UNIVERZITA PALACKÉHO V OLOMOUCI
PŘÍRODOVĚDECKÁ FAKULTA
KATEDRA MATEMATICKÉ ANALÝZY A APLIKACÍ MATEMATIKY

DIPLOMOVÁ PRÁCE

Biologicky inspirované algoritmy

Vedoucí diplomové práce:
Mgr. Jaroslav Marek, Ph.D.
Rok odevzdání: 2010

Vypracovala:
Lucie Koděrová
AME, II. ročník

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracovala samostatně pod vedením pana Mgr. Jaroslava Marka, Ph.D. s použitím uvedené literatury.

V Olomouci dne 7. dubna 2010

Poděkování

Na tomto místě bych chtěla poděkovat svému vedoucímu diplomové práce panu Mgr. Jaroslavu Markovi Ph.D. Také bych ráda poděkovala své rodině a přátelům, že mě po celou dobu studia podporovali.

Obsah

Abstrakt:
V práci jsme se zaměřili na studium biologicky inspirovaných algoritmů z oblasti neuronových sítí a optimalizačních algoritmů pro úlohy s podmínkami, bez podmínek a kombinatorické problémy. Po analýze principů fungování neuronových sítí byly vytvořeny aplikace sítí vhodných pro shlukovou analýzu. Heuristiky jsou netradiční metody založené převážně na iteračním zlepšování řešení. Diferenciální evoluci (DE) řadíme mezi heuristické algoritmy pro optimalizaci bez podmínek. Aplikaci DE tvoří vlastní algoritmus Rypoš. Studovali jsme také zobecněnou verzi DE vhodnou pro optimalizaci s podmínkami. Sestrojena a otestována byla alternativa této metody, algoritmus Complex Method. Mezi deterministicky obtížně řešitelné problémy řadíme kombinatorickou úlohu obchodního cestujícího (TSP). Pro nalezení aproximace TSP byl použit heuristický algoritmus Mravenčích kolonií. Úspěšnost algoritmů byla vyhodnocena dle zpracovaných výsledků testování.
Klíčová slova: neuronové sítě, diferenciální evoluce, Mravenčí kolonie.

Abstract:
In the present paper we focused on the study of biology–inspired algorithms from the field of neural networks and algorithms for constrained and unconstrained optimization and combinatorial problems. After analysis of principles of neural networks, there are developed applications of neural networks suitable for cluster analysis. Heuristic methods are based on iterative improvement of solutions. The Differential Evolution (DE) rank among the heuristic algorithms for unconstrained optimization. The DE is applied in an own algorithm Rypoš. We also studied a generalized version of DE suitable for constrained optimization. There was constructed and tested alternative to this method, the algorithm Complex Method. Finding of the deterministic solution of the combinatorial Traveling Salesman Problem (TSP) is very difficult. To find the approximation of TSP we used heuristic Ant Colony algorithm. The succes of applied algorithms was evaluated by processing the results.
Keywords: neural networks, differential evolution, Ant Colony algorithm.

Cíle práce

Cílem práce je prostudovat problematiku biologicky inspirovaných algoritmů a tyto poznatky využít při vytváření aplikací některých z nich. Budeme se zabývat netradičními metodami pro řešení úloh shlukové analýzy, optimalizací bez podmínek i s podmínkami a kombinatorickou optimalizací. Tyto metody jsou vhodné pro deterministicky neřešitelné problémy a proto je účelem práce provést experimenty, který by vedly k ohodnocení úspěšnosti zvolených algoritmů. Moderní biologický algoritmus neuronových sítí použijeme pro shlukovou analýzu. Záměrem je sestavit dva neuronové algoritmy a pomocí nich redukovat datovou strukturu. Aplikací na stejný datový soubor získáme možnost vyhodnotit a srovnat úspěšnost jednotlivých algoritmů.

Pomocí algoritmu diferenciální evoluce (DE) budeme hledat globální minimum vícekriteriálních funkcí. Jedná se o optimalizaci bez podmínek. Cílem je na základě nastudované teorie o DE sestrojit vlastní algoritmus a vyhodnotit jeho úspěšnost pomocí známých testovacích funkcí.

Teoretické poznatky o DE budeme rozvíjet pro případ optimalizace s vazbami. Pro zobecnění DE budou použita kritéria využívající podmíněné dominance. Zpracování této látky vyvrcholí v popisu, jak je možné upravit algoritmus založený na DE i pro úlohy s podmínkami. Heuristické postupy budou doplněny implementací algoritmu Complex Method. Tento algoritmus bude použit pro vyhodnocení experimentu, hledání globálního minima vícekriteriální testovací funkce při splnění přípustných podmínek, a získání základních statistických charakteristik. Výsledky tohoto pokusu mohou posloužit v budoucím výzkumu pro porovnání úspěšnosti například se zobecněným algoritmem DE.

Z oblasti kombinatorické optimalizace je zvolena úloha obchodního cestujícího (TSP), jejíž výpočetní a časová náročnost je obrovská. Tato úloha je neustále studována a rozvíjena. Probíhají například i soutěže, které vypisují finanční odměnu za vyřešení určité modifikace úlohy TSP. Cílem této sekce bylo předložit heuristickou a biologicky inspirovanou metodu, která je svou strukturou vhodná pro řešení právě tohoto problému.

Úvod

Náš každodenní život je propojen s elektronickými systémy, jen těžko si umíme představit život bez počítačů, internetu a podobných vynálezů. Většina důležitých procesů a operací, se kterými se setkáváme, je zpracována elektronicky prostřednictvím výpočetních systémů. Používáme je nejen pro provedení nejrůznějších výpočtů, ale často za nás mohou také rozhodovat. Přesto bychom neměli zapomínat, jak bohatý zdroj informací představuje příroda. Stále se totiž setkáváme s úlohami a situacemi, které nejsou matematicky výpočetně zvládnutelné, nebo jejich zpracování není efektivní. Vědci si všimli, že v biologické říši existují analogie těchto problémů a živé organizmy jsou upraveny tak, aby se s nimi poměrně lehce vypořádaly. To vedlo ke zjištění, že příroda může být dobrou učebnicí, jak takové úlohy řešit umělou cestou.

Cílem diplomové práce bude nastudovat a objasnit principy přírodou inspirovaných algoritmů. Zvolili jsme čtyři různé zástupce – neuronové sítě, algoritmy pro optimalizaci s podmínkou i bez podmínky a algoritmus Mravenčích kolonií.

Člověk jako biologický počítač je nejzajímavější výpočetní jednotkou. Principy zpracování informací, usuzování, rozhodování a učení vedoucí k výborným výsledkům se vědci snažili implementovat i do umělých systémů. Vznikla tak oblast neuronových sítí, ve které v současné době dochází k velkému rozvoji. Tato metoda přestala být pouze teoretickou a v určitých oblastech je dnes již běžně aplikovaná. Protože však doposud není k dispozici volně přístupný software, rozhodli jsme se v rámci této práce popsat princip neuronových sítí a sestrojit aplikaci vybraných algoritmů. V první kapitole jsme se zaměřili na algoritmy určené pro shlukovou analýzu – Kohonenovy samoorganizující se mapy a Neuronový plyn, které mohou být použity jako alternativní postupy k metodám nejbližšího a nejvzdálenějšího souseda, centroidní metodě, nebo metodě K-průměrů (např. [1]). Následně byla srovnána úspěšnost vytvořených algoritmů při shlukování Fisherovy datové množiny irisů [10] (podkapitola 1.6.3.).

Druhá kapitola je věnována stochastickým optimalizacím. Budeme se zabývat vývojem populací živočichů. V podkapitole 2.1. je představena diferenciální evo-

luce (DE) [11] jako zástupce heuristických algoritmů pro optimalizaci bez podmínek. Výhodou těchto algoritmů je schopnost nalézt aproximaci řešení i v úlohách, kdy všechny běžně používané postupy selhávají. Algoritmus DE je založen na sestavování nových populací jedinců na základě tří pravidel – křížení, mutace a selekce. Jako aplikaci uvádíme v podkapitole 2.1.2. vlastní algoritmus Rypoš, který byl vytvořen v rámci mé bakalářské práce na téma Heuristiky [11]. Jedná se o australského hlodavce žijícího v koloniích s uspořádáním podobném hmyzu. Účelem algoritmu bylo hledání aproximace globálního minima. Úspěšnost algoritmu byla vyhodnocena pomocí tzv. testovacích funkcí [2], pro které je hodnota globálního minima známá. Heuristické algoritmy jsou účinným nástrojem, který dokáže poskytnout řešení klasickými postupy neřešitelných úloh. Tato skutečnost byla jedním z motivů pro výběr tématu.

Příroda se obecně dělí na živou a neživou. Neuronové sítě i Rypoše řadíme do přírody živé. Zástupcem druhé skupiny je potom algoritmus Complex Method (ACM) [13], který je zobecněním Nelder-Meadovy simplexové metody [12] a řeší otázku optimalizace s vazbami (podkapitola 2.2.2.). V průběhu algoritmu se vytváří diamantové struktury (simplexy), jejichž těžiště konverguje k nalezené aproximaci. Spolu s algoritmem ACM doplňují oblast optimalizace s vazbami alternativní heuristická kritéria, jejichž propojením s algoritmem DE vzniká zobecněná verze vhodná pro podmíněnou optimalizaci. Kritéria GDE1, GDE2 a GDE3 [14] jsou založena na podmíněné dominanci. Na rozdíl od optimalizačních metod bez podmínek, kde existuje velké množství numerických i heuristických algoritmů, je pro úlohy s podmínkami hledání použitelných metod velice obtížné.

Kapitola 2.3. je věnována kombinatorické optimalizaci. Zabýváme se složitým a klasickými metodami v přijatelném čase neřešitelným problémem, úlohou obchodního cestujícího (TSP) [18]. Jedná se o problém, kdy hledáme minimální cestu mezi městy tak, abychom každým prošli právě jednou a na závěr se ocitli ve výchozím bodě. V přírodě byla objevena analogie u mravenčích kolonií. Mravenci vybíhají z hnízda ke zdroji potravy a zpět po přípustných trasách. Cestou uvolňují do prostředí feromon, který slouží pro komunikaci s ostatními jedinci.

Na základě pozorování byl sestaven algoritmus Mravenčích kolonií (ACO) [15]. Podobnost ACO s živou mravenčí kolonií je vysoká. Umělý mravenec využívá informace od ostatních průzkumníků zakódované ve feromonové stopě, rozhoduje se na základě pravděpodobnosti a je schopen se učit.

Existuje samozřejmě řada dalších algoritmů založených na podobnosti s nějakým živočišným druhem, například se včelami (Honey Bee mating optimization algorithm, [22]), vlky (SOMA- Self organizing migrating algorithm, [2]) a jinými zástupci živočišné říše. Pro účely této práce byly záměrně vybrány algoritmy s aplikací ve čtyřech rozdílných oblastech, abychom mohli demonstrovat univerzalitu poznatků získaných z přírody. Zpracované metody jsou aktuální a zajímavé. Tato práce poskytuje alternativní metody použitelné i v mezních a velmi složitých případech. Vytváří také prostor pro další studium a rozvoj této tématiky. Pro aplikaci uvedené teorie byly v programu Matlab vytvořeny funkční algoritmy, které slouží nejen pro větší pochopení problematiky, ale také pro otestování účinnosti metod na zvolených příkladech.

1. Neuronové sítě v úlohách shlukové analýzy

V dnešní době je počítač neodmyslitelnou součástí každodenního života. Usnadňuje rozmanité výpočetní operace a řeší za uživatele úlohy, se kterými by si samostatně jen těžko poradili.

Ještě před tím, než byly vynalezeny počítače, existovala celá řada výpočetních zařízení. Ve své podstatě můžeme chápat také člověka jako „biologický počítač“ zpracovávající pomocí mozku a nervové soustavy výkonně a rychle nepřeberné množství informací. Klasické počítače potřebují pro zpracování některých zjednodušených úloh velké množství času. V tomto směru je mozek nedosažitelným konkurentem. Proto se věnuje velká pozornost pokusům o vytvoření umělého systému napodobujícího činnost nervové soustavy, který by dosahoval srovnatelných výsledků.

Za tvůrce prvního umělého neuronu jsou považováni McCulloch a Pitts, kteří v článku [4] z roku 1943 představili zjednodušený model umělého neuronu. Tento model měl však svá omezení a nedisponoval schopností učit se. Na tuto jednoduchou verzi navázal v 60. letech 20. století F. Rosenblatt, autor Perceptronu, umělého neuronu se schopností adaptace [5].

K opravdovému zlomu došlo v 80. letech 20. století v souvislosti s prudkým rozvojem výpočetní techniky a současným poklesem cen. Výzkum byl zaměřen na otázky týkající se extrakce charakteristických rysů z obrazů, klasifikaci a řešení rozhodovacích otázek v reálném čase. Ukázalo se, že pro řešení těchto úloh tradičními prostředky neexistuje obecný algoritmus. Přitom přístup pomocí neuronových sítí umožňoval řešit celé třídy úloh na základě několika zákonitostí.

Dnes již existuje mnoho modelů neuronových sítí, některé mají navíc schopnost rychle se adaptovat - naučit na danou situaci. Účelem těchto metod je především rozpoznávat, klasifikovat, zobecňovat, optimalizovat a predikovat.

Úspěšná aplikace je možná díky [1]:

Nelinearitě neuronových sítí – Doposud se využívalo linearizace modelu, pro kterou byly známy optimalizační strategie. Při silné nelinearitě však dochá-

zelo k chybným výsledkům a velkým nepřesnostem.

Zvládnutí problému dimenzionality – Problém s dimenzionalitou vznikal při zadávání nelineárních funkcí s velkým počtem proměnných.

Snadné aplikaci – Pokud uživatel ví, jak správně vybrat a předpřipravit data a jak vybrat vhodnou síť, nemusí již nic dalšího programovat.

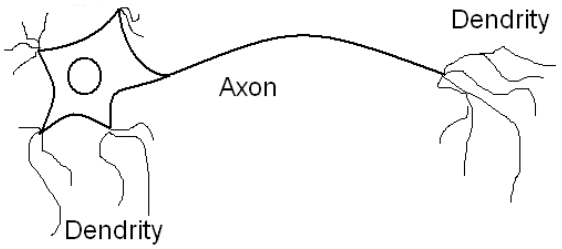
Přirozenému paralelizmu – Lze použít paralelních výpočtů, což je výpočetně efektivní.

Inkrementálnímu učení – Schopnost pracovat s lokálními daty je výhodou při načítání velkých souborů dat.

1.1. Biologická inspirace

Umělá neuronová síť je založena na analogii s nervovou soustavou živých organizmů. Cílem je vytvořit systém napodobující lidskou inteligenci.

Základní stavební jednotkou biologické nervové soustavy je neuron, specializovaná buňka, která má schopnost zpracovávat a šířit elektrické a chemické signály. Lidský mozek má přibližně deset miliard silně propojených neuronů. Do každého neuronu vbíhá velké množství vstupů a po průchodu buňkou jsou přeměněny na jeden výstup. Vstupní struktura je tvořena dendrity, výstupní axony. Axon výchozí buňky se připojuje na dendrity ostatních buněk pomocí synapsí. Synaptický spoj obsahuje přechod s chemickými neurotransmitery připravenými přenést signál. Zajímavostí je, že délka všech axonů přesahuje u jednoho člověka více než čtyřikrát vzdálenost Země - Měsíc.



Obr. 1.1. Schéma biologického neuronu.

Princip fungování neuronu

Neuron reaguje na velikost podráždění na svých vstupech a stane se aktivním, pouze pokud celkový součet všech signálů obdržených z dendritů překročí jistou úroveň (aktivační práh). Tuto prahovou hodnotu má tělo každého neuronu. Můžeme to chápat tak, že pod ní má na svém výstupu např. logickou nulu a nad touto úrovní logickou jedničku.

V případě, že je neuron aktivován, potom vysílá z axonu elektrický signál. Tento signál se šíří synapsemi k ostatním neuronům, které se mohou také stát aktivními. Síla signálu přicházejícího k neuronu přes synapsi závisí především na síle synaptického spoje. Toto spojení je možné posilovat trénováním - učením.

Fakt, že se živé organismy vybavené neuronovou sítí dovedou chovat adaptivně, je dán schopností učit se, dělat závěry ze zkušeností. Donald Hebb, hlavní osobnost zabývající se nervovou soustavou, přišel s poznatkem, že učení je založeno na proměnlivé síle synaptického spoje.

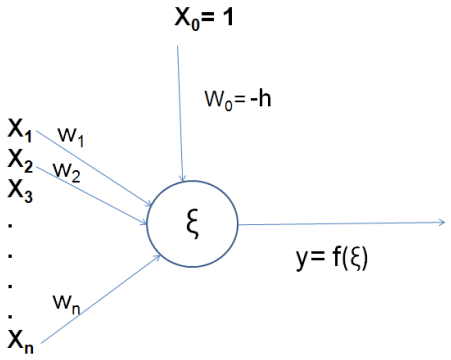
„Jestliže je axon buňky A dostatečně blízko, aby excitoval buňku B, a jestliže se trvale a opakovaně podílí na jejím „pálení“, potom vlivem růstového procesu dochází k metabolickým změnám v jedné nebo obou buňkách tak, že je zvýšena účinnost buňky A, jedné z buněk excitujících B.“ [6]

Příklad: Při Pavlovově experimentu došlo k vytvoření podmíněného reflexu na zvuk zvonku, který byl psům pouštěn vždy v souvislosti s podáním jídla. Hebb vysvětluje vznik tohoto reflexu schopností asociovat zvuk zvonku s příjmem jídla a následným posílením synaptických spojů v oblasti sluchu a slinných žláz. Díky velké intenzitě těchto spojů stačí k aktivaci slinění pouhé zazvonění.

1.2. Model umělého neuronu

Z předchozího textu je zřejmé, že neuron, který načítá vstupy a pomocí synapsí a prahové funkce je šíří dál, je v podstatě jednoduché zařízení. Přesto propojením neuronů vznikne síťová struktura schopná dosáhnout značných výpočetních úspěchů.

Umělý neuron přesně zachovává výše navrženou strukturu. Obsahuje několik vstupů (dendrity), které jsou ohodnoceny vahami, a jeden výstup (axon). V neuronu probíhají dva procesy – výpočet postsynaptického potenciálu a výpočet hodnoty výstupu pomocí aktivační funkce. Ta je ve formě nelineární funkce, nejčastěji sigmoidy. Postsynaptický potenciál je roven součtu všech vstupních signálů násobených příslušnými vahami. Překročí-li hodnota postsynaptického potenciálu prahovou hodnotu, neuron se excituje. Prahová funkce nabude logické jedničky. Výstupní signál se dále beze změny šíří k ostatním neuronům pomocí synaptických spojů. Výsledná hodnota je zároveň výstupem z aktuálního neuronu a vstupem do ostatních neuronů. Pokud vznikne potřeba negovat či zesilovat signál, dochází k tomu právě na přechodu mezi axonem a dendritem - v synapsi [1].



Obr. 1.2. Model umělého neuronu.

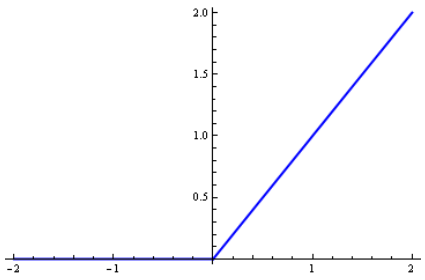
x_i pro $i = 1, 2, \ldots, N$ představují na obrázku příchozí signály od ostatních neuronů. Síla synaptického spoje je dána vahami w_i. Součet vážených signálů tvoří postsynaptický potenciál ξ. Ten vstupuje jako argument do aktivační funkce f. Aktivační funkce nabývá pro binární model hodnot 0 nebo 1, obecně však také -1 a 1.

Toto pojetí platí pro diskrétní model, kdy vstupy i váhy jsou celočíselné hodnoty. Obecně lze však model vnímat jako spojitý. Vstupy, váhy i výstupy mohou být reálná čísla a prahovou funkci potom považujeme za spojitou, omezenou, nelineární a neklesající. Důležitý je požadavek na monotónnost funkce, aby přiřazení výstupu bylo jednoznačné.

Typů přenosových funkcí je mnoho, následující však patří k nejpoužívanějším, jejich grafy byly vytvořeny v programu Mathematica.

Lineární - použita v první Rosenblattově síti [5], řeší pouze lineárně separabilní problémy.

$$\mathbf{F}(\mathbf{P}) = \mathbf{x} \quad \forall \mathbf{x} > 0, \text{ jinak } \mathbf{x} = 0.$$

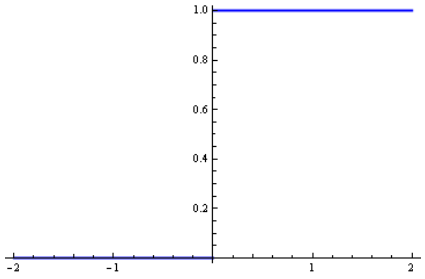


Obr. 1.3. Lineární funkce.

Perceptron se používá například pro klasifikaci lineárně separovatelných obrazů.

Binární - skoková funkce nabývající pouze hodnot 0 a 1.

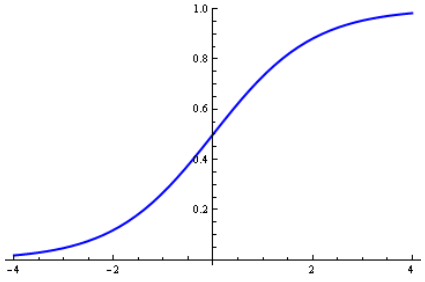
$$\mathbf{F}(\mathbf{B}) = 1 \quad \forall \mathbf{x} > 0, \text{ jinak } \mathbf{x} = 0.$$



Obr. 1.4. Binární funkce.

Logistická (sigmoida) - nejpoužívanější přenosová funkce odvozená jako aproximace přenosové funkce biologického neuronu.

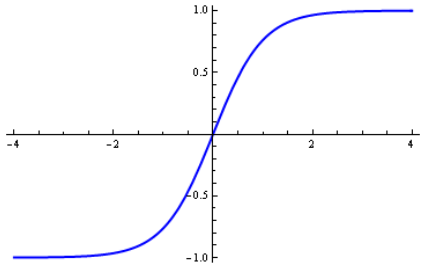
$$\mathbf{F}(\mathbf{S}) = \frac{1}{1 + e^{-x}}.$$



Obr. 1.5. Standardní logistická sigmoida.

Hyperbolický tangens - protože nabývá hodnot $\langle -1, 1 \rangle$ zahrnuje větší lineární úsek okolo počátku, což je výhodné.

$$\mathbf{Tanh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$



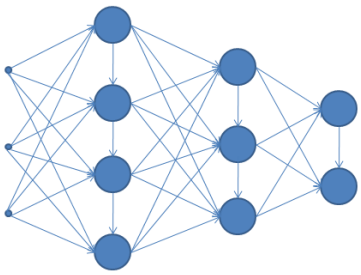
Obr. 1.6. Hyperbolický tangens.

U metod založených na porovnávání vzdáleností mezi daty (vzory) je přenosová funkce nahrazena metrikou. Mezi takové metody patří také dále zmiňované algoritmy Kohonenovy samoorganizující se mapy a algoritmus Neuronový plyn.

1.3. Umělé neuronové sítě

Umělá neuronová síť funguje jako soustava velkého množství jednoduchých a navzájem propojených umělých neuronů. Každý z nich může přijímat libovolné konečné množství vstupních dat a produkovat libovolné konečné množství identické informace. Spojením vzniká velice rozvětvený výstup. Myšlenkou bylo sestrojit výpočetní systém s maximální podobností s biologickou nervovou soustavou. Abychom dosáhli srovnatelných charakteristik, je v modelech dodržena stejná struktura, vlastnosti a zákonitosti, které platí pro biologický vzor.

K popisu sítě využíváme orientovaného grafu s ohodnocenými hranami a uzly (Obr. 1.7.). Množina uzlů (neuronů) $\{x_1, x_2, x_3, ..., x_N\}$ a hran (i, j) udává topologii sítě. Ta se obecně během výpočtu nemění. Algoritmus Neuronový plyn uvedený v podkapitole 1.6.2. je výjimkou, kdy se během iterací mění počet i rozmístění neuronů v síti.



Obr. 1.7. Příklad topologie vícevrstevné sítě.

Neurony obecně rozlišujeme na vstupní a výstupní. Je-li celkový počet neuronů N, počet vstupních neuronů S a počet výstupních R. Potom platí: $R < N$ a $S < N$, ale může platit: $R + S > N$, tzn. některé neurony se mohou stát zároveň vstupními i výstupními.

Síť je charakterizována [3]:
architekturou - uspořádáním uzlů a propojením hran,
algoritmem učení - tj. postup jakým se mění váhy v jednotlivých synapsích,
aktivační funkcí - principem, jakým se ze vstupů neuronu počítá jeho výstup.

Následující příklady [3] demonstrují, v čem spočívá výhoda neuronových sítí v porovnání s klasickými výpočetními systémy. Konvenční počítače postupují podle předem určeného postupu - algoritmu, ve kterém vykonávají jednotlivé dílčí operace. Neuronové sítě provádějí velké množství dílčích operací zároveň a v průběhu zpracování se postupně díky učení adaptují na daný problém tak, aby bylo nalezeno optimální řešení.

Příklad 1 – Předem neklasifikovaný problém

Pokud se při klasifikaci objektů setkáme s předem neklasifikovaným problémem, musí v běžném případě znovu nastoupit programátor a nastavit algoritmus i pro tuto odchylku. V případě neuronové sítě není nutné nový algoritmus vymýšlet. Pokud máme navíc nastaveno kvalitní učení a konfiguraci, síť zareaguje správně a objekt pravděpodobně zařadí do vhodné třídy.

Příklad 2 – Netypický vzorek

Chceme rozčlenit dopravní prostředky na kola, auta a letadla dle jejich propozic. Pokud ve vzorcích bude zařazena tříkolka, nebo kolo z 18. století, klasický program je pravděpodobně vyřadí. Případně upozorní programátora, že je nutné provést úpravy. Jakékoliv další zasahování do kódu je však spojeno s velkým nebezpečím vzniku dalších chyb. Díky adaptivnosti neuronových sítí se při předložení vhodných vzorů tomuto problému vyhneme.

Následující tabulka [3] shrnuje rozdíly mezi PC a neuronovou sítí.

Neuronová síť	**Počítač**
Určena vahami, prahy a strukturou.	Určen příkazy typu IF, THEN, GO TO.
Paměťové a výkonné prvky uspořádány spolu.	Separace procesu a paměti.
Paralelismus	Sekvenčnost
Vyrovnají se s odchylkami od vzorů	Nezvládají odchylky
Samoorganizace během učení	Statičnost programu

Tab. 1.1. Rozdíl mezi PC a neuronovou sítí.

Nyní jsme porovnali klasické výpočetní systémy a neuronové sítě. Zajímavé je ale také srovnání umělé a biologické neuronové sítě, jak jej ukazuje následující tabulka [3].

Biologická síť	**Umělá neuronová síť**
Dendrity.	n obecně reálných vstupů $x_1, x_2, ..., x_n$.
Synaptické brány (propustnost)	n obecně reálných váhových koeficientů $w_1, w_2, ..., w_n$ Záporné váhy *inhibiční* a kladné *excitační*.
Celkové podráždění neuronu (úhrnný el. potenciál).	Potenciál neuronu $\xi = \sum_{i=1}^{n} w_i x_i$, resp. $\xi = \sum_{i=0}^{n} w_i x_i$, kde $w_0 = -h$ a $x_0 = 1$.
Prahová hodnota vzruchu.	Práh h.
Elektrický impulz axonu indukovaný po dosažení prahové hodnoty vzruchu.	Výstup (stav) neuronu $y = f(\xi)$, f....přenosová funkce.

Tab. 1.2. Srovnání biologické a umělé neuronové sítě.

1.4. Kritéria dělení neuronových sítí

Umělé neuronové sítě mají velké množství podob, liší se dle typu úloh, pro které jsou určeny, výpočetní náročností a dalšími parametry. Kritéria dělení neuronových sítí jsou [3]:

Počet vrstev:

jedna vrstva – Kohonenova síť [1],

více vrstev – Percepton [5].

Algoritmus učení:

s učitelem – Učitelem je myšlena třída vektorů tvořených vzorem a požadovaným výstupem, na který danou síť chceme naučit.

bez učitele – Síť používající pouze informace obsažené ve vstupních datech.

Styl učení:

deterministické učení, *stochastické učení* - náhodné nastavování vah během algoritmu.

V algoritmech neuronových sítí rozlišujeme dvě fáze učení - fázi aktivační (vybavovací) a adaptační (učící). V případě sítě s učitelem je k učení zapotřebí trénovací množina. Množina dvojic vstup-výstup. Pokud se jedná o učení bez učitele, postačují nám informace obsažené ve vstupních datech. Pokud se lidská paměť již jednou naučí na nějaký vzor, má schopnost rozeznat originál, přestože jí předložíme „poničený“ vzorek. Existuje zároveň také nevýhoda. Pokud mozek

již jednou tento vzor rozezná, objeví jej pokaždé znovu. Pokud se setká s novou předlohou, která je mírně podobná již rozpoznané, může ji chybně zařadit do nesprávné třídy. Musíme tedy dbát na to, co neuronové síti předkládáme.

Pro maximální počet vzorů, na které se je síť schopna naučit, platí následující vztah:

$$\text{Počet vzorů} < 0.15 \times \text{Počet neuronů (vstupů)}.$$

1.5. Optimalizace modelu neuronové sítě

V současnosti je k dispozici velké množství neuronových sítí. Každá z nich je však vhodná pro jinak definovanou úlohu.

Nastavení modelů je individuální a liší se v závislosti na typu problému. Existují pouze doporučení, jak optimalizovat výsledky testování. Zlepšení lze dosáhnout pomocí nastavení velikosti parametru učení, šumu, určením vhodné topologie a počtu vrstev nebo přidáním neuronů.

1.5.1. Nastavení počtu vrstev

Neexistují přesné metody na stanovení počtu vrstev a neuronů. Vícevrstevné sítě mívají obvykle 3 nebo 4 vrstvy tří druhů: vstupní, výstupní a skryté. Důvod můžeme zjistit aplikací Kolmogorova teorému [7] na problematiku neuronových sítí. Vyplývá z něj, že v případě minimálně třívrstevné sítě s odpovídajícím počtem neuronů může být transformační funkce aproximována libovolnou funkcí f.

Síť se třemi vrstvami se vyznačuje:

- kratší dobou učení,
- rychlejším zpracováním zadané úlohy,
- stabilním vývojem globální chyby,
- nachází „horší“ aproximace než čtyřvrstevné sítě.

Síť se čtyřmi vrstvami:

- je vhodná pro aproximaci, predikci a odstranění šumu,
- má delší dobu učení,
- má méně stabilní vývoj globální chyby – větší členitost chybové plochy.

1.5.2. Nastavení počtu neuronů

Určení počtu neuronů v jednotlivých sítích je ještě o stupeň komplikovanější než stanovení počtu vrstev. U skrytých vrstev platí pravidlo „Čím méně, tím lépe“. Testování zpravidla zahajujeme se dvěma skrytými neurony a postupně počet navyšujeme dokud prudce klesá hodnota minimalizačního kritéria a vzrůstá kvalita neuronové sítě. Pokud se zvyšováním počtu neuronů současně nedosahujeme výrazných rozdílů u hodnot minimalizačního kritéria, případně pokud jeho hodnota stoupá, potom je další dodávání skrytých neuronů zbytečné a vede na jev zvaný přetečení, „overfitting“. U ostatních vrstev řešíme tento problém individuálně. Existuje však několik obecně používaných přístupů [3].

Pevné nastavení

Počet neuronů stanovíme podle univerzálních avšak mírně nepřesných vzorců vhodných především pro dvou– a třívrstevnou strukturu.

$$N_{Skryt} = \sqrt{N_{\text{Vstup}} \times N_{\text{Výstup}}}$$

$$N_{Skryt1} = N_{\text{Výstup}} \times \left(\sqrt[3]{\frac{N_{\text{Vstup}}}{N_{\text{Výstup}}}} \right)^2$$

$$N_{Skryt2} = N_{\text{Výstup}} \times \left(\sqrt[3]{\frac{N_{\text{Vstup}}}{N_{\text{Výstup}}}} \right).$$

Adaptivní nastavení

Další metodou pro určení nastavení počtu neuronů je algoritmus, který na základě měnící se globální chyby přidává a ubírá neurony ve skrytých vrstvách.

Nastavení pomocí genetických algoritmů

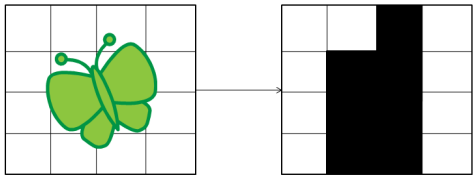
Nastavení pomocí genetických algoritmů patří mezi časově náročnější přístupy, ale poskytuje výrazně přesnější výsledky. Struktura sítě je tvořena několika faktory: počtem vrstev, počtem neuronů ve vrstvách, topologií spojení mezi vrstvami a neurony, hodnotami vah. Vstupní informace je tvořena chromozomem složeným z počtu vrstev, počtu neuronů a typu spojení. Tento chromozom reprezentuje celou síť. Jeho struktura je generována buď náhodně, nebo deterministicky (známe-li počet vstupů a výstupů). Principem je vybírání těch jedinců, kteří nejlépe vyhovují funkci vhodnosti (životnímu prostředí).

1.5.3. Nastavení trénovací množiny

Určení vhodné struktury trénovací množiny dat je klíčovým faktorem pro spolehlivost modelu. Tréninková množina dat se skládá z dvojic vektorů vstupních a výstupních dat. Ty dodají síti námi požadované vlastnosti. Při trénování neuronové sítě nevadí rozsah, který bude například z poloviny zasahovat mimo pracovní oblast. Mnohem nepříznivější důsledky by měla situace, kdy by požadované vstupní a výstupní signály nepokrývaly celou pracovní plochu.

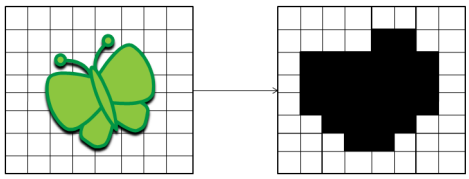
Důležité je také zachování reálného zastoupení vzorků v trénovací množině. Třídy by měly pokrýt všechny možné kombinace, které mohou nastat. Například pokud bychom chtěli rozeznávat automobily, učíme síť na třídách Škodovek, Porsche, Jaguarů a Trabantů. Jestliže bychom zahrnuli do každé třídy málo vzorků s malými rozdíly, pak se může stát, že při zařazování neznámého auta, které je oproti standardu upraveno, dojde k chybnému zařazení. Třídy informací, které jsou předkládány jako vzory by měly být na sobě částečně závislé. Jinak může nastat případ, že se síť nenaučí, ale zhloupne. Například při predikci vývoje cenného papíru na burze je vhodnější použít nejen data z historie, ale vzít v úvahu také index odvětví nejsilnějšího konkurenta. [7]

Délka vektorů popisujících daný problém je úměrná kvalitě, s níž je problém popsán.



Obr. 1.8. Popis obrázku čtvercovou sítí typu 4×4.

V prvním případě byl použit příliš krátký vektor a transformace dat byla hrubá. Proto byla použita hustší čtvercová síť, která lépe vystihla povahu obrázku.



Obr. 1.9. Interpretace Obr 1.8. popsaného čtvercovou sítí typu 8×8.

Pokud správně určíme oblast, ze které budeme data vybírat, musíme ještě určit jejich počet. Bude-li trénovacích dat nedostatek, výsledný model nebude dostatečně vyhovovat. Velké množství dat může naopak vést k duplikování informací o modelovaném systému. Při vyhodnocování zkoumaného vzorku potom může být nejasné, ke kterému z více duplicitních (téměř identických) vzorů jej přiřadit. Správně by měla být každá dvojice hodnot jedinečná. Počet vzorků a iterací testovacího algoritmu má vliv mimo jiné i na časovou náročnost.

Kvalitu modelu ovlivňuje také druh testovacího algoritmu. Zásadní vliv má dále počáteční nastavení parametrů. Optimální nastavení zjistíme experimentálně. Provádíme testování se stejnou trénovací množinou a proměnlivými parametry. Vybereme takové hodnoty parametrů, se kterými model dosahoval nejlepších výsledků.

Vzhledem k tomu, že přenosové funkce se pohybují obvykle v rozmezí $\langle 0, 1 \rangle$, nesmíme zapomenout na transformaci vstupních dat do tohoto intervalu. Pokud není nutné použití lineární funkce bez omezení, pak je lepší použít alternativní přenosovou funkci (například logistickou) a její vstupní i výstupní veličiny trans-

formovat do intervalu $\langle 0.2, 0.8 \rangle$. Tím je dosaženo transformace všech veličin do lineárních částí, zatímco nelineární části zůstanou zachovány pro eliminaci poruch. V případě, že jsou hodnoty rozmístěny rovnoměrně, používá se transformace lineární (podělení konstantou). Pro nerovnoměrné rozmístění se používají například následující transformace podle J. Turkeyho [7]

$$y = \arcsin\left(\sqrt{\frac{x}{n+1}}\right) + \arcsin\left(\sqrt{\frac{x+1}{n+1}}\right),$$

používaná pro transformaci hodnot z malého množství dat a

$$y = \sqrt{x} + \sqrt{x+1}$$

pro transformaci hodnot z časově omezených intervalů.
Optimalizace vah je dosaženo pomocí simulovaného žíhání, genetických algoritmů nebo lineární regrese.

1.6. Aplikace neuronových sítí

1.6.1. Kohonenovy mapy

Kohonenovy samoorganizující se mapy (SOM, Self–Organizing Map) se používají především pro shlukování objektů a zjednodušování vícerozměrné struktury. Při procesu učení pracují bez učitele. Principem algoritmu je nelineární projekce bodů z p–rozměrného prostoru do prostoru s nízkým počtem dimenzí, nejčastěji do roviny.

Kohonenovy mapy se vyznačují dvěma užitečnými vlastnostmi: adaptivností a schopností zachovat topografii. Pojem adaptivnost vyjadřuje, že neuronovou síť je možné natrénovat na určitou situaci, resp. určitý typ dat. Potom ani po dodání dodatečného pozorování nezměníme výpočetní náročnost celé úlohy, protože algoritmus nemusí znovu přehodnotit celý datový soubor. Zachování topografie zaručuje, že body, které ve vstupním prostoru ležely blízko sebe, budou blízko sebe také v prostoru výstupním.

Pokud chceme výsledky testování srovnávat graficky, je nutné dodržet přesné vzdálenosti mezi shluky i ve výstupní Kohonenově mapě. Kohonenova mapa se

skládá ze čtvercové nebo šestiúhelníkové mřížky neuronů. Bez újmy na obecnosti můžeme předpokládat jednotkovou vzdálenost mezi neurony. Velikost mřížky udává výslednou přesnost rozlišení.

Pro každý neuron c z vytvořené mřížky inicializujeme p–rozměrný vektor vah $\mathbf{w}_c = (w_{c_1},\ w_{c_2},\ \ldots,\ w_{c_p})^T$. Váhy lze určovat zcela náhodně, pro rychlejší konvergenci je ale vhodné volit hodnoty podle výsledků jiných metod. Po nastavení vah začíná v čase t první iterace algoritmu náhodným výběrem objektu $\mathbf{x}_i(t)$ z datového množiny [8].

Nyní je úkolem algoritmu nalézt v Kohonenově mapě tzv. vítězný neuron v, který je vzhledem ke svému vektoru vah vybranému vstupu nejblíže ve smyslu euklidovské metriky. Jeho určení je dáno vztahem [8]

$$v = \operatorname{argmin}_{c\in\Omega} \|x(t) - w_c\| .$$

Nyní následuje proces učení. Nemáme k dispozici žádného učitele, tzn. vzorové řešení, ke kterému by algoritmus měl směřovat, ale získané informace zahrneme do modelu modifikací vah vítězného neuronu. Ty se pro krok $t+1$ upraví podle vztahu

$$\mathbf{w}_v(t+1) = \mathbf{w}_v(t) + \alpha(t)[\mathbf{x}(t) - \mathbf{w}_v(t)],$$

kde $\alpha(t)$ vyjadřuje rychlost učení. Její hodnota s rostoucím t monotónně klesá. Pro zachování topografie a vzdálenosti bodů ve výstupní vrstvě musíme současně aktualizovat také vektory vah neuronů ležících v okolí vítěze. Pokud by byl v další iteraci vybrán objekt s podobnými hodnotami, díky aktualizaci vah by byl s větší pravděpodobností zobrazen na neuron z okolí předchozího vítěze. Body tak budou v blízkosti i ve výstupní vrstvě. Aktualizace sousedních neuronů probíhá podle vztahu

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \alpha(t)\eta(v,k,t)\left([\mathbf{x}(t) - \mathbf{w}_v(t)] + [\mathbf{w}_v(t) - \mathbf{w}_k(t)]\frac{d_{vk} - \Delta_{vk}\lambda}{\Delta_{vk}\lambda}\right),$$

kde $\eta(v,k,t)$ je funkce okolí monotónně klesající s rostoucí vzdáleností mezi vítězným neuronem v a okolním neuronem k v neuronové síti. Nebo-li s rostoucí $\|k - v\|$. [8]

d_{vk} a Δ_{vk} jsou vzdálenosti mezi objekty v datové množině, resp. mezi neurony v síti. λ je předem stanovený kladný parametr rozlišení. Výraz $\beta = \frac{d_{vk}-\Delta_{vk}\lambda}{\Delta_{vk}\lambda}$ představuje vylepšení. Platí, že pokud je d_{vk} větší než Δ_{vk} při daném λ, přitahuje vítěz okolní neuron blíže k sobě, a naopak. Tím převedeme vzdálenosti z datové skupiny do Kohonenovy mapy. [8]

Parametr η bývá často nastaven jako exponenciální funkce

$$\eta(v,k,t) = \exp\left(-\frac{\|k-v\|^2}{2\sigma(t)^2}\right),$$

která s rostoucím t monotónně klesá. Výsledné zobrazení potom závisí na parametru rozlišení λ a parametru velikosti okolí $\sigma(t)$. Optimální nastavení těchto parametrů hledáme experimentálně, závisí však také na velikosti mapy. Obecně pro SOM platí, že velikost okolí $\sigma(t)$ by měla být zpočátku relativně velká a s rostoucím t by měla postupně klesat k 1. Volba parametru λ závisí na velikosti mapy, variabilitě dat a požadovaném rozlišení. Čím menší je λ, tím vyššího rozlišení můžeme dosáhnout. Samozřejmě vyšší rozlišení vyžaduje také větší mapu. [8]

Stručně lze algoritmus shrnout do následujících kroků [8]:

1. Vytvoření sítě a přiřazení vah jednotlivým neuronům.
2. Náhodné vybrání objektu z datové matice a nalezení vítězného neuronu.
3. Aktualizace vah vítězného neuronu.
4. Aktualizace vah okolních neuronů.
5. Nepovinný krok - Aktivace nečinných neuronů, kteří dlouho nezvítězili, pomocí nového přiřazení náhodného vektoru vah.
6. Provádění kroků 2 až 5, dokud zobrazení na mapě nekonverguje, nebo neproběhne předem stanovený počet iterací algoritmu.

Kohonenova síť je bez učitele, ale existují také verze s učitelem (Learning Vector Quantization) LVQ1, LVQ2 a LVQ3 [9]. Jedná se o rozšíření Kohononenovy sítě vhodné pro klasifikaci. Principem učící vektorové kvantifikace je přiřazení hodnocení jednotlivým třídám neuronů. Děje se tak na základě četnosti, s jakou se vyskytují v dominantní skupině.

1.6.2. Algoritmus Neuronový plyn

V publikaci [1] je uveden další zástupce neuronových sítí vhodný pro shlukovou analýzu. V modelu neuronový plyn (Neural gas – NG) neexistuje pevně určená topologie. Každá síť se skládá z množiny N uzlů (neuronů).

$$\mathbf{A} \;=\; \{c_1, c_2, \ldots, c_N\}$$

Každému uzelu c_i je přiřazen referenční vektor $w_i \;\in\; \mathcal{R}^m$, identifikující pozici ve vstupním poli. Pro uzel c označíme N_c množinu jeho přímých topologických sousedů.

Předpokládá se, že vstupní m–rozměrný signál je generován rozdělením se spojitou hustotou pravděpodobnosti

$$p(\xi), \xi \;\in\; \mathcal{R}^m,$$

nebo z konečné trénovací množiny

$$\mathbf{D} = \{\xi_1, \xi_2, \xi \ldots, \xi_N\}\,.$$

Pro daný vstupní signál ξ je vítězný neuron $s(\xi)$ z uzlů množiny $\mathbf{A}$ ten, který má nejbližší referenční vektor

$$s(\xi) = \mathrm{argmin}_{c\in\mathbf{A}} \left\|\xi - \mathbf{w}_c\right\|.$$

V případě existence více uzlů se stejným nejlepším referenčním vektorem je náhodně vybrán jeden z nich.

Algoritmus NG třídí pro každý vstupní signál ξ uzly sítě podle vzdálenosti jejich referenčních vektorů $\mathbf{w}_{c_j}$. Koeficienty změny počtu uzlů a adaptace se průběžně zmenšují podle předem zadaného schématu. Je potřeba vybrat vhodné počáteční hodnoty $\lambda_i,\ c_i$ a vhodné konečné hodnoty $\lambda_f,\ c_f$.

Algoritmus lze definovat následujícím popisem [1]:

1. Inicializuj množinu $\mathbf{A}$ tak, aby obsahovala $\mathbf{N}$ uzlů c_i, tj.

$$A \;=\; \{c_1, c_2, \ldots, c_N\}$$

   s referenčními vektory $\mathbf{w}_{c_i} \in \mathcal{R}^m$ náhodně vygenerovanými s pravděpodobností $p(\xi)$. Inicializuj proměnnou čas $t = 0$.

2. Náhodně vygeneruj vstupní signál s pravděpodobností $p(\xi)$.

3. Seřaď všechny elementy množiny $\mathbf{A}$ podle jejich vzdálenosti od ξ, tj. vytvoř uspořádaný seznam indexů $\{i_0, i_1, \ldots, i_{N-1}\}$ takový, že
   $\mathbf{w}_{i_0}$ je referenční vektor nejbližší k ξ,
   $\mathbf{w}_{i_k}$ $(k = 0, 1, \ldots, N-1)$ je referenční vektor takový, že pro vektory $\mathbf{w}_{i_k}$ platí

$$\left\|\xi - \mathbf{w}_{i_{k-1}}\right\| \;<\; \left\|\xi - \mathbf{w}_{i_k}\right\|.$$

   Označme $k_i\left(\xi, \mathbf{A}\right)$ číslo k asociované s $\mathbf{w}_{i_0}$.

4. Adaptuj referenční vektory $\mathbf{w}_i$ podle vztahu

$$\Delta\mathbf{w}_i = \epsilon\left(t\right) \cdot h_\lambda\left(k_i\left(\xi, \mathbf{A}\right)\right) \cdot \left(\xi - \mathbf{w}_i\right)$$

   s následujícími časovými závislostmi (koeficienty útlumu):

$$\lambda\left(t\right) = \lambda_i \; \left(\lambda_f/\lambda_i\right)^{t/t_{max}},$$

$$\epsilon\left(t\right) = \epsilon_i \left(\epsilon_f/\epsilon_i\right)^{t/t_{max}},$$

$$h_\lambda\left(k\right) = \exp(-k/\lambda(t)).$$

5. Zvyš hodnotu čítače iterací: $t = t + 1$.

6. Pokud $t < t_{max}$, jdi na 2, jinak STOP.

Nastavení parametrů je doporučeno: $\lambda_i = 10,\ \lambda_f = 0.01,\ \epsilon_i = 0.5,$ $\epsilon_f = 0.005,\ t_{max} = 40000.$

1.6.3. Testování uvedených algoritmů

Testování pomocí algoritmu SOM

Podle postupu popsaném v sekci 1.6.1. byl v rámci diplomové práce vytvořen algoritmus Kohonen. Hodnoty parametrů a váhy neuronů byly zvoleny náhodně.

Pro testování byla použita asi nejznámější a nejpoužívanější datová množina pro rozpoznávání vzorů, Iris flower data set, neboli Fisherova datová množina irisů [10]. Vícerozměrná datová množina byla představena Sirem Ronaldem Aylmerem Fisherem jako příklad diskriminační analýzy. Někdy je označována jako Andersonův Iris data set, protože Edgar Anderson posbíral data pro vyčíslení geografické variability irisů v Gaspé Peninsule. Data set se skládá ze tří tříd s 3 $\times$ 50 zástupci irisů. Třídy odpovídají rostlinným druhům Iris setosa, Iris virginica a Iris versicolor.

Obr. 1.10. Iris setosa, Iris virginica a Iris versicolor [10].

První třída je lineárně oddělitelná od ostatních dvou, ale zbylé dvě nelze rozlišit.

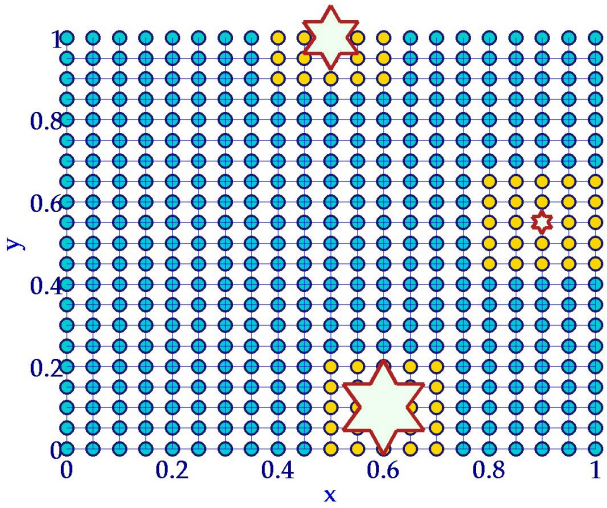
Na každém jedinci byly naměřeny čtyři znaky:

- délka kališních lístků v cm,
- tloušťka kališních lístků v cm,
- délka okvětních plátků v cm,
- tloušťka okvětních plátků v cm.

Na základě kombinace těchto čtyř charakteristik vytvořil Fisher lineární diskriminační model pro určování rostlinného druhu.

Datová matice irisů byla použita jako vstup do sestrojeného algoritmu Kohonen. Obrázek 1.11. ukazuje grafický výstup testování pomocí tohoto algoritmu. Modrá kolečka tvoří neuronovou síť. Hvězdičky reprezentují vybrané neurony, jejich velikost odpovídá četnosti s jakou byly vybrány. Žluté body označují neurony, jejichž váhy byly měněny v průběhu algoritmu, aby se docílilo konvergence ke správnému neuronu. Algoritmus v tomto případě data vhodně rozdělil do tří skupin. Konkrétní výsledky znějí takto:

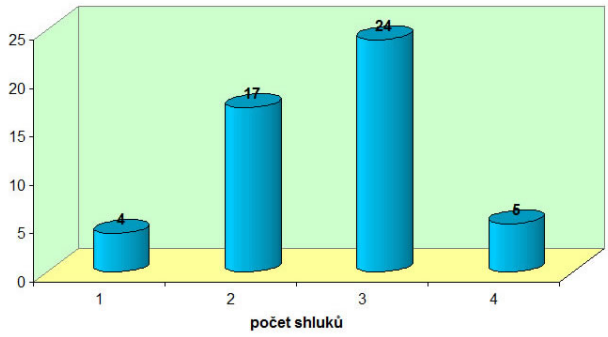
Neuron číslo 231 byl vybrán 18-krát, z toho 18-krát zástupce druhu Iris setosa.
Neuron číslo 255 byl vybrán 31-krát, z toho 18-krát zástupce druhu Iris virginica a 13-krát zástupce druhu Iris versicolor.
Neuron číslo 390 byl vybrán jednou a byl mu přiřazen zástupce druhu Iris versicolor.



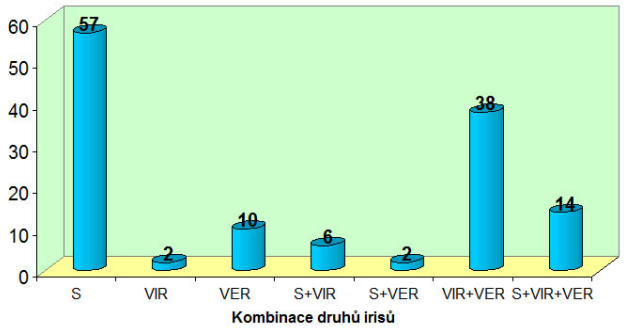
Obr. 1.11. Výstup algoritmu Kohonen.

Algoritmus Neuron byl spuštěn padesátkrát, přičemž v průběhu jednoho spuštění proběhne 50 iterací. Algoritmus spouštíme v programu Matlab pomocí souboru vyber_neuronu_iris.m. Na výstupu se objeví informace o číslech a indexech vybraných neuronů a počtu, kolikrát byly vybrány. Z prvního sloupce matice V identifikujeme druh irisu, který byl přiřazen neuronu ze třetího sloupce matice V. Jednotliví zástupci irisů jsou očíslováni následovně: $1-50$ Iris setosa, $51-100$ Iris versicolor, $101-150$ Iris virginica. Do tabulky Testování Kohonen.xls byly za-

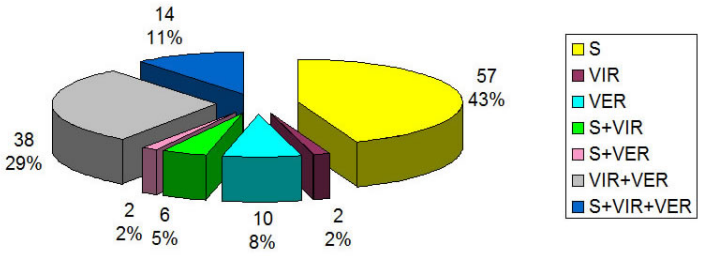
znamenány údaje o číslu neuronu, počtu, kolikrát byl vybrán, kombinacích irisů, které byly během testovaní přiřazeny k sobě, a počtu shluků. Z těchto údajů byly zpracovány grafické výstupy. Tabulka je součástí elektronické přílohy.



Obr. 1.12. Četnosti výsledného počtu shluků.



Obr. 1.13. Četnosti sloučených kombinací irisů.



Obr. 1.14. Zastoupení kombinací irisů.

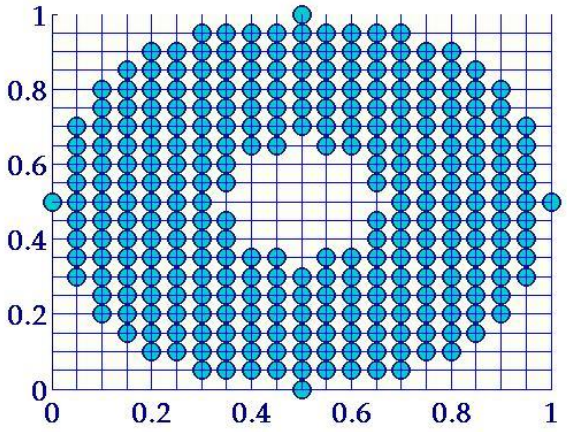
Z grafu 1.12. je vidět, že algoritmus dobře odhadl datovou strukturu a nejčastěji byly vytvořeny tři shluky. Četnost vytvoření dvou shluků je také relativně velká, což může být způsobeno neseparabilitou druhů Iris virginica a Iris versicolor, které splývají. Tento fakt je potvrzen také na obrázku 1.13., kde si můžeme všimnout, že irisy druhu setosa (S) byly nejčastěji identifikovány pro samostatný shluk. Navíc počet shluků, kde byly irisy druhu setosa chybně sloučeny s jinými druhy, je velice malý. Naopak druhou nejvýznamnější skupinu netvoří již samostatné druhy Iris virginica a Iris versicolor, ale právě jejich kombinace (VIR+VER). Tyto dva shluky (S a VIR+VER) tvoří 73% celkového počtu.

Testování pomocí algoritmu Neuronový plyn

Algoritmus Neuronový plyn byl v rámci diplomové práce naimplementován do programu Matlab. Startovní parametry byly nastaveny dle doporučení na $\lambda_i = 10,\ \lambda_f = 0.01,\ \epsilon_i = 0.5,\ \epsilon_f = 0.005,\ t_{max} = 40000$ [1]. Protože je tento algoritmus vhodný pro shlukovou analýzu, byla pro testování použita opět Fisherova datová matice irisů. Podle dosažených výsledků bude možné srovnat účinnost tohoto algoritmu s algoritmem Kohonenových samoorganizujících se map pro tento konkrétní případ.

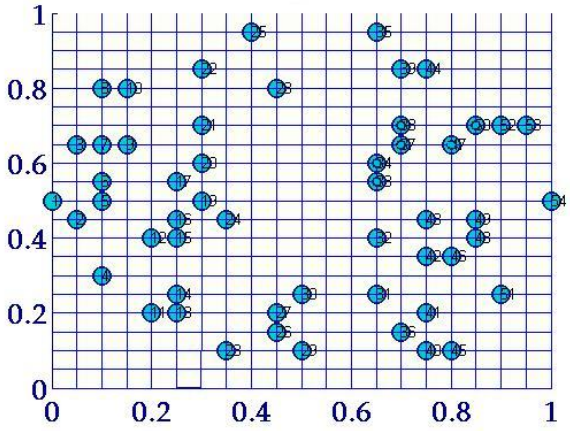
Popis algoritmu dle výše navržené struktury:

1. Náhodnost topologie a proměnlivý počet neuronů v síti jsou řízeny následovně. Nejvprve byla vytvořena síť neuronů ve tvaru mezikruží (Obr. 1.15.).



Obr. 1.15. Pomocná síť s pravidelnou topologií ve tvaru mezikruží.

Neuronům ležícím uvnitř mezikruží byla přiřazena pravděpodobnost vybrání 0.2 a ostatním 0. Příklad výsledné stochastické mřížky s proměnlivou topologií a počtem neuronů zobrazuje Obr. 1.16. Tím jsme splnili oba požadavky na náhodnost.



Obr. 1.16. Stochastická mřížka.

Referenční vektory přiřazené neuronům jsou pro tuto úlohu čtyřrozměrné a generované z normálního rozdělení. Čas je v první iteraci nastaven na $t = 0$.

2. Vstupní signál negenerujeme, ale poslouží nám náhodně vybraný zástupce

z testovací datové množiny.

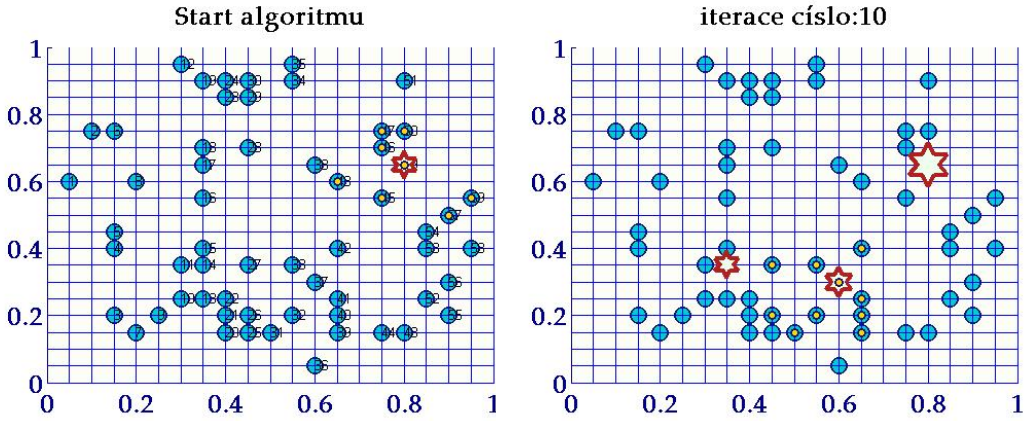
3. Spočteme vzdálenost referenčních vektorů od vybraného vstupu a určíme vítězný neuron.

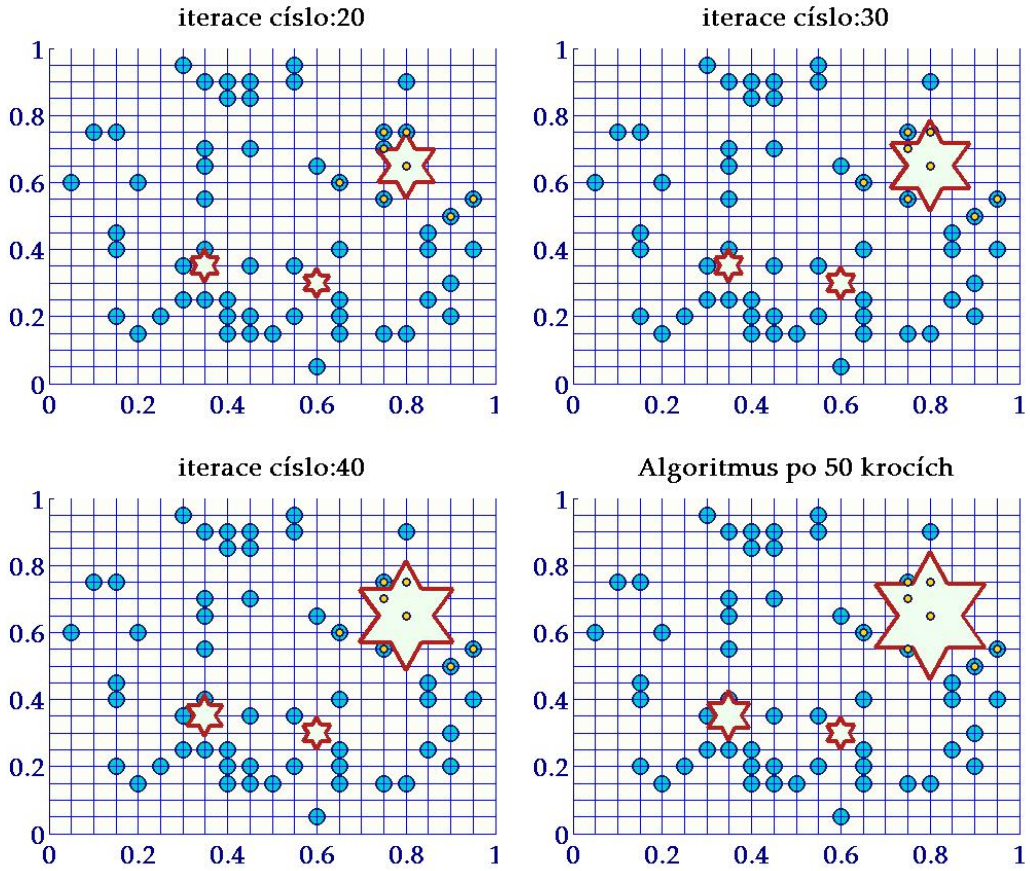
4. V dalším kroku dochází k modifikaci vah neuronů z okolí vítěze.

5. Přejdeme do další iterace tím, že nastavíme $t = t + 1$. Koeficienty útlumu jsou funkcemi parametru t, takže k jejich modifikaci dochází automaticky.

6. Hodnota t_{max} je nastavena na 50 iterací.

Grafický průběh algoritmu znázorňuje následující sekvence obrázků. Značení je zachováno stejné jako u algoritmu Kohonen. Modrá kolečka tvoří neuronovou síť, žlutá kolečka zobrazují neurony, jejichž váhy byly v daném algoritmu měněny. Hvězdičky reprezentují vítězné neurony, jejich velikost odpovídá počtu, kolikrát byly doposud vybrány. Vítězný neuron ze zobrazované iterace poznáme tak, že v jeho okolí jsou žlutě označeny neurony, jejichž váhy byly v této iteraci měněny.

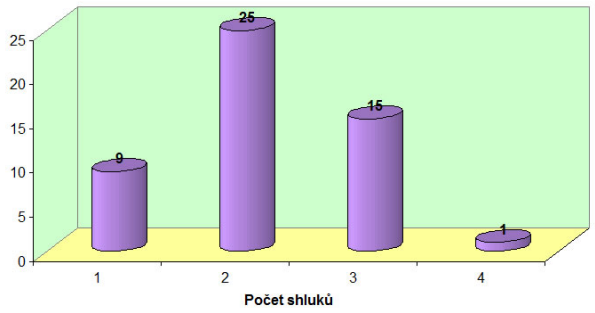




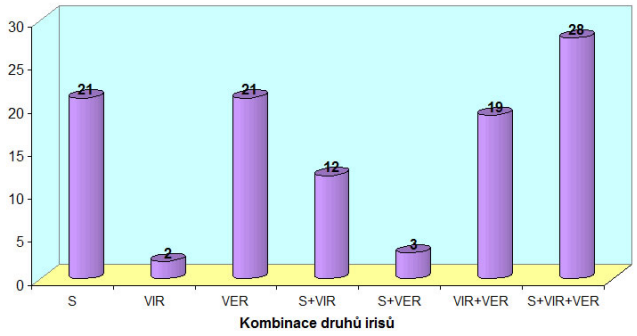
Obr. 1.17. Grafické výstupy algoritmu v uvedených iteracích.

Výstup algoritmu po padesáti iteracích byl následující:
Neuron číslo 13 na pozici $(6, 5)$ byl vybrán $1\times$.
Neuron číslo 18 na pozici $(7, 2)$ byl vybrán $18\times$.
Neuron číslo 37 na pozici $(15, 14)$ byl vybrán $31\times$.

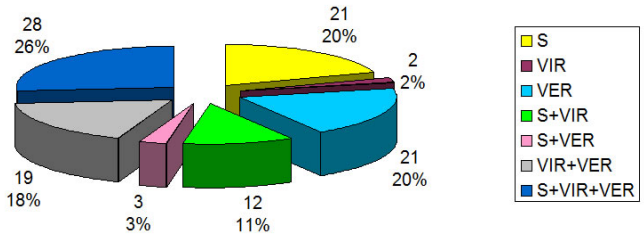
Při testování byly dodrženy stejné podmínky jako u algoritmu Neuron – algoritmus byl spuštěn padesátkát, přičemž pro každý průběh je nastaveno 50 iterací. Spouštěcí soubor má název vyber_neuronovy_plyn_stochastmrizka.m a druh irisu opět identifikujeme z matice V. Výsledky byly zaznamenávány do tabulky Testování Neuronový plyn.xls a jsou znázorněny grafy 1.18. - 1.20. Tabulka Testování Neuronový plyn.xls je součástí elektronické přílohy.



Obr. 1.18. Četnosti výsledného počtu shluků.



Obr. 1.19. Četnosti sloučených kombinací irisů.



Obr. 1.20. Zastoupení kombinací irisů.

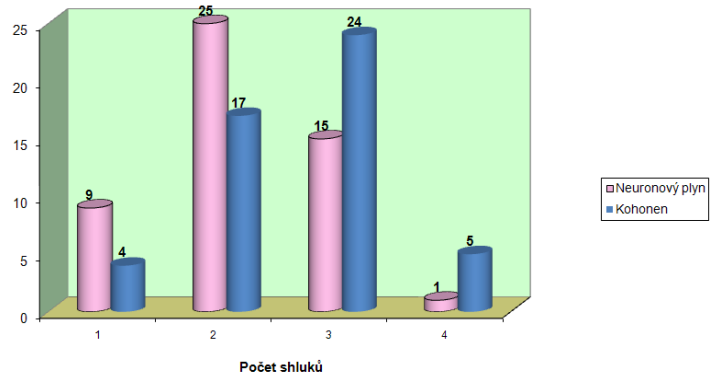
Agoritmus Neuronový plyn nebyl při testování stejně efektivní jako Kohonenovy samoorganizující se mapy. Data nejčastěji rozdělil do dvou shluků (Obr. 1.18.). Také struktura shluků není příliš přehledná. Na Obr. 1.19. pozorujeme čtyři významné kombinace sloučených rostlin: S+VIR+VER, S, VER, VIR+VER.

V tomto případě nelze potvrdit, že by algoritmus byl schopen kvalitně odlišit Iris setosa od ostatních zástupců.

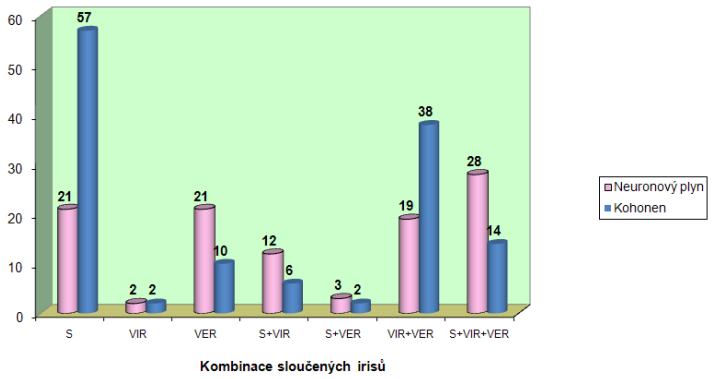
Porovnání výsledků

V předchozích podkapitolách byly uvedeny dva algoritmy pro řešení úlohy shlukové analýzy - Kohonenovy samoorganizující se mapy a Neuronový plyn. Protože byla k testování použita datová množina, o které máme určité informace, můžeme výsledky lépe interpretovat a vyhodnotit úspěšnost algoritmů.

Z testování vyplývá, že algoritmus Kohonenovy samoorganizující se mapy se ukázal být pro tuto úlohu vhodnější než algoritmus Neuronový plyn. Jeho výsledky jsou lépe interpretovatelné a odpovídají informacím o datové struktuře (počet shluků, neseparabilita druhů Iris virginica a Iris versicolor). Následující grafy umožňují srovnání algoritmů.



Obr. 1.21. Porovnání četnosti dosaženého počtu shluků.



Graf 1.22. Porovnání kombinací irisů vzniklých shlukováním.

Výrazné rozdíly na grafu 1.22. jsou způsobeny faktem, že algoritmus Kohonen častěji zobrazoval data na vyšší počet shluků než algoritmus Neuronový plyn. Díky tomu je například počet shluknutí izolovaného druhu Iris Setosa výrazně vyšší.

2. Stochastická optimalizace

2.1. Evoluční algoritmy v optimalizačních úlohách bez podmínky

Optimalizační problémy jsou součástí našeho každodenního života. Často potřebujeme zjistit nejefektivnější nastavení parametrů, nejvýnosnější složení portfolia nebo chceme minimalizovat ztrátovou funkci nějaké jiné veličiny. Můžeme se ale setkat s úlohami, kdy deterministické postupy selhávají, nebo neposkytují řešení v přijatelném čase. Proto se v těchto případech dává přednost stochastickým algoritmům. Jako zástupce můžeme uvést genetické algoritmy, simulované žíhání, slepé prohledávání nebo algoritmus SOMA [2]. Tato kapitola je věnována heuristickým algoritmům diferenciální evoluce a vlastnímu algoritmu Rypoš [11], které oba čerpají inspiraci v přírodě.

Stochastické algoritmy jsou vesměs založeny na postupném zlepšování řešení. Musíme mít na vědomí, že heuristické metody nemusí poskytnout přímo optimum, ale pouze jeho „dobré přiblížení“. Konvergence algoritmů není mnohdy matematicky podložena, přesto je vysoká účinnost těchto metod odzkoušena experimentálně.

2.1.1. Diferenciální evoluce

Diferenciální evoluce patří do třídy užívaných heuristických algoritmů, přesněji mezi algoritmy genetické. Tato kapitola pochází ze zdroje [11].

Genetické algoritmy jsou druhem evolučních algoritmů. Jejich základní myšlenka spočívá v zakódování informace při hledání tak, aby slabší generace byla v dalším kroku nahrazena generací lepší. Při hledání optima můžeme považovat body na funkci za části dědičného materiálu a potom uplatnit evoluční operátory (křížení, mutaci a selekci), stejně jako je pozorujeme v přírodě.

Zmíněné evoluční operátory fungují na následujících principech:

selekce – předpokládáme, že nejsilnější jedinci z populace mají větší pravděpo-

dobnost přežít a předat své vlastnosti,

křížení – dva nebo více jedinců z populace si vymění informace a vzniknou noví jedinci, kteří kombinují vlastnosti rodičů,

mutace – informace zakódovaná v jedinci může být náhodně změněna.

Řízení heuristiky je možné pomocí parametrizace nebo selekce. Nabízí se zde možnosti jako:

- Mutaci a křížení lze řídit elitářstvím, řízenou mutací (určíme sílu mutace a její úbytek v průběhu generace), řízením křížení (určíme procentuální část neelitního podílu populace, která se bude křížit).
- Nastavením maximálního počtu generací.
- Řízením výběru rodiče na základě jeho fitness (síla, robustnost, ohodnocení).
- Porovnávání fitness pro snazší nalezení vhodných rodičů pro budoucí generaci.
- Handicapováním oblasti hledání, o které víme, že zde pravděpodobně globální extrém neleží.
- Užití vyhledávacího algoritmu na dohledání optima poté, co genetické algoritmy ukončily svou činnost.

Diferenciální evoluce (DE)

Nová populace Q je v algoritmu diferenciální evoluce vytvářena tak, že každému bodu ze staré populace P se vytvoří jeho potenciální konkurent y. Do nové populace zahrneme ten z této dvojice bodů, který má nižší funkční hodnotu.

Zkřížíme-li bod $\mathbf{u}$ a bod $\mathbf{x_i}$ tak, že kterýkoliv prvek x_{i_j} nahradíme hodnotou u_j s pravděpodobností C, získáme nového potenciálního konkurenta $\mathbf{y}$.

Nejčastěji používané postupy na generování nového bodu jsou:

Postup *RAND*, který generuje bod $\mathbf{u}$ ze tří prvků staré populace podle vztahu:

$$\mathbf{u} = \mathbf{r_1} + F(\mathbf{r_1} - \mathbf{r_3}),$$

kde $\mathbf{r_1}$, $\mathbf{r_2}$ a $\mathbf{r_3}$ jsou navzájem různé body náhodně vybrané z populace P s výjimkou aktuálního bodu $\mathbf{x_i}$, F je vstupní parametr.

Heuristika *BEST* využívá modifikace bodu $\mathbf{x}^{\text{best}}$, nejlepšího bodu z populace P podle vztahu:

$$\mathbf{u} = \mathbf{x}_{\text{best}} + F(\mathbf{r_1} + \mathbf{r_2} - \mathbf{r_3} - \mathbf{r_4}),$$

kde $\mathbf{r_1}$, $\mathbf{r_2}$, $\mathbf{r_3}$ a $\mathbf{r_4}$ jsou vzájemně různé náhodně vybrané body z populace P různé od aktuálního bodu $\mathbf{x_i}$ od bodu $\mathbf{x}^{\text{best}}$ s nejnižší funkční hodnotou v P. $F > 0$ je vstupní parametr.

Nový vektor $\mathbf{y}$ vznikne „křížením“ vektoru $\mathbf{u}$ a náhodně vybraného vektoru $\mathbf{x}$ tak, že každá složka x_i je přepsána hodnotou u_i s pravděpodobností C. $C \in \langle 0, 1 \rangle$ je další vstupní parametr heuristiky.

$$y_i = \begin{cases} u_i & \text{pro } U < C \text{ nebo } i = j \\ x_i & \text{jinak} \end{cases} \qquad i = 1, 2, \ldots, d,$$

(kde U je náhodná veličina rovnoměrně rozdělená na intervalu $\langle 0, 1)$ a j je náhodně vybraný index složky vektoru zabezpečující přepis alespoň jedné složky vektoru $\mathbf{x}$ při jakékoliv volbě hodnoty C.

Agoritmus DE je vhodný pro použití především díky poměrně výpočetně nenáročnému generování bodu $\mathbf{y}$ i celkové jednoduchosti. Velkou pozornost musíme ale věnovat výběru vstupních hodnot F a C, na které je algoritmus velmi citlivý.

Žádná z těchto heuristik nezaručuje, že vygenerovaný bod $\mathbf{y}$ bude ležet v D. Pokud nastane tento případ, je aplikována tzv. perturbace, která zrcadlově překlopí souřadnici $y_i \notin \langle a, b \rangle$.

2.1.2. Aplikace – Vlastní algoritmus RYPOŠ

V této sekci bude představen vlastní algoritmus Rypoš, který byl vytvořen v rámci mé bakalářské práce na téma Heuristiky [11]. Je založen na principech diferenciální evoluce a analogii s australským hlodavcem Rypošem lysým. Tento živočich žije v podzemních chodbách pod povrchem pouště. Rypoš patří mezi živočišné druhy tvořící kolonie se sociálním uspořádáním a hierarchií podobnou

hmyzu. V kolonii se rozmnožuje jen jedna samice – královna, k páření má několik plodných samců a zbytek kolonie tvoří nerozmnožující se dělníci. Z toho vyplývá, že dělníci jsou více spřízněni s královnou, než sami se sebou, a jejich rozmnožování je nevýhodné. Naopak královna rodí efektivní potomstvo nesoucí veškeré v populaci se vyskytující geny. Tím se zároveň zvyšuje úspěšnost i četnost kolonie.

Obr. 2.1. Rypoš lysý.

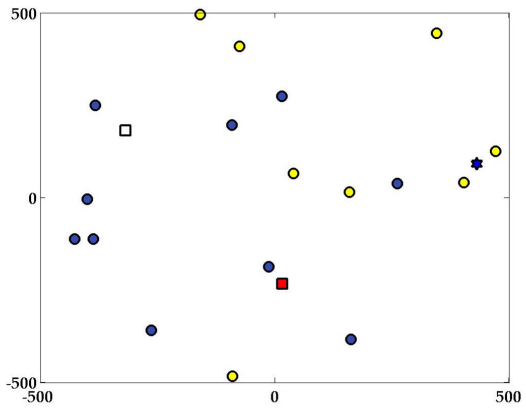
Princip fungování algoritmu RYPOŠ:

- Nejprve náhodně vygenerujeme původní populaci o 10 jedincích (žlutá kolečka).
- Poté vybíráme místa, kde mohou vzniknout hnízda (červený čtverec). První hnízdo je zvoleno v prvním generovaném bodě populace, jako další jsou zaznamenána jen ta hnízda, která mají lepší funkční hodnotu, než v historii nalezené hnízdo.
- Ostatní žluté body se stávají dělníky, kteří se budou starat o shánění potravy pro kolonii.
- Dále je náhodně vygenerována královna (modrá hvězdička).
- Královna si vybírá „ženicha“ (hnízdo, bílý čtverec) podle dvou hledisek: buď zvolí takové hnízdo, které je nejblíž (ženichovy feromony na ni působí nejvíce, je jejímu srdci nejblíž). Nebo zvolí hnízdo v bodě s nejnižší funkční

hodnotou, protože tento ženich je pro ni nejlepší, je „nejbohatší“. Je také středem pro generování další populace.

- Královna má s ženichem 10 potomků (modrá kolečka), kteří jsou v našem případě generováni z normálního rozdělení kolem vybraného hnízda. Všechny potomky z jednoho vrhu posouváme o stejnou vzdálenost (disperzi) závislou na pořadí vrhu (číslu iterace).
- Nyní máme k dispozici 20 jedinců, ale pro tak velkou kolonii by v okolí nebyl dostatek hlíz, a proto musí 10 jedinců zahynout. Královna již splnila svou funkci a také kolonii opouští. Máme tedy opět kolonii nejlepších 10 rypošů složenou z jedinců původní populace i z přeživších potomků (světle modrá kolečka).
- Nyní se opět náhodně objeví (vygeneruje) královna a cyklus rození probíhá znovu dokola. Cyklus je ukončen po 20 vrzích. Nejlepší hnízdo z posledního vrhu je výsledným nalezeným minimem.

Výstup algoritmu po padesáti iteracích zobrazuje Obr. 2.2. Žluté body značí původní body populace. Bílé čtverečky tvoří plodní samci. Královna je reprezentována modrou hvězdičkou. Hnízdo (ženich zvolený královnou) je značeno červeným čtverečkem. Modré body značí novou populaci rypošů. Jako aproximace je vyhodnoceno hnízdo s nejlepší (minimální) hodnotou.



Obr. 2.2. Výstup algoritmu Rypoš.

Testování tohoto algoritmu bylo provedeno v rámci mé bakalářské práce [11]. Pro vyhodnocení úspěšnosti byly použity tzv. testovací funkce, u kterých je hodnota globálního minima známá.

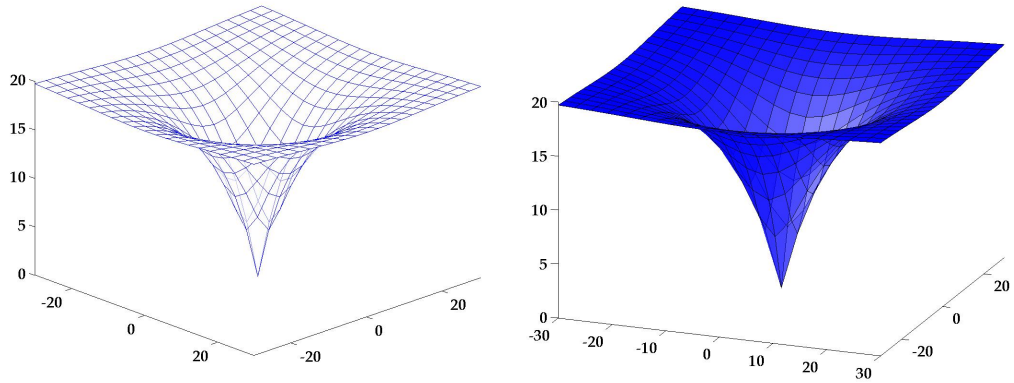
2.1.3. Testovaní algoritmu Rypoš

Pro testování spolehlivosti a úspěšnosti algoritmů jsou využívány speciální testovací funkce, jejichž hodnotu globálního minima známe a u nichž je úloha optimalizace extrému velmi složitá.

Pro praktické testování bylo použito těchto pět funkcí: Ackleyho funkce, první De Jongova funkce, Griewangkova funkce, Rastrigova funkce a Rosenbrockovo sedlo (Druhá De Jongova funkce). Jejich funkční předpisy, definiční obory a reálné hodnoty minima jsou převzaté z [2] a jsou uvedeny u jednotlivých grafů (2.3. a 2.4.), které byly vykresleny pomocí Matlabu vždy pro případ Dim $= d = 2$.

Ackleyho funkce

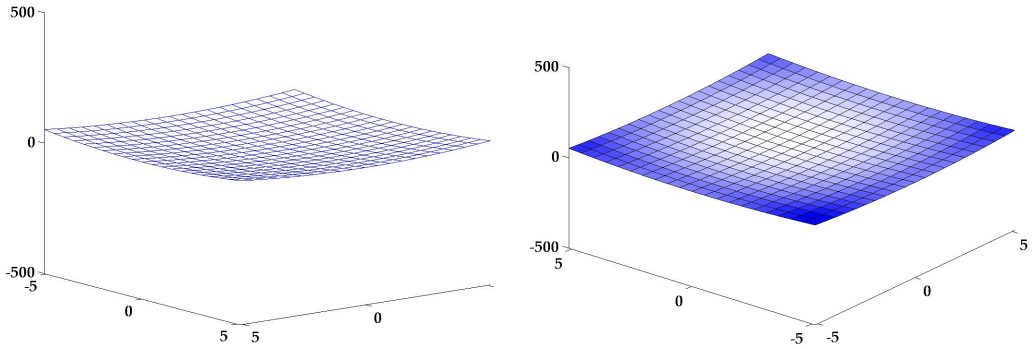
$D = \langle -30, 30\rangle \times \cdots \times \langle -30, 30\rangle$, minimum $\mathbf{x} = (0, \ldots, 0)$, $\mathbf{f(x)} = 0$



funkční předpis: $\mathbf{f(x)} = \sum_{i=1}^{Dim-1}(20 + e^{-0.2\sqrt{0.5(x_{i-1}^2 - x_i^2)}} - e^{0.5(\cos(2\pi x_{i+1}) + \cos(2\pi x_i))})$

První DeJongova funkce

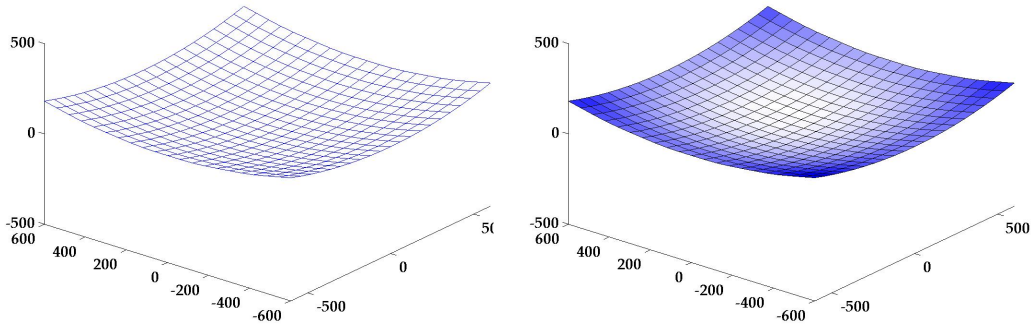
$D = \langle -5.12, 5.12 \rangle \times \cdots \times \langle -5.12, 5.12 \rangle$, minimum $\mathbf{x} = (0, \ldots, 0)$, $\mathbf{f(x)} = 0$



funkční předpis: $\mathbf{f(x)} = \sum_{i=1}^{Dim} x_i^2$

Griewangkova funkce

$D = \langle -600, 600 \rangle \times \cdots \times \langle -600, 600 \rangle$, minimum $\mathbf{x} = (0, \ldots, 0)$, $\mathbf{f(x)} = 0$

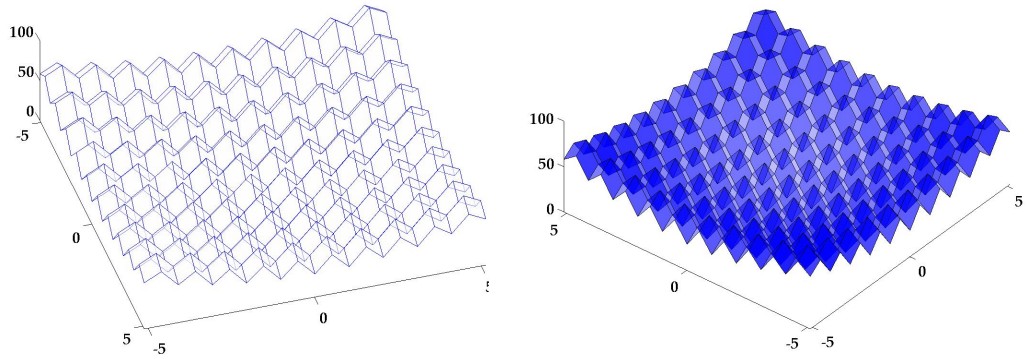


funkční předpis: $\mathbf{f(x)} = -\prod_{i=1}^{Dim} \cos(\frac{x_i}{\sqrt{i}} + \sum_{i=1}^{Dim} \frac{x_i^2}{4000} + 1$

Obr. 2.3. Testovací funkce, předpisy, globální minima a jejich funkční hodnoty

Rastrigova funkce

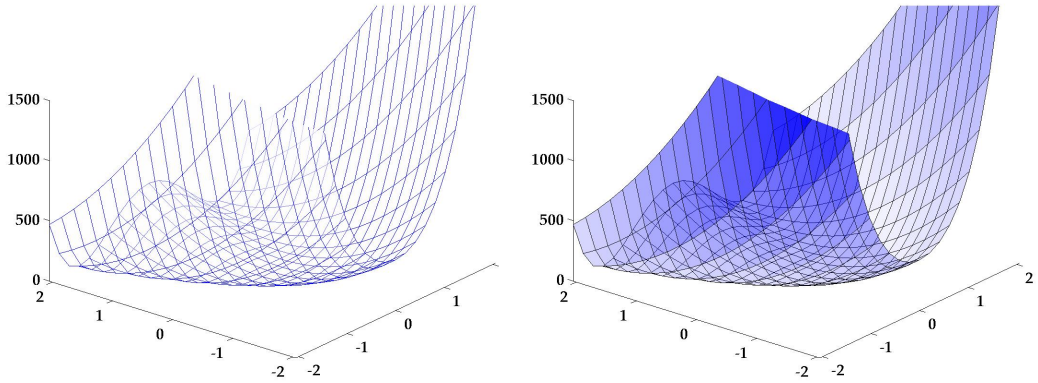
$D = \langle -5.12, 5.12 \rangle \times \cdots \times \langle -5.12, 5.12 \rangle$, minimum $\mathbf{x} = (0, \ldots, 0)$, $\mathbf{f(x)} = 0$



funkční předpis: $\mathbf{f(x)} = 10Dim \sum_{i=1}^{Dim}(x_i^2 - 10\cos(2\pi x_i))$

Rosenbrockovo sedlo

$D = \langle -2.048, 2.048 \rangle \times \cdots \times \langle -2.048, 2.048 \rangle$, minimum $\mathbf{x} = (1, \ldots, 1)$, $\mathbf{f(x)} = 0$

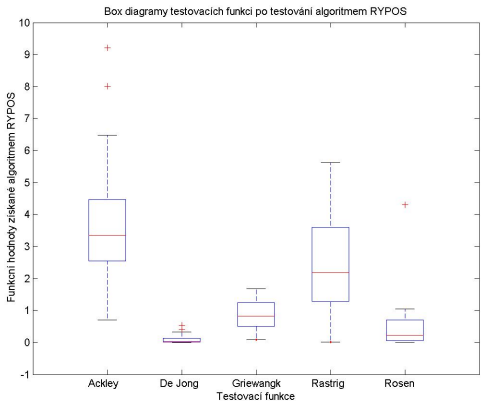


funkční předpis: $\mathbf{f(x)} = \sum_{i=1}^{Dim-1}(100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2)$

Obr. 2.4. Testovací funkce, předpisy, globální minima a jejich funkční hodnoty

Na Grafu 2.5. je krabičkový diagram pro funkce Ackleyho, první De Jongovu, Griewangkovu, Rastrigovu a pro Rosenbrockovo sedlo vytvořený z funkčních hodnot získaných testováním algoritmu RYPOŠ. Z testování vyplývá, že nejlépe algoritmus vyhodnotil úlohu první DeJongovy funkce, pro kterou je medián téměř roven nule a rozpětí hodnot je velmi malé. Za velmi uspokojivý výsledek můžeme považovat také výsledky dosažené pro Rosenbrockovo sedlo a Griewangkovu funkci. Algoritmus Rypoš se během testování dobře vyrovnal s problémem uváznutí v lokálním extrému. V rámci mé bakalářské práce [11] byl algoritmus Rypoš porovnán také s jinými běžně používanými heuristickými algoritmy a dle

dosažených výsledků prokázal, že je stejně efektivní, jako jeho ostatní konkurenti. Navíc se řadil mezi jedny z nejlepších.



Krabičkový diagram
1 – Ackleyova funkce
2 – DeJongova funkce
3 – Griewangkova funkce
4 – Rastringova funkce
5 – Rosenbrockovo sedlo

Obr. 2.5. Výsledek testování algoritmu RYPOŠ

2.2. Algoritmy pro optimalizaci s podmínkou

V rámci této kapitoly se budeme zabývat heuristickými optimalizačními algoritmy řešícími úlohu hledání extrému vícekriteriální funkce při splnění přípustných podmínek.

V části 2.2.2. je zmíněn algoritmus zobecněné simplexové metody, jejíž předlohou je známý Nelder-Meadův algoritmus [12]. Minimum cenového funkcionálu je hledáno pomocí simplexů vytvářených v oblasti přípustných řešení. Porovnáváme funkční hodnoty ve vrcholech simplexu a nejhorší z vrcholů, bod s maximální hodnotou cenového funkcionálu $J(x)$, je nahrazen svou α–reflexí.

Existují však i jiné postupy pro splnění přípustných podmínek. V druhé části kapitoly jsou zpracována tři selektivní kriteria: GDE1, GDE2 a GDE3 [14]. Všechna jsou založena na využití podmíněné dominance. Tato kritéria je možné implementovat například do výše zmíněného algoritmu Rypoš [11] a získat tak heuristický optimalizační postup, který již není omezen pouze na optimalizaci bez podmínek.

2.2.1. Obecná úloha optimalizace s vazbami

V celé kapitole se budeme zabývat úlohou optimalizace s vazbami. Obecně lze tento problém definovat následovně:
Hledáme globální minimum cenového funkcionálu

$$J := \mathbb{R}^N \to \mathbb{R} (N \in \mathbb{N}) \text{ na množině } U_{ad},$$

$$U_{ad} := \{x \in \mathbb{R}^N;\, g_j(x) \leq 0 \text{ pro } j = 1, \ldots, M,\ s_i \leq x_i \leq h_i \text{ pro } i = 1, \ldots, N\}.$$

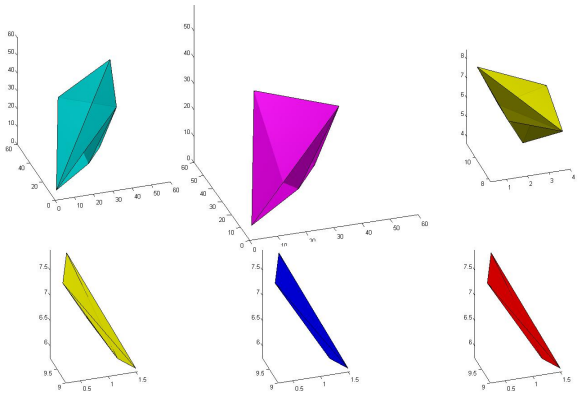
Budeme předpokládat, že U_{ad} je neprázdná a konvexní. Toho dosáhneme, pokud zvolíme všechny funkce $g_j \colon \mathbb{R}^N \to \mathbb{R}$ konvexní. Horní a spodní omezení h_i, s_i na proměnnou $x \in \mathbb{R}^N$ tvoří totiž „kvádr“ v N-rozměrném stavovém prostoru. [13]

Úloha podmíněné optimalizace je poměrně složitá. Neexistuje velké množství obecně aplikovatelných heuristických algoritmů, které by byly schopny ji řešit. V následujícím textu se seznámíme s algoritmem Complex Method a s algoritmy zobecněné diferenciální evoluce.

2.2.2. Aplikace – Algoritmus Complex Method (ACM)

V této sekci se budeme věnovat algoritmu uveřejněnému v knize [13]. Jedná se o rozšíření Nelder-Meadovy simplexové metody [12] na speciální úlohu optimalizace s vazbami. Algoritmus hledá aproximaci řešení obecné úlohy optimalizace s vazbami z podkapitoly 2.2.1.

Cílem metody je sestrojit v U_{ad} polyedr s K vrcholy a v jeho okolí nalézt minimum J patřící do U_{ad}. Přičemž velikost okolí se odvíjí od polohy vrcholů a těžiště polyedru.



Obr. 2.5.: Generace simplexu v průběhu iterací.

Na obrázku je zobrazen průběh generacemi simplexu v šesti iteracích. Je zde zřetelně viditelné postupné „ořezávání“. Vrcholy diamantu konvergují k nalezené aproximaci.

Průběh algoritmu má několik fází:

Počáteční nastavení

1)$K \geq N+1$ počet vrcholů simplexu (doporučuje se nastavit $K \approx 2N$ a $\alpha_{\max} \approx 1.3$),
N rozměr stavového prostoru a počet „jednoduchých“ lineárních omezení,
M počet složitějších omezení g_j.
Jednoduchá lineární omezení jsou dána pomocí reálných čísel h_i, s_i $(s_i < h_i)$ pro $i = 1, \ldots, N$, složitější omezení jsou dána pomocí funkcí $g_j \colon \mathbb{R}^N \to \mathbb{R}$ pro $j = 1, \ldots, M$.

2) Dále musíme zvolit bod $x^{(1)} = (x_1^{(1)}, x_2^{(1)}, \ldots, x_N^{(1)})^T$ splňující všech $2N + M$ omezení, t.j. $x^{(1)} \in U_{ad}$. A určit kladná reálná čísla $\alpha_{\min} < \alpha_{\max}$, kde $\alpha_{\max} \geq 1$.

Sestavení startovního polyedru $\{x^{(k)}\}_{k=1}^K \subset \boldsymbol{U}_{ad}$

3) Nejprve jsou generovány body $y^{(2)}, \ldots, y^{(K)}$ podle předpisu

$$y_i^{(k)} := s_i + r_{ik}(h_i - s_i)$$

pro $i = 1, \ldots, N$, $k = 2, \ldots, K$, kde $r_{ik} \in \langle 0, 1 \rangle$ je náhodné číslo (např. z rovnoměrného rozdělení).
Body $\{y^{(k)}\}_{k=1}^{K}$ splňují pouze podmínky $s_i \leq y_i \leq h_i$, ale slouží také k vytvoření bodů $\{x^{(k)}\}_{k=1}^{K}$ splňujících všechny podmínky.

4) Dále dosadíme za $k = 2, \ldots, K$ a zkoumáme, zda bod $y^{(k)}$ vyhovuje všem g-omezením, t.j. $\{g_j(y^{(k)})\}_{j=1}^{M} \leq 0$.
Pokud ano, položíme $x^{(k)} = y^{(k)}$ a pokračujeme dalším bodem $y^{(k+1)}$.

5) V opačném případě vypočteme těžiště t ze všech již známých bodů $\{x^{(l)}\}_{l<k}$, které všem podmínkám vyhovují,

$$t := \frac{1}{k-1} \sum_{l=1}^{k-1} x^{(l)}.$$

6) Následně hledáme nejmenší číslo $i = 1, 2, 4, 8, \ldots$ takové, že bod

$$t + \frac{1}{2^i}(y^{(k)} - t)$$

splňuje všechna omezení.
Jinými slovy, posouváme bod $y^{(k)}$ směrem k těžišti $t \in U_{ad}$.
(Díky předpokladu konvexnosti množiny U_{ad} stačí otestovat pouze splnění g-omezení.)
Pro nalezené i položíme $x^{(k)} = t + \frac{1}{2^i}(y^{(k)} - t)$ a pokračujeme bodem $y_{(k+1)}$.

Určení nového vrcholu

7) Nejprve vyčíslíme funkční hodnoty jednotlivých vrcholů $\{x^{(k)}\}_{k=1}^{K}$ a označíme vrchol $x_{\max}$ jako vrchol s maximální funkční hodnotou.

8) Poté spočteme α-reflexi bodu $x_{\max}$ vzhledem k těžišti t vypočtenému ze všech zbývajících vrcholů. Reflexi označíme x_r a platí:

$$x_r = t - \alpha(x_{\max} - t).$$

Protože bod x_r nemusí ležet v množině U_{ad}, budeme jej v takovém případě přesouvat směrem k těžišti t tak dlouho, dokud se nestane přípustným bodem.
Tedy v řadě určíme takové nejmenší číslo i, pro něž
$t - \frac{\alpha(x_{\max}-t)}{2^i}$ patří do U_{ad}.
Je důležité sledovat, zda hodnota $\frac{\alpha}{2^i}$ je větší než předepsaná hodnota $\alpha_{\min}$. Pokud hodnota tohoto parametru poklesne pod stanovenou hranici, může dojít ke „zhroucení“ simplexu. V takovém případě restartujeme celou proceduru.

9) Dále vypočteme funkční hodnotu nalezeného bodu $x_r \in U_{ad}$, a porovnáme, zda $J(x_r) < J(x_{\max})$.
Pokud ano, získali jsme lepší bod, vrchol $x_{\max}$ nahradíme bodem x_r a zopakujeme celý postup s novým simplexem.

10) V opačném případě opět posouváme bod x_r směrem k těžišti t tak dlouho, dokud nezískáme bod s menší funkční hodnotou. Konvexita U_{ad} zaručuje přípustnost takto získaného bodu.
Pokud je ovšem během posunu porušena podmínka na $\alpha_{\min}$, musíme celou proceduru restartovat.

11) Pokud vše proběhlo v pořádku, zopakujeme celý postup s novým simplexem.

Tento algoritmus byl naimplementován do programu Matlab a otestován.

2.2.3. Aplikace – Algoritmy zobecněné diferenciální evoluce

V této sekci se budeme věnovat algoritmům zobecněné diferenciální evoluce. Jedná se o jiný postup, jak se můžeme v optimalizačních úlohách s podmínkami vyrovnat s případy, kdy nové body nepadnou do oblasti přípustných řešení.

Autoři Kukkonen a Lampinen ve svém článku [14] konstruují několik evolučních algoritmů (GDE1, GDE2, GDE3) založených na úpravě standardně užívaných evolučních algoritmů pro úlohy bez podmínek. Využívají definice pojmu slabě podmíněné dominance, která slouží pro porovnávání jedinců v populaci.

Nyní označme obecnou úlohu definovanou v podkapitole 2.2.1. jako

$$J := \ \mathbb{R}^N \to \mathbb{R}(N \in \mathbb{N}) \text{ na množině } U_{ad} \tag{1}$$

$$U_{ad} := \ \{x \in \mathbb{R}^N;\, g_j(x) \leq 0 \text{ pro } j = 1, \ldots, M,\ s_i \leq x_i \leq h_i \text{ pro } i = 1, \ldots, N\}. \tag{2}$$

Definice 1. Řekneme, že v úloze (1) $\mathbf{x}_1$ slabě dominuje $\mathbf{x}_2$ a zapíšeme $\mathbf{x}_1 \preceq \mathbf{x}_2$, jestliže $J(\mathbf{x}_1) \leq J(\mathbf{x}_2)$. Řekneme, že v úloze (1) $\mathbf{x}_1$ dominuje $\mathbf{x}_2$ a zapíšeme $\mathbf{x}_1 \prec \mathbf{x}_2$, jestliže $J(\mathbf{x}_1) < J(\mathbf{x}_2)$.

Obdobně zavedeme pojem podmíněné dominance „$\prec_c$“ a slabé podmíněné dominance „$\preceq_c$“.

Definice 2. Řekneme, že v úloze (1), (2) bod $\mathbf{x}_1$ podmíněně dominuje $\mathbf{x}_2$ a zapíšeme $\mathbf{x}_1 \prec_c \mathbf{x}_2$, pokud bude platit jedna z následujících podmínek:

(i) $\mathbf{x}_1$ a $\mathbf{x}_2$ nejsou přípustná řešení úlohy (2) a $\mathbf{x}_1 \prec \mathbf{x}_2$ v úloze (1),

(ii) $\mathbf{x}_1$ je přípustné řešení úlohy (2) a $\mathbf{x}_2$ není přípustné řešení úlohy (2),

(iii) $\mathbf{x}_1$ a $\mathbf{x}_2$ jsou přípustná řešení úlohy (2) a $\mathbf{x}_1 \prec \mathbf{x}_2$ v úloze (1).

Definice 3. Řekneme, že bod $\mathbf{x}_1$ podmíněně slabě dominuje $\mathbf{x}_2$ a zapíšeme $\mathbf{x}_1 \preceq_c \mathbf{x}_2$, pokud platí jedna z podmínek:

(i) $\exists k \in \{1, \ldots, K\}\colon g_k(\mathbf{x}_1) > 0 \wedge \forall k \in \{1, \ldots, K\}\colon g'_k(\mathbf{x}_1) \leq g'_k(\mathbf{x}_2)$,

(ii) $\forall k \in \{1, \ldots, K\}\colon g_k(\mathbf{x}_1) \leq 0 \wedge \exists k \in \{1, \ldots, K\}\colon g_k(\mathbf{x}_2) > 0$,

(iii) $\forall k \in \{1, \ldots, K\}\colon g_k(\mathbf{x}_1) \leq 0 \wedge g_k(\mathbf{x}_2) \leq 0 \wedge J(\mathbf{x}_1) \leq J(\mathbf{x}_2)$,

kde $g'_k(\mathbf{x}) = \max\{g_k(\mathbf{x}), 0\}$.

Výše zadefinované druhy dominance lze využít v návaznosti na algoritmus diferenciální evoluce uvedený v sekci 2.1.1.

Dále se budeme zabývat algoritmem zobecněné diferenciální evoluce. Tento heuristický algoritmus vychází z metody uvedené v 2.1.1., ale na rozdíl od perturbace používá k zajištění přípustnosti generovaných bodů podmíněné dominance. Kritérií pro výběr nové generace je však několik. [14]

Algoritmy GDE1, GDE2 a GDE3

Pro vytváření nových populací v rámci zobecněné DE lze využít následující kritéria [14].

A) Algoritmus GDE1

$$\mathbf{x}_{i,G+1} = \begin{cases} \mathbf{u}_{i,G}, & \text{jestliže } \mathbf{u}_{i,G} \preceq_c \mathbf{x}_{i,G} \\ \mathbf{x}_{i,G} & \text{jinak,} \end{cases}$$

kde $\mathbf{x}_{i,G}$ je náhodně vygenerovaná počáteční populace jedinců a $\mathbf{u}_{i,G}$ je generace vzniklá užitím DE na původní populaci, tj. užitím operací křížení, mutace a selekce. Kritérium GDE1 tedy udává, kteří jedinci mají být v nové populaci zachováni s využitím slabě podmíněné dominance.

B) Algoritmus GDE2

$$\mathbf{x}_{i,G+1} = \begin{cases} \mathbf{u}_{i,G}, & \text{jestliže } \mathbf{u}_{i,G} \preceq_c \mathbf{x}_{i,G} \vee (\forall k \in \{1,\ldots,K\}: \\ & g_k(\mathbf{u}_{i,G}) \leq 0 \ \wedge \neg(\mathbf{x}_{i,G} \prec \mathbf{u}_{i,G}) \ \wedge d_{\mathbf{u}_{i,G}} \geq d_{\mathbf{x}_{i,G}}), \\ \mathbf{x}_{i,G} & \text{jinak,} \end{cases}$$

kde d_i je vzdálenost i-tého řešení k nejbližšímu jinému řešení v populaci. Používá se „crowding distance“ d, která je průměrnou vzdáleností k jedincům z množiny tvořené zvoleným procentem nejbližších jedinců. Pokud se nevyskytuje žádné nedominované třídění, potom shlukování probíhá celou populací.

C) Algoritmus GDE3

V tomto algoritmu dochází ke změně DE nejen v otázce výběru, ale také při porovnávání přípustnosti a nedominovaných řešení. V případě porovnávání přípustnosti a nedominantních řešení se ukládají oba vektory. Díky tomu je na konci generování populace delší než původně. Před vytvářením další generace je velikost populace redukována pomocí nedominovaného třídění a očištění, založeném na diverzitě zachování tak, aby byli zachováni ti nejlepší kandidáti na řešení.

Vstupy: dimenze$D$, maximální počet generací $G_{\max}$, počet jedinců v populaci $NP \geq 4$, $F \in (0, 1_+]$, $CR \in [0, 1]$, počáteční meze $\mathbf{x}^a, \mathbf{x}^b$.

Inicializace: $\forall i \leq NP \wedge \forall j \leq D$: $\mathbf{x}_{i,i,G} = \mathbf{x}_j^a + \text{rand}[0,1] * (\mathbf{x}_j^b - \mathbf{x}_j^a)$, $i = \{1, 2, \ldots, NP\}$, $j = \{1, 2, \ldots, D\}$, $G = 0$, $a = 0$.

Zde generujeme původní populaci s využitím horních a dolních přípustných mezí a koeficientu z rovnoměrného rozdělení.
Zdrojový kód algoritmu [14]:

while $G < G_{\max}$

Mutate and recombine

$r_1,\ r_2,\ r_3 \in \{1, 2, \ldots, NP\}$

zvolíme náhodně, vzájemně rozdílné a různé od i

$j_{\text{rand}} \in \{1, 2, \ldots, D\}$ náhodně zvolíme $\forall i\ \forall j \leq D$,

$$\mathbf{u}_{i,j,G} = \begin{cases} \mathbf{u}_{i,r_3,G} + F(\mathbf{x}_{j,r_1,G} - \mathbf{x}_{j,r_2,G}), & \text{jestliže } \text{rand}_j\ \langle 0,1) < CR \vee j = j_{\text{rand}}, \\ \mathbf{x}_{j,i,G} & \text{jinak} \end{cases}$$

Select:

$$\mathbf{x}_{i,G+1} = \begin{cases} \mathbf{u}_{i,G}, & \text{jestliže } \mathbf{u}_{i,G} \preceq_c \mathbf{x}_{i,G}, \\ \mathbf{x}_{i,G} & \text{jinak}, \end{cases}$$

Set:

$$n = n + 1,\ \mathbf{x}_{NP+n,G+1} = \mathbf{u}_{i,G}, \text{ jestliže } \begin{cases} \forall k\colon g_k(\mathbf{u}_{i,G}) \leq 0 \\ \wedge\ \mathbf{x}_{i,G+1} = \mathbf{x}_{i,G} \\ \wedge\ \neg\{\mathbf{x}_{i,G} \prec \mathbf{u}_{i,G}\} \end{cases}$$

while $n > 0$

Select $\mathbf{x}\mathcal{P} = \{\mathbf{x}_{1,G+1}, \mathbf{x}_{2,G+2}, \ldots, \mathbf{x}_{NP+n,G+1}\}$,

$\mathbf{x} \in$ poslední nedominované množiny $\mathcal{P}$

$\wedge\ \mathbf{x}$ je nejvíce naplněná nedominovaná množina.

Remove $\mathbf{x}$

$n = n - 1$

end

$G = G + 1$

end

Zakomponováním těchto pravidel do heuristických algoritmů založených na principech diferenciální evoluce získáme stochastický algoritmus pro optimalizaci s podmínkou, kterých je v současné době velmi malé množství. Třída heuris-

tických algoritmů je obecně poměrně široká, všechny jsou ale určeny pouze pro optimalizaci bez podmínek.

Ještě zde naznačíme,jak by bylo nutné pozměnit například algoritmus Rypoš pro řešení podmíněné optimalizace. Původní algoritmus bychom doplnili o pasáž obsahující testování přípustnosti řešení s využitím jednoho z kritérií GDE1, GDE2, nebo GDE3 [14]. Testování slabě podmíněné dominance by se provádělo v předposledním kroku výše naznačeného algoritmu.

Popis by vypadal takto:

- Nyní máme k dispozici 20 jedinců, ale pro tak velkou kolonii by v okolí nebyl dostatek hlíz, a proto musí 10 jedinců zahynout. Přežijí ti nejlepší ve smyslu slabě podmíněné dominance. (Zde by byl proveden test dominance dle jednoho z kritérií.) Královna již splnila svou funkci a také kolonii opouští. Máme tedy opět kolonii nejlepších 10 rypošů složenou z jedinců původní populace i z přeživších potomků (světle modrá kolečka).

Sestavení takovéhoto algoritmu je jistě možností pro získání nových zkušeností a doporučuji jej jako možný postup při rozvoji této metody. Heuristické algoritmy jsou známé pouze pro optimalizační úlohy bez podmínek, proto by sestrojení výše navrženého algoritmu bylo jistě cenným přínosem pro rozšíření aplikability stochastických algoritmů.

2.2.4. Testování algoritmu ACM

Pro testování algoritmu ACM jsme zvolili úlohu hledání globálního minima známé testovací funkce, Rosenbrockova sedla [2].

Graf této funkce vytvořený v programu Matlab, funkční předpis, definiční obor a hodnotu globálního minima si můžete prohlédnout na Obr. 2.4. v kapitole 2.1.3.

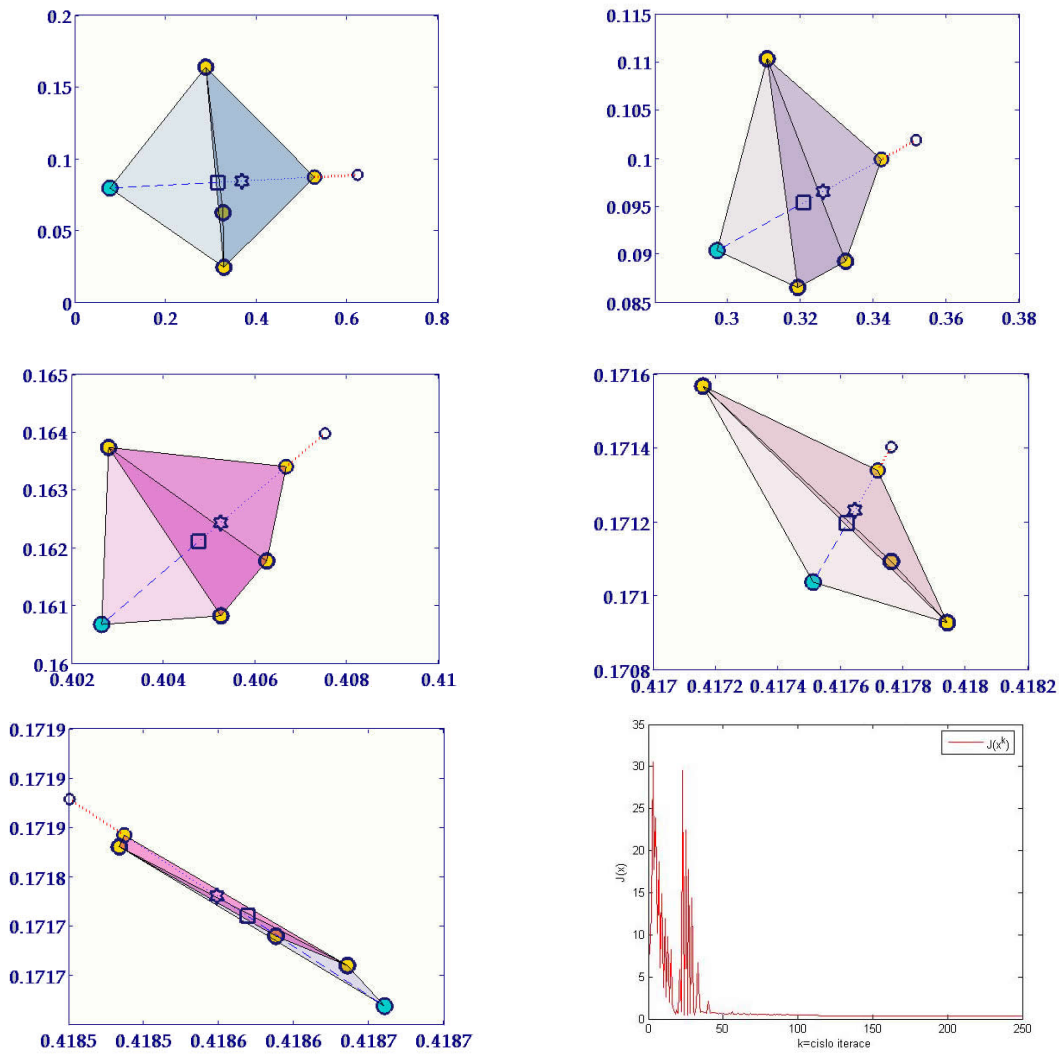
K takto zadefinované úloze jsme doplnili podmínku ve tvaru:

$$\mathbf{x_2} - \frac{3}{4}\mathbf{x_1}^2 - 4 \leq 0.$$

Jednoduchá omezení byla tvořena hranicemi definičního oboru této funkce. Zvolili jsme $N = 2$, tedy počet vrcholů $K = 4$.

Algoritmus ACM se spouští pomocí souboru spust_ACM_RO.m, postup algoritmu je popsán v souboru comp_met_RO.m a v souboru rosen.m je zapsána testovací úloha. V rámci jednoho spuštění probíhá 250 iterací. Algoritmus kreslí simplexy z první až čtvrté a každé padesáté iterace. Dále získáme také graf průběhu cenového funkcionálu J.

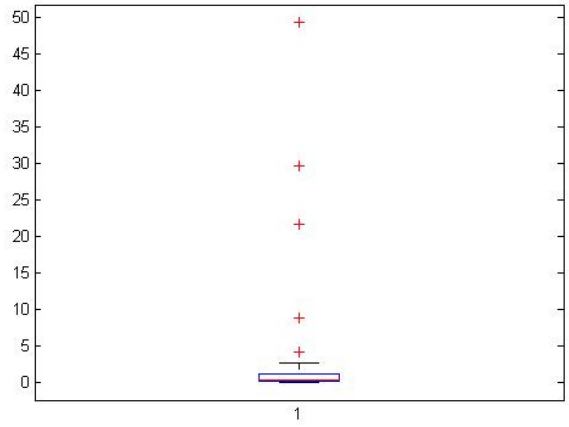
Následující sekvence grafů zobrazuje vývoj simplexů v iteracích. Na obrázku je vždy světlejší barvou znázorněn výchozí simplex z předchozí iterace (popř. startovní simplex). Jeho vrcholy jsou značeny modře. V další fázi iterace je vyhodnocen vrchol s maximální funkční hodnotou, z ostatních vrcholů je spočteno těžiště (čtvereček) a pomocí něj je prováděna α–reflexe bodu s největší funkční hodnotou (modré kolečko), dokud nový bod nepadne do množiny přípustných řešení. Bílé kolečko naznačuje maximálně vzdálený bod, kam mohl být vrchol s maximální funkční hodnotou překlopen s hodnotou $\alpha = 1,3$. Z tohoto bílého bodu je červeně naznačena cesta zpět k těžišti a žluté kolečko na konci této červené linie značí výslednou α–reflexi vrcholu x_{max} v množině přípustných řešení. Vrcholy nového simplexu jsou značeny žlutě a nový simplex je vybarven tmavším odstínem. Jeho celkové těžiště je reprezentováno hvězdičkou.



Obr. 2.6. Vývoj simplexů v každé 50. iteraci a průběh cenového funkcionálu J.

V posledním okně Obr. 2.6. vpravo vidíme průběh cenového funkcionálu J během 250 iterací.

Tato úloha byla otestována celkem 50–krát. Hodnoty výsledných aproximací byly zaznamenány do tabulky Testování ACM.xls a byly použity pro vytvoření box–diagramu pro určení průměrné hodnoty aproximace. Tento diagram je znázorněn na Obr. 2.7., střední hodnota nalezených aproximací je 0,23 a směrodatná odchylka má hodnotu 8,57.



Obr. 2.7. Box–diagram nalezených aproximací.

2.3. Kombinatorická optimalizace

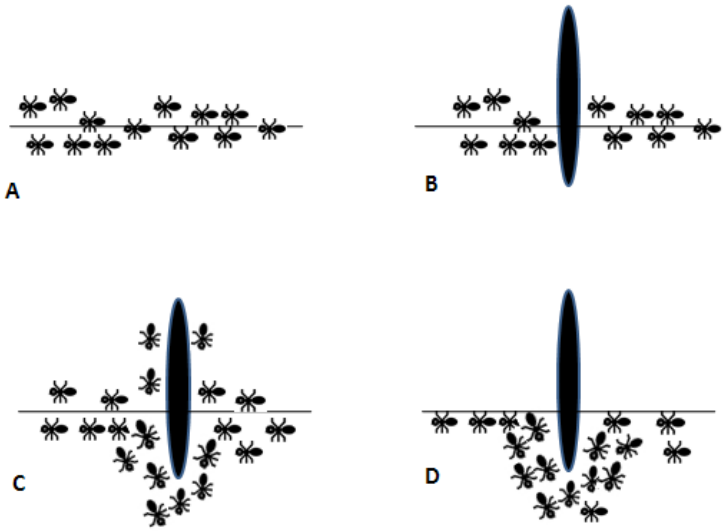
2.3.1. Mravenčí kolonie

Mravenci se řadí mezi společenstva se sociálním uspořádáním. Při zkoumání kolektivního chování takovýchto systémů byly objeveny zákonitosti v chování nazývané jako rojová inteligence. Rojová inteligence je typická pro společenstva, kde jednotliví jedinci komunikují mezi okolím a sebou navzájem. Komunikace může probíhat přímo, nebo působením na okolí. Vzorce zakódované v chování jednotlivců odhalují podstatu chování celého společenstva. Příkladem mohou být mravenci, včely nebo bakterie. Algoritmy inspirované takovouto samoorganizací jsou úspěšné při řešení náročných optimalizačních úloh a řadí se mezi heuristické algoritmy.

2.3.2. Biologická inspirace

Inspirací tohoto algoritmu bylo rojové chování u mravenčích kolonií. Mravenci se řadí mezi sociální hmyz a podstatou jejich globálního chování je snaha o zachování kolonie. Komunikace je zprostředkovávána pomocí feromonu, který mravenec neustále vylučuje. Každý jedinec může ovlivňovat koncentraci, s jakou feromon uvolňuje. Pokud se několik průzkumníků vydalo hledat potravu a

ostatní je budou následovat, „stopaři“ se vydají po trase s největší koncentrací feromonu. Ta povede nejrychleji k potravě, potrava je nutná pro udržení kolonie, což je životním účelem každého jednotlivého mravence. Obrázek 2.8. zobrazuje chování mravenců v případě, že se na cestě za potravou objeví překážka a formu předávání informací.



Obr. 2.8. Principy mravenčího hledání potravy.

- A – Mravenci hledají cestu mezi hnízdem a zdrojem potravy.
- B – Objevila se překážka na cestě: mravenci se rozhodují, zda se vydají vpravo nebo vlevo. Pravděpodobnost obou možností je stejná.
- C – Feromon je rychleji ukládán na kratší cestu.
- D – Všichni mravenci vybírají kratší cestu.

V mravenčí kolonii si nejsou všichni jedinci rovnocenní. Je zde vytvořeno hierarchické uspořádání. Na vršku celého společenství se nachází královna, dále se zde vyskytují plodní samci, dělnice a tzv. velké dělnice s extrémně vyvinutými kusadly, které tvoří „armádu“ kolonie. V každé kolonii nalezneme vždy minimálně jednu královnu a několik dělnic.

Mravence bychom objevili téměř všude, většinu bychom ale našli v oblastech s teplým podnebím a vysokou vlhkostí vzduchu. Významnou živočišnou skupinu tvoří v tropických deštných lesích, kde tvoří až 15% celkové živočišné biomasy. V roce 2006 bylo známo 11844 mravenčích druhů. Hnízda mravenců můžeme hledat pod zemí, ve stromech, rostlinách, mezi listy stromů nebo volně v přírodě. Běžný mravenec lesní se může dožít věku 7 až 10 let a mravenčí královna dokonce 10 až 20 let. [16]

Chování mravenců bylo podrobeno četnému zkoumání. V 50. a 60. letech 20. století francouzský entomolog Pierre-Paul Grassé pozoroval, jak některé druhy termitů reagují na určité stimuly. Pozoroval, že reakce na tyto podněty jsou společné pro termity, mravence a další hmyz žijící v koloniích. [16]

Byl zaveden pojem stigmergie, vyjadřující typ komunikace, kdy jsou dělníci stimulováni výkony, kterých mají dosáhnout. Mezi charakteristické rysy stigmergie patří:

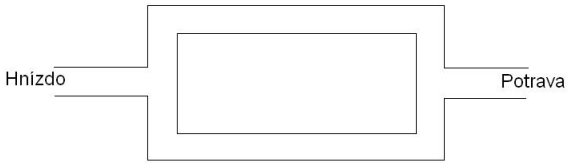
- Stigmergie je nepřímá, nesymbolická forma komunikace prostřednictvím životního prostředí – hmyz reaguje na změny okolního prostředí.
- Stigmergická informace je fixovaná na místo. Může ji získat pouze jedinec, který dané oblast navštíví.

Tento princip předávání informace je v mravenčích koloniích využíván při hledání cesty mezi hnízdem a potravou. Díky informaci uložené v koncentraci feromonu jsou mravenci schopni přepravit potravu do svého hnízda pozoruhodně efektivně.

Byla provedena řada experimentů, ve kterých bylo prokázáno, že výběr cesty ke zdroji potravy je skutečně založen na samoorganizaci. Proces ukládání feromonu byl předmětem tzv. „experimentu s dvojitým mostem“, který byl prováděn Deneubourgem a jeho kolegy. [16] a [17]

Experiment s binárním mostem

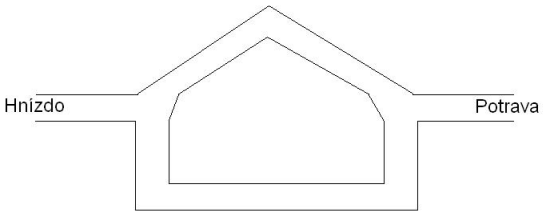
V průběhu tohoto pokusu uvažujeme dvě cesty A a B vedoucí z hnízda k potravě. Obě trasy jsou stejně dlouhé a na počátku po nich neprošel žádný mravenec - není zde uložen žádný feromon. Pravděpodobnost výběru každého z ramen byla shodná. Vlivem náhodných fluktuací došlo k situaci, kdy si větší počet mravenců vybral jednu z cest. Ukládání feromonu způsobilo, že tato cesta byla mravenci postupně naprosto preferována.



Obr. 2.9. Schéma experimentu s binárním mostem.

Experiment s mosty různé délky

Experiment s binárním mostem můžeme lehce modifikovat na situaci s cestami A a B, které nemají stejnou délku. V tomto případě je nejčastěji zvolena kratší trasa na cestě k potravě i zpět. Tím, že se mravenec po této kratší spojnici i vrací, nanáší feromon dvakrát, zvyšuje koncentraci cesty a tím sděluje ostatním jedincům, aby také zvolili tuto cestu. Experimenty prokázaly, že šance vybrání kratšího ramene se zvyšuje s poměrem délek obou ramen.



Obr. 2.10. Experiment s nestejnou délkou mostů.

Pokud je kratší cesta přidána do systému později, mravenci většinou zůstanou (uváznou) v původní delší trati kvůli silné koncentraci feromonu z období, kdy kratší cesta ještě nebyla k dispozici. Existuje ale druh mravenců, který v takovém případě reaguje jinak. Ve chvíli, kdy se ocitne v polovině delší cesty, uvědomí si, že směřuje kolmo k potřebnému směru, a vrátí se zpět. Zde je mimo kolektivní zkušenosti předávané feromonem využita navíc ještě směrová paměť.

2.3.3. Problém obchodního cestujícího

Uvedené experimenty byly zaměřeny na nalezení nejkratší cesty mezi hnízdem a potravou. Zobecníme-li tuto situaci na minimalizaci trasy mezi několika místy v prostoru, dostáváme úlohu Obchodního cestujícího (TSP – Traveling Salesman Problem) [18].

Řešením problému obchodního cestujícího se matematici zabývají již déle než dvě stě let. První zmínky o TSP spadají do roku 1800. Irský matematik W. R. Hamilton a Brit T. P. Kirkmanem se zabývali řešením Hamiltonovy hry Icosian [18]. Účelem bylo najít nejkratší cestu přes 20 bodů použitím pouze určitých spojnic.

Zadání úlohy obchodního cestujícího je následující: Je dána množina N měst a matice jejich vzájemných vzdáleností. Chceme určit takovou cestu mezi městy (pořadí), aby byly splněny podmínky:

1) Každé město je navštíveno právě jednou.

2) Algoritmus končí návratem do výchozího stanoviště.

3) Trasa mezi městy je minimální.

Časová náročnost této úlohy je exponenciální vzhledem k počtu měst.

2.3.4. Aplikace – Algoritmus Mravenčích kolonií (ACO)

Na základě v přírodě odpozorovaných vlastností byl vytvořen umělý model mravence. Také umělí mravenci mezi sebou komunikují na základě feromonových značek. Množství feromonu je řízeno funkcí kvality řešení a s časem „vyčichá“. V případě potřeby, se rozhodují na základě pravděpodobností. Pohybují

se v diskrétním prostoru, pamatují si své předchozí činnosti a nejsou slepí. Hledání optima je iterační proces. Mravenec se zároveň řídí heuristickou informací o řešeném problému a zkušenostmi předchozích mravenců, které jsou kódované ve feromonové stopě. Schopnost uchovávat si v paměti nejlepší dosažená řešení vede k lepším výsledkům [16].

V dalším textu je zpracováno, jak lze pomocí algoritmu ACO (Ant Colony Optimization) řešit problém obchodního cestujícího [19]. Nejprve pro všechny dvojce mest C_i , C_j inicializujeme v čase 0 hodnotu intenzity feromonu $\tau_{i,j}$ malým číslem ϵ, tedy $\tau_{i,j}(0) = \epsilon$.

Princip algoritmu ACO lze popsat následovně [19]:

for t = 1 .. maximální počet iterací **do**

repeat

Vybere náledující město C_j s pravděpodobností

$P_{i,j}^k(t) = \frac{(\tau_{i,j})^\alpha \ (\eta_{i,j})^\beta}{\sum_{h \in allowed_k(t)} \ (\tau_{i,h})^\alpha \ (\eta_{i,h})^\beta}$,

jestliže k-tý mravenec může v daném kroku použít hranu (i,j).

Jinde je pravděpodobnost nulová.

η je parametr viditelnosti reprezentovaný převrácenou hodnotou vzdálenosti mezi městy a $allowed_k$ je množina měst, kterou k-tý mravenec v čase t doposud nenavštívil.

until

k-tý mravenec neprošel všemi městy.

Stanovení nové hodnoty feromonu pro všechny dvojice měst podle vzorce

$\tau_{i,j}(t+n) = (1-\rho) \ \tau_{i,j}(t) + \Delta\tau_{i,j}(t, t+n)$,

kde $\Delta\tau_{i,j} = \sum_{k=1}^{m} \Delta\tau_{i,j}^k(t, t+n)$

$\Delta\tau_{i,j}^k(t, t+n) = \frac{Q}{L_k}$, jestliže se k-tý mravenec vydal z města C_i do města C_j.

Q je konstanta a L_k je celková trasa mravence.

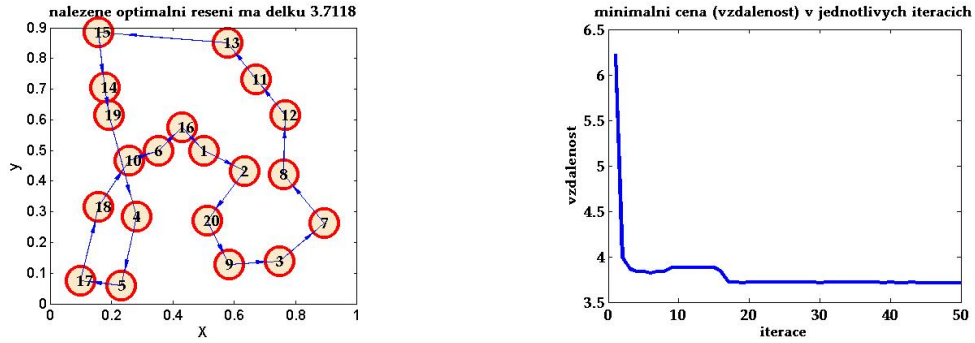
V jiném případě je intenzita feromonu nulová.

ρ je koeficient rychlosti vyprchání feromonu $\rho \in \langle 0, 1 \rangle$.

end

Jako příklad uvádíme Obr. 2.11. výstupu algoritmu Mravenčích kolonií, který byl vytvořen Bc. Lenkou Bláhovou v rámci bakalářské práce na téma Matematické modely v logistice [21]. Předlohou tohoto algoritmu byl zdroj [19] a [15].

Algoritmus hledá nejkratší cestu přes množinu dvaceti měst. Ta jsou situovaná do čtverce o rozměrech 1×1, což reprezentuje 1000 km. Při tomto spuštění byla mezi městy nalezena cesta o délce 3,7118 km.



Obr. 2.11. Výsledná trasa nalezená algoritmem ACO.

Stejně jako existují testovací úlohy pro optimalizaci bez podmínek [2], známe podobné metody také pro testování algoritmů řešících problém obchodního cestujícího. Při dodržení určitých pravidel lze dokonce odhadnout očekávanou délku trasy mezi městy [24].

Následující metodu odhadu lze použít v případech, kdy jsou města generována náhodně z rovnoměrného rozdělení a vzdálenost mezi nimi určuje Euclidovská metrika. Potom pro minimální délku trasy L^* platí

$$L^* = k\sqrt{n \cdot R},$$

kde n charakterizuje počet měst a R je plocha čtverce obsahujícího množinu všech měst. Hodnota empirické konstanty k se pohybuje okolo hodnoty 0,75. Podle Helda–Karpa [25] je ohraničena $\sqrt{n}$ a pro n–random Euclidian TSP a $n > 100$ platí

$$k = 0{,}70805 + \frac{0{,}52229}{\sqrt{n}} + \frac{1{,}31572}{n} - \frac{3{,}07474}{n\sqrt{n}}.$$

Bonomi a Lutton [23] doporučují volit $k = 0{,}749$.

Tento odhad může posloužit k testování úspěšnosti nových algoritmů.

3. Závěr

Cílem diplomové práce bylo zvolit zástupce biologicky inspirovaných algoritmů a provést experimenty vedoucí k ohodnocení jejich úspěšnosti. Zabývali jsme se čtyřmi odlišnými úlohami – neuronovými sítěmi, optimalizacemi s podmínkou i bez podmínky a kombinatorickou optimalizací.

V oblasti neuronových sítí dochází v současné době k velkému rozvoji a začínají se prosazovat v nejrůznějších odvětvích. V první kapitole jsou zpracována doporučení, jak správně nastavit neuronové sítě tak, aby bylo dosaženo optimálních výsledků. Zaměřili jsme se na algoritmy vhodné pro shlukovou analýzu, Kohonenovy samoorganizující se mapy [8] a algoritmus Neuronový plyn [1].

Přestože jsou v současné době k dispozici komerční softwary neuronových sítí doplněné výukovými programy, jejich finanční náročnost je však velmi vysoká. Proto byly v rámci diplomové sítě oba uvedené algoritmy sestaveny v programu Matlab. Protože cílem bylo porovnání úspěšnosti těchto algoritmů, byla pro testování zvolena jednotná úloha redukce dat. Použili jsme Fisherovu datovou množinu irisů [10] obsahující původně 150 zástupců.

Algoritmus Kohonenových samoorganizujícíh se map se pro tuto úlohu velmi osvědčil. Z výsledků testování vyplývá, že určil rozčlenění dat nejčastěji do tří shluků. Protože jsou data tvořena třemi druhy irisů, shoduje se tento výsledek s naším očekáváním. Četnost vzniku dvou shluků tvoří také velký podíl na celkovém počtu pokusů. Důvodem takovéhoto chování sítě může být další charakteristika, kterou o datech víme. A to, že zatímco druh Iris setosa lze oddělit od ostatních, zbylé dva druhy nejsou linárně separovatelné. Fakt, že byly samostatně odděleny právě irisy druhu setosa, můžeme potvrdit z dalšího z výstupů. Na grafu četnosti jednotlivých kombinací shluků (Obr. 1.12.) je patrné, že z izolovaných shluků je nejčetněji vybírán právě Iris setosa a druhý nejčetnější je shluk tvořen irisy typu virginica a versicolor. Z těchto ukazatelů můžeme usoudit, že úspěšnost algoritmu Kohonen je velmi dobrá.

Stejnou úlohu i nastavení jsme použili pro testování algoritmu Neuronový plyn. Zde pozorujeme, že algoritmus data nejčastěji sloučil do dvou skupin. Tato

skutečnost opět poukazuje na možný problém se separací Iris virginica a versicolor. Tento fakt ale již není jednoznačně potvrzen strukturou shluků. Existují zde čtyři významné skupiny kombinací irisů s procentuálním podílem na celku v rozmezí 18% – 26%. Podíl izolovaných složek Irisu setosa a virginica je velmi vysoký, přesto je převýšen četností kombinace všech tří druhů irisů. Tento jev napovídá tomu, že nesprávné zařazení nevzniklo pouze kvůli problému s neseparabilitou.

Z porovnání výsledků provedených experimentů je patrné, že algoritmus Kohonenových samoorganizujících se map lépe odhadl a zachoval datovou strukturu. A naprosto potvrdil námi očekávané vlastnosti vstupních dat. Naproti tomu výsledky algoritmu Neuronový plyn jsou v tomto smyslu nejasné. Algoritmus měl problémy i s rozlišením lineárně separabilních dat.

Ve druhé části diplomové práce jsme se věnovali různým odvětvím heuristické optimalizace. V podkapitole Evoluční algoritmy v optimalizačních úlohách bez podmínky byl představen algoritmus diferenciální evoluce [11], jehož principy byly podkladem pro vytvoření vlastního algoritmu Rypoš [11] určeného pro hledání globálního minima vícekriteriálních funkcí. Inspirací tohoto algoritmu bylo chování australského hlodavce Rypoše lysého. Úspěšnost algoritmu byla vyhodnocena pomocí tzv. testovacích funkcí [2]. Z dat vzniklých testováním byl vytvořen krabičkový diagram, ze kterého plyne, že výborných výsledků dosahuje algoritmus pro DeJongovu funkci, protože medián se téměř rovná nule, hodnotě globálního minima DeJongovy funkce, a rozpětí je také velmi malé. Také pro Rosenbrockovo sedlo a Griewangkovu funkci jsou výsledky velmi uspokojivé. U ostatních funkcí jsou větší odchylky způsobeny vyšší složitostí funkce. Algoritmus se velmi dobře vypořádal také s problémem uváznutí v lokálním extrému, který je řešen dvojí možností, jak si královna může zvolit hnízdo. Dochází tak k oscilaci a zamezí se případnému uváznutí.

Na tuto část tématicky navazuje kapitola představující metody optimalizace s podmínkami. Jako zástupce neživé přírody je zde popsán algoritmus Complex method. Jeho principem je vytváření diamantových struktur (simplexů) v oblasti přípustných řešení a jejich postupná modifikace a „ořezávání“ v průběhu

iterací. Tato metoda byla implementována do programu Matlab a otestována pro Rosenbrockovu testovací funkci [2] za dodání podmínky ve tvaru nerovností. Z dat získaných experimentem jsme sestavili box-diagram a zjistili jsme základní statistické charakteristiky. Hodnota mediánu je 0.23, směrodatná odchylka je vlivem odlehlých pozorování vyšší a má hodnotu 8.57. Algoritmů pro podmíněnou optimalizaci je velice málo. Proto je tato aplikace velice významná a data zde zjištěná mohou posloužit pro porovnání s aplikací vytvořenou při dalším rozvoji této práce. Pro srovnání doporučuji sestrojit zobecněný algoritmus Rypoš, který lze s využitím slabě podmíněné dominance upravit pro optimalizaci s podmínkou. Doporučení, jak takové zobecnění sestrojit jsou uvedena v sekci 2.2.3.

Poslední část diplomové práce byla věnována kombinatorické optimalizaci. Úkolem bylo nalézt biologicky inspirovaný algoritmus pro řešení úlohy obchodního cestujícího [18]. Zvolili jsme algoritmus Mravenčích kolonií [15], jehož struktura se nejvíce shoduje se zadaným problémem. Byla prostudována analogie s biologickou mravenčí kolonií a zpracován princip metody umělých Mravenčích kolonií. Problém obchodního cestujícího je stále aktuální a zajímavý. Proto musíme hledat a rozvíjet metody jako je algoritmus Mravenčích kolonií, které nám umožňují nalézt přijatelnou aproximaci. Snaha o vyřešení tohoto problému je podporována také finančně, momentálně je vypsána odměna pro úspěšného řešitele problému nazvaného Mona Lisa TSP [18].

V rámci této práce byly shrnuty důležité poznatky o vybraných biologicky inspirovaných algoritmech. Protože tyto metody nejsou součástí běžně přístupných softwarů, byly naprogramovány čtyři z uvedených algoritmů, které byly dále použity pro provedení experimentů, jejichž vyhodnocení bylo cílem této práce. Jedná se o nové, moderní a efektivní metody použitelné i v okamžicích, kdy jsou klasické postupy neefektivní. Jsou tedy cenným doplněním běžných metod shlukové analýzy a optimalizace s podmínkami i bez podmínek.

Práce zároveň poskytuje prostor pro další rozvoj a studium uvedené problematiky. Doporučuji sestrojení zobecněné verze původního algoritmu Rypoš [11] a porovnání jeho úspěšnosti s algoritmem Complex Method [13] vytvořeným a

otestovaným v této práci. Další možností je využití algoritmu Mravenčích kolonií pro řešení aktuálního problému Mona Lisa TSP [18].

Literatura

[1] Řezanková H., Húsek D., Snášel V.: Shluková analýza dat. Kamil Mařík - Professional Publishing, Praha, 2007.

[2] Zelinka I.: What is it SOMA, http://www.ft.utb.cz/people/zelinka/soma/ [online 3.4. 2010].

[3] Zelinka I.: Umělá inteligence I.: neuronové sítě a genetické algoritmy, VUTIUM, Brno , 1998.

[4] McCulloch W.S., Pitts W.: A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics, 5:115-133, 1943.

[5] Rosenblatt F.: The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Psychological Review, Vol 65, 386-408, 1958.

[6] Hebb D.O., The Organization of Behavior, Wiley, New York, str. 62, 1949.

[7] Šnorek M., Jiřina M.: Neuronové sítě a neuropočítače, ČVUT, Praha, 1996.

[8] Hebák P. a kol., Vícerozměrné statistické metody(3), Informatorium, Praha, 2007.

[9] Biehl M., Ghosh A., Hammer B.: Learning vector quantization: The dynamics of winner-takes-all algorithms, Neurocomputing, 69, 660-670, 2006.

[10] Encyklopedie Wikipedia, Iris flower data set, http://en.wikipedia.org/wiki/Iris_flower_data_set, [online 3.4. 2010].

[11] Koděrová L.: Heuristiky, bakalářská práce, UPOL, Přírodovědecká fakulta, Olomouc, 2008.

[12] Nelder J.A., Mead R.: A simplex method for function minimization. Computer Journal,7: 308–313, 1965.

[13] Rao, S.S.: Optimization. Theory and Application. New Delphi, Willey Eastern Limited, 1989, ISBN 0852267568.

[14] Kukkonen, S., Lampinen, J.: Generalized Differential Evolution for General Non-Linear Optimization.

[15] Dorigo M., Maniezzo V., Coloni A.: The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics - Part B, 1996.

[16] Němec M.: Optimalizace pomocí mravenčích kolonií, shttp://www.milosnemec.cz/clanek.php?id=78, [online 3.4. 2010].

[17] Deneubourg J.L., Aron S., Goss S., Pasteels J.M.:
The selforganizing exploratory pattern of the Argentine ant, Journalof Insect Behavio, 1990.

[18] The Traveling Salesman Problem,
http://www.tsp.gatech.edu/history/index.html, [online 3.4. 2010].

[19] Michalewicz Z., Fogel D.B.: How to Solve It: Modern Heuristics, Second edition, Springer-Verlag Berlin Heideberg New York, 1998.

[20] Doroto M., Gambardella M.L.: Ant colonies for the traveling salesman problem, IRIDIA, Université Libre de Bruxelles, Belgium, 1996.

[21] Bláhová L.: Matematické modely v logistice, bakalářská práce, UPOL, Přírodovědecká fakulta, Olomouc, 2008.

[22] Babak A., Mohammad F.: Integraion of self organizing feature maps and honey bee mating optimization algorithm for market segmentation, Journal of Theoretical and Applied Information Technology, 70–84, 2007.

[23] Bonomi E., Lutton J.L.: The n-city travelling salesman problem: statistical mechanics and the Metropolis algorithm. SIAM Rev. 26, 1984.

[24] Simchi-Levi D., Bramel J., Chen X.: The Logic of Logistics, Springer–Verlag, New York, 2004.

[25] Held M., Karp R.M., The traveling-salesman problem and minimum spanning trees: part II, Mathematical Programming 1, 1971.