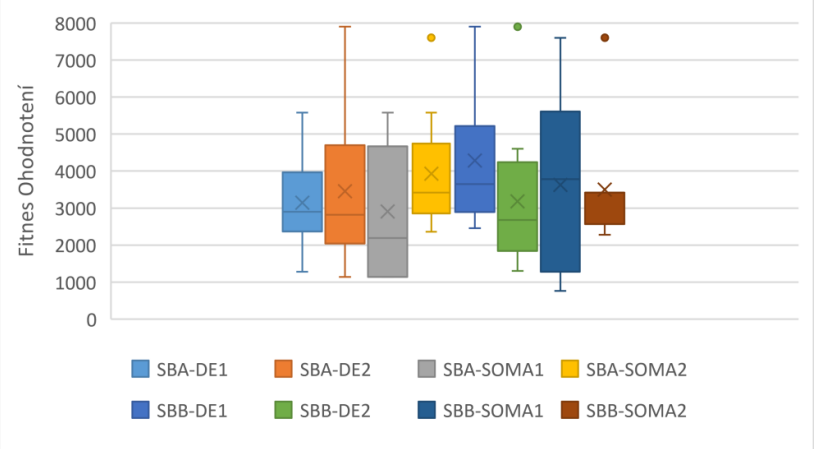


Obr. 37: Experiment SAB

Všetky 4 verzie EA boli aplikované a testované len preto aby našli správnu sekvenciu PiningControl a Pining pre riadenie CML. Na Obr. 38 a 39 možno vidieť výsledky experimentov znázorňujúcich počet potrebných ohodnotení k nájdenie vhodného riešenie pre jednotlivé verzie EA a použitej účelovej funkcie. Experimenty SBA a SBB potrebovali k nájdenie riešenie miestami viacej ohodnotení a to z toho dôvodu pretože hľadali na vyššej dimenzii. Ako môžeme na grafoch vidieť tak väčšina algoritmov nedosiahla maximálny počet iterácií čo nasvedčuje tomu, že tieto testy dopadli úspešne. Samozrejme došlo aj k niekoľkým zlyhaniam, ale len v minimálnom množstve. Tabuľka 6 znázorňuje minimálny, priemerný a maximálny počet ohodnotení účelovej funkcie pre každý experiment v závislosti na type použitého algoritmu.



Obr. 38: Počet ohodnotení pre SAA a SAB



Obr. 39: Počet ohodnotení pre SBA a SBB

Tabuľka 6: Výsledky Experimentov v statickom režime

Experiment SAA

	DE1	DE2	SOMA1	SOMA2
Min	1240	1180	1250	1240
Avg	2362	1970	1890	1909
Max	4000	3600	3380	3420

Experiment SBA

DE1	DE2	SOMA1	SOMA2
1280	1140	1140	2360
3142	3454	2910	3926
5580	7900	5580	7600

Experiment SAB

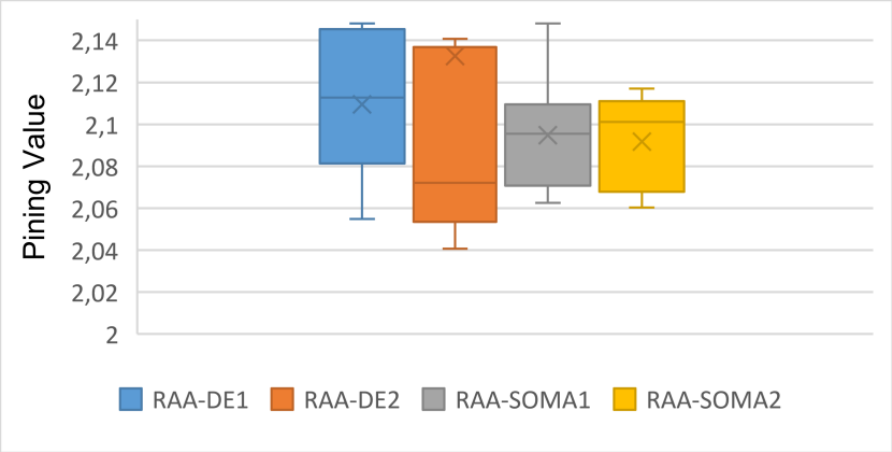
	DE1	DE2	SOMA1	SOMA2
Min	560	700	1140	1320
Avg	2032	2510	2341	2474
Max	3800	4000	3800	3800

Experiment SAB

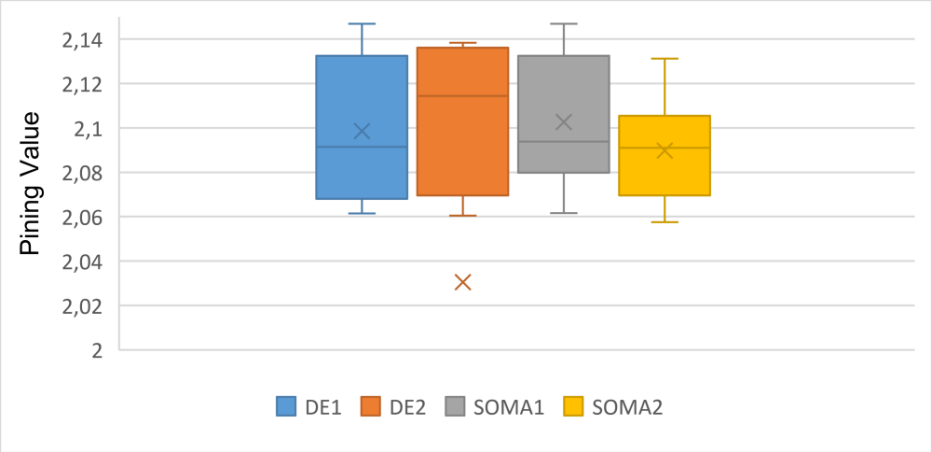
DE1	DE2	SOMA1	SOMA2
2460	1300	760	2280
4286	3178	3622	3496
7900	7900	7600	7600

7.3 Real-time režim

Podobne ako pre statický režim boli vykonané totožné experimenty, ale len v real-time režime. Porovnaním výsledkov zo statickými testami experimentov pre hľadanie PiningControls a jedinej Pining je možno vidieť takmer porovnateľné výsledky. Škatuľkové grafy na Obr. 40 a 41 obsahujú výsledky hľadanej hodnoty PiningValue. Prevažná väčšina hodnôt PiningControls sa pohybuje okolo hľadanej hodnoty 2.1, ale pre lepšiu vizualizácia boli odľahlé pozorovania (neúspešné testy) odstránené. Tieto výsledky dokazujú, že nezáleží na vybranom režime.



Obr. 40: Experiment RAA



Obr. 41: Experiment RAB

Výsledky jednotlivých testov v real-time režime sú zobrazené v Tabuľke 7. Obsahujú informácie o minimálnom, priemernom a maximálnom počte fitnes ohodnotení potrebného k nájdeniu optimálneho riešenia. Priemernej hodnote sa neprikladá príliš veľký význam, jedná sa len o čisto štatistickú hodnotu. Obr. 42 a 43 prestavujú grafické znázornenie výsledkov v závislosti na verzii EA. Obdobne experimenty RBA a RBB majú vyšší počet fitnes ohodnotení vzhľadom na ich vyššiu dimenziu hľadania. Ako môžeme vidieť niektoré algoritmy dopadli úspešne pre všetky prevedené testy tj. nedosiahli nikdy maximálny Fce ohodnotení.

Tabuľka 7: Výsledky Experimentov v real-time režime

Experiment RAA

	DE1	**DE2**	**SOMA1**	**SOMA2**
Min	960	800	780	780
Avg	1898	2402	2388	2456
Max	4000	4000	3620	3800

Experiment RBA

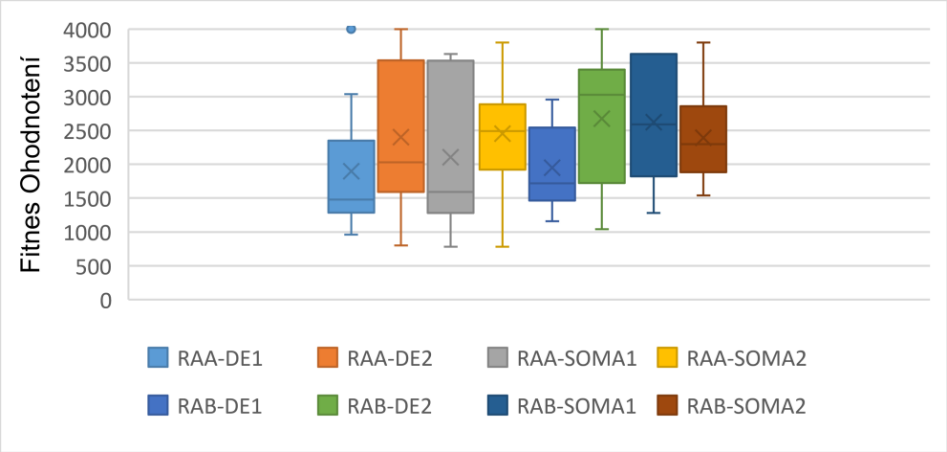
DE1	**DE2**	**SOMA1**	**SOMA2**
1360	2900	970	1800
4377	4380	3100	4307
8000	8000	7430	6050

Experiment RAB

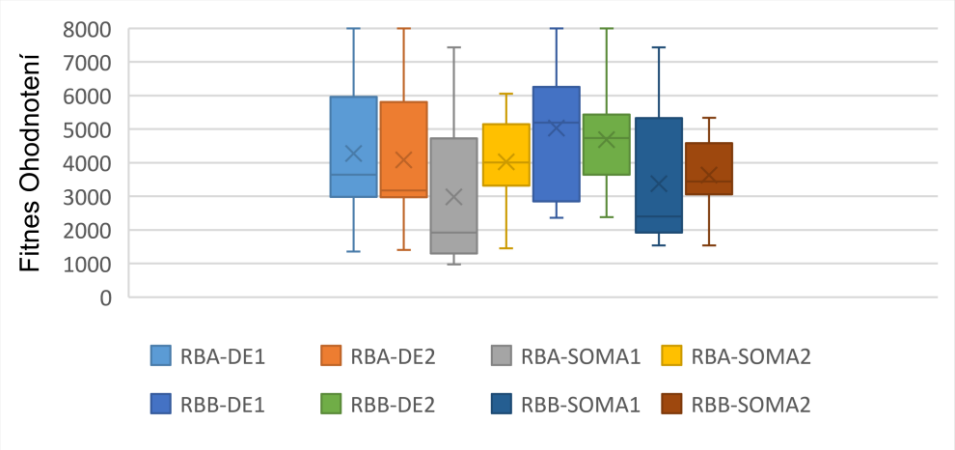
	DE1	**DE2**	**SOMA1**	**SOMA2**
Min	1160	1040	1280	1540
Avg	1950	2676	2624	2394
Max	2960	4000	3630	3800

Experiment RBB

DE1	**DE2**	**SOMA1**	**SOMA2**
2360	2380	1540	1540
5046	4320	3511	3440
8000	5520	7430	4580



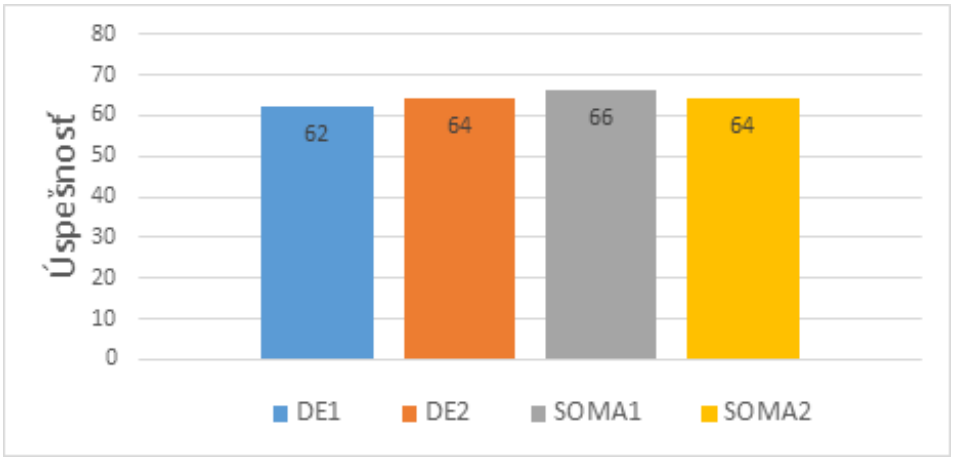
Obr. 42: Počet ohodnotení pre RAA a RAB



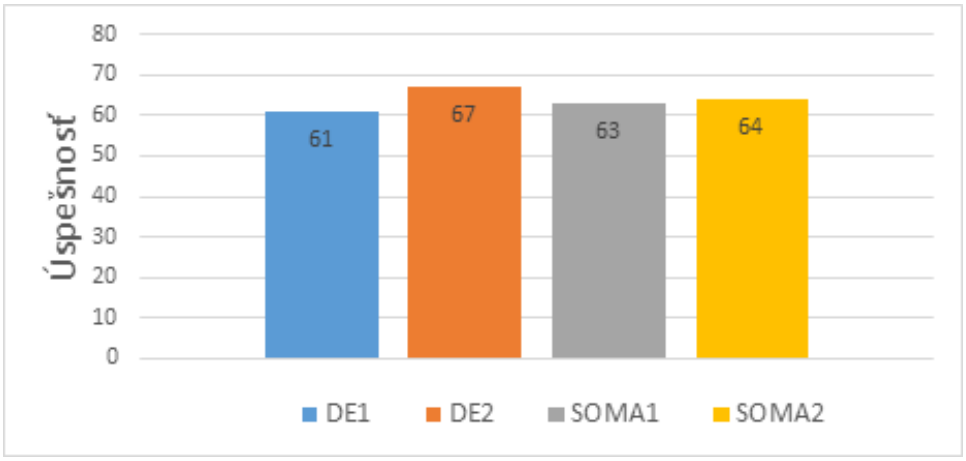
Obr. 43: Počet ohodnotení pre RBA a RBB

7.4 Zhodnotenie výsledkov

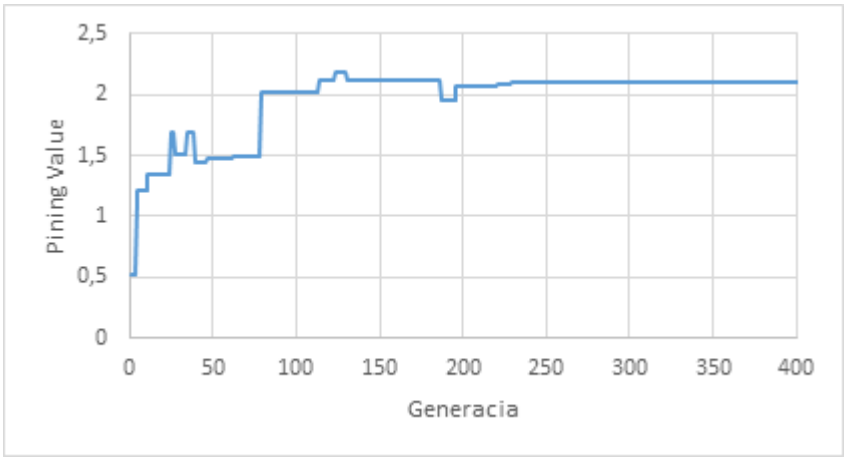
Každá zo 4 verzií algoritmov (DE1, DE2, SOMA1, SOMA2) bola aplikovaná 160 krát pri odlišných podmienkach (viz kapitola 6.2). Prvá polovica testov prebiehala v statickom a druhá v real-time režime. Primárny účel nebol ukázať aká verzia je najlepšia, ale že EA dokážu vyriešiť časopriestorové riadenie chaosu na CML systéme, čo sa na základne výsledkov zo statického režimu (viz Obr. 36-39) a real-time režimu (viz Obr. 40-43) potvrdilo. Typický postup hľadania najlepšieho jednotlivca je možné vidiť na Obr. 46. Už na začiatku 100 generácii bola EA schopná nájsť adekvátne riešenie. Po dokončení cca 250 generácií sa ustálilo na konštantnej hodnote v blízkosti 2.1. Z celkového pohľadu sa ukázalo , že EA sú naozaj schopné vyriešiť aj takýto druh problému, no samozrejme nie vždy. Obr. 44 a 45 znázorňujú grafy úspešnosti jednotlivých algoritmov v použitých režimoch. Ako je možné vidieť vybratý režim nespôsobuje žiadne výrazné rozdiely vo výsledkoch. Vo všetkých experimentoch je úspešnosť EA približne 76% až 86%.



Obr. 44: Výsledky EA v statickom režime



Obr. 45: Výsledky EA v real-time režime



Obr. 46: Hľadanie Pining Value

8 Závěr

Hlavným účelom tejto práce bolo poukázať na možnosť riadenia komplexných sietí pomocou CML systému. Bolo vysvetlené ako je možné previesť ľubovoľnú komplexnú sieť na CML a nasledovne ju analyzovať alebo riadiť pomocou vybraných technik.

Súčasťou riešenia bolo vytvorenie uživateľskej aplikácie na vizualizáciu CML a aplikovanie evolučných algoritmov na základný CML systém. Výsledná aplikácie umožnuje nájsť také zásahy, ktoré dokážu dostať systém do očakávaného periodického alebo konstantného stavu.

Experimentálna časť potvrdila, že EA sú naozaj schopné riadiť chaotické systémy. Zo všetkých prevedených 32 experimentov ktoré boli opakované 20 krát bolo preukázaných len pár zlyhaní (viz Obr. 44, 45). To len dokazuje robustnosť EA. Čo sa týka zrovnávanie jednotlivých použitých verzií tak nemožno určiť najlepšiu. Všetky dosahovali celkom dobré výsledky. V niektorých prípadoch horšie, v iných zase lepšie. Nastavenia parametrov bolo na základe heuristického prístupu, ale pripúšťame možnosť, že existuje aj oveľa lepšie nastavenie.

Ako bolo načrtnuté v tejto práci, populačná dynamika sa dá previesť na komplexnú sieť. V každom evolučnom kole sa sieť mení a preto sa stáva dynamickou. Na konci každej generácie/migrácie je možné túto sieť zmeniť na CML systém a následovne riadiť ako bolo názorne ukázané. Výsledkom by boli také zmeny do systému , ktoré by zlepšili dynamiku jednotlivcov v dalšej generácii. Touto víziou by sa mohla zaoberať dizertačná práca.

Literatúra

[1] LORENZ, Edward N. Deterministic nonperiodic flow. Journal of the atmospheric sciences, 1963, 20.2: 130-141.

[2] Schuster,H.:Handbook of Chaos Control.Wiley-VCH,NewYork(1999) Wiley-Interscience, New York (2002)

[3] Ott E, Greboki C, Yorke JA. Controlling chaos. Phys Rev Lett 1990;64:1196–9.

[4] GLEICK, James. Chaos: vznik nové vědy. Ando Publishing, 1996. 350s. ISBN 80- 86047-04-0.

[5] Edward Norton Lorenz. Bibliografia 2013 [online] [cit. 2017-07-01]. Dostupné z: http://www-history.mcs.st-andrews.ac.uk/Biographies/Lorenz_Edward.html

[6] Tisnovsky Pavel. Nelineární dynamické systémy, 1999 [online]. [cit. 2017-07-01]. Dostupné z: http://www.fit.vutbr.cz/ tisnovpa/publikace/diplomka/doc/node29.html

[7] Ivan Zelinka. Chaos control 2003 [online]. [cit. 2017-07-01]. Dostupné z: http://www.ft.utb.cz/people/zelinka/chaos/Chaos_control.html

[8] Kaneko, Kunihiko. Överview of coupled map lattices."Chaos: An Interdisciplinary Journal of Nonlinear Science 2.3 (1992): 279-282.

[9] Kaneko, Kunihiko. "Period-Doubling of Kink-Antikink Patterns, Quasiperiodicity in Antiferro-Like Structures and Spatial Intermittency in Coupled Logistic Lattice Towards a Prelude of a “Field Theory of Chaos”."Progress of Theoretical Physics 72.3 (1984): 480-486.

[10] Petr Kovár. Úvod do teorie grafu, 2012. [online] [cit. 2017-07-01]. Dostupné z: http://www.soubor.eu/zcu/FAV-bc/KMA/Uvod%20do%20teorie%20grafu.pdf

[11] Kvasnička V., Pospíchal J., Tiňo P. 2000, Evolučné algoritmy, STU Bratislava, ISBN 85-246-2000, 2000

[12] Fleetwood, Kelly. Än introduction to differential evolution."Proceedings of Mathematics and Statistics of Complex Systems (MASCOS) One Day Symposium, 26th November, Brisbane, Australia. 2004.

[13] Zelinka, Ivan. "SOMA—self-organizing migrating algorithm."New optimization techniques in engineering. Springer Berlin Heidelberg, 2004. 167-217.

[14] Senkerik, Roman, Ivan Zelinka, and Eduard Navratil. Ïnvestigation on evolutionary EDTAS chaos control."proc. 20th European Conference on modelling and simulation ECMS. 2006.

[15] Kuznetsov, S. P., and A. S. Pikovskii. Üniversality of period-doubling bifurcations in one-dimensional dissipative media."Radiofizika 28 (1985): 308-319.

[16] Zelinka, Ivan, Donald David Davendra, Mohammed Chadli, Roman Senkerik, Tran Trong Dao, and Lenka Skanderov. "Evolutionary dynamics as the structure of complex networks." In Handbook of Optimization, pp. 215-243. Springer Berlin Heidelberg, 2013.

[17] Zelinka I., Davendra D., enkek R., Jaek R., Do Evolutionary Algorithm Dynamics Create Complex Network Structures? Complex Systems, 2, 0891-2513, 20, 127-140

[18] Bakalárska práca: Sirotiar, Matej. "Vizualizace chaotických atraktorů a jejich vlastností."(2015).

[19] SHAN, Liang, et al. Chaotic optimization algorithm based on Tent map. Control and Decision, 2005, 20.2: 179-182.

[20] HABUTSU, Toshiki, et al. A secret key cryptosystem by iterating a chaotic map. In: Advances in Cryptology—EUROCRYPT'91. Springer Berlin Heidelberg, 1991. p. 127-140.

[21] GRIFFIN, Jory. The Sine Map [online]. 2013 [cit. 2017-07-15] Dostupné z: https://people.maths.bris.ac.uk/ macpd/ads/sine.pdf

[22] Ivan Zelinka and Guanrong Chen. Evolutionary Algorithms, Swarm Dynamics and Complex Networks, ISBN 3662556618, 2017

A Príloha na CD/DVD

Adresárová štrúktúra priloženého CD:

Adresár	**Popis**
/src	Zdrojový kód aplikácie
/doc	Text diplomovej práce