VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

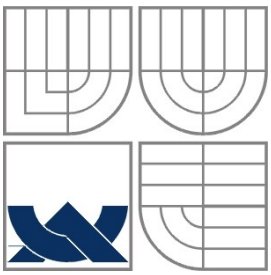
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ MODEL S UČENÍM (LEM) PRO OPTIMALIZAČNÍ ÚLOHY

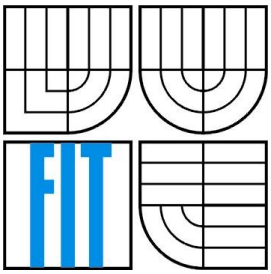
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Martin Weiss
AUTHOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ MODEL S UČENÍM (LEM) PRO OPTIMALIZAČNÍ ÚLOHY

LEARNABLE EVOLUTION MODEL (LEM) FOR OPTIMIZATION PROBLEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Martin Weiss

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. Josef Schwarz Csc.

BRNO 2011

Abstrakt

Numerická optimalizace multimodálních či jinak netriviálních funkcí se stále drží blízko středu pozornosti výzkumníků v této oblasti. Jednou ze slibných metod je i hybridní přístup Learnable Evolution Model kombinující zavedené postupy z oblasti umělé inteligence a strojového učení s poslední dobou populárními a efektivními metodami evolučního programování. V této práci byla metoda zhodnocena z hlediska co už bylo implementováno a vyzkoušeno a bylo navrženo několik dalších možných implementací. Vybrané přístupy byly realizovány a otestovány na vybraných netriviálních spojitých funkcích. Výsledky byly následně porovnány s výsledky dosaženými pomocí EDA algoritmů.

Abstract

Numerical optimization of multimodal or otherwise nontrivial functions has stayed around the peak of the interest of many researchers for a long time. One of the promising methods that appeared is the hybrid approach of the Learnable Evolution Model that combines the well-established ways of artificial intelligence and machine learning with recently popular and efective methods of evolutionary programming. In this work, the method itself was reviewed with respect to what has been already implemented and tested and several possible new implementations of the method were proposed and some of them consequently implemented. The resulting program was then tested against a set of chosen nontrivial real-valued functions and its results were compared to those achieved with EDA algorithms.

Klíčová slova

evoluční algoritmy, evoluční výpočetní techniky, soft-computing, optimalizace, strojové učení, klasifikace, LEM, EDA, RCE, PRISM

Keywords

evolutionary algorithms, evolutionary computation, soft-computing, machine learning, classification, LEM, EDA, RCE, PRISM

Citace

Martin Weiss: Evoluční model s učením (LEM) pro optimalizační úlohy, diplomová práce, Brno, FIT VUT v Brně, 2011

Evoluční model s učením (LEM) pro optimalizační úlohy

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením doc. Ing. Josefa Schwarze Csc.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Martin Weiss
16. 5. 2011

Poděkování

Děkuji panu doc. Ing. Josefu Schwarzovi Csc. za pomoc při obstarávání materiálů k práci a osobní konzultace, i přesto, že můj přístup nebyl příkladný.

Obsah

1 Úvod

Odjakživa se lidé snažili provádět běžné činnosti co nejefektivněji, šetřit prostředky, pokud bylo možné a dosáhnout největšího užitku. Po většinu času bylo hledání těch nejlepších – optimálních – postupů a návrhů svěřeno analytikům, matematikům, ale s příchodem a rozvojem výpočetní techniky se tato role plně přenesla do rukou programátorů a jimi navrhovaných a programovaných počítačů. Byly definovány koncepty technicky spočítatelného a nespočítatelného, na pomezí algoritmizace a matematiky byl vyhraněn obor optimalizace a doteď, a dovolím si říct, že ještě dlouho, bude žít vlastním životem.

Úzce spojen s algoritmy a pokroky v oboru umělé inteligence a algoritmizace si obor prošel mnoha změnami. Přes počáteční úspěšné algoritmy lokálního prohledávání stavového prostoru jako Hill climbing, Simulované žíhání apod.[1][3][8] se postupně pozornost začala obracet k globálněji pojatějším metodám a dále k algoritmům často inspirovaným v přírodních evolučních procesech – evoluční a genetické programování, algoritmy na bázi částicových rojů, evoluční strategie, diferenciální evoluce atd.[1][2][8] Bohužel však přes stále sofistikovanější a nadějnější přístupy se ale nepodařilo nikdy nalézt univerzální optimalizační algoritmus, který by fungoval pro všechny možné problémy a s největší pravděpodobností se ani nikdy podařit nemusí [9]. Vznikla celá řada algoritmů na míru šitých určitým problémům, ale ani to nestačí poptávce po spolehlivých optimalizačních technikách. Čím dál více se setkáváme s problémy, jejichž podstata se těžko formalizuje či analyzuje, jsou v proměnné v čase, multimodální – je potřeba navrhovat techniky schopné efektivní optimalizace předem takto neznámeho problému - „black box optimalizace“.

Jednou z těch relativně mladších technik je metoda Learnable Evolution Model navržená původně R. Michalskim [4], které nebylo dle mého názoru věnována až taková pozornost, jakou by si zasloužila a i proto se stala námětem pro tuto práci. Algoritmus v sobě kombinuje postupy z evolučních programovacích technik s metodami strojového učení a ukázala se jako značně efektivnější než některé doposud zavedené metody. I přesto se neobjevilo moc odlišných implementací a modifikací a i toto by tedy mělo být předmětem této práce.

Nejprve se v textu seznámíme s numerickou optimalizací a souvisejícími předpoklady a důsledky pro optimalizační algoritmus. Dále se podíváme zblízka na stěžejní část této práce, tedy Learnable Evolution Model, jeho princip a aspekty, kterým bude věnována pozornost při další práci a také si krátce představíme EDA algoritmy, s nimiž má být na konci výše zmíněný algoritmus srovnáván. Pokračovat bude práce představením různých metod strojového učení a klasifikace, které se jeví vhodnými pro použití v LEM metodě. U každé bude diskutován její princip a možný přínos při jejím použití v numerické optimalizací pomocí LEM. Dále bude představená má vlastní implementace LEM metody – rozdíly oproti původní verzi/verzím a celkově detaily implementace. Poté budou postupně diskutovány naměřené výsledky na různých variantách numerických optimalizačních problémů jak pro jednotlivé varianty mé implementace, tak pro varianty implementace ve srovnání s výsledky dosaženými EDA algoritmy, což bude také všechno následně shrnuto v závěrečné kapitole.

2 Numerická optimalizace

2.1 Úvod do problematiky

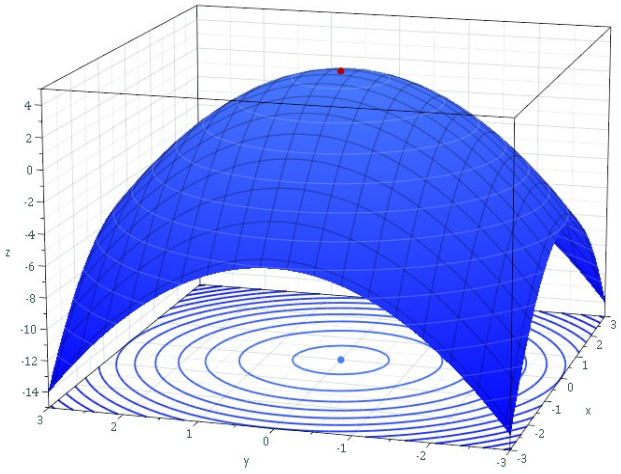
O numerické optimalizaci se v rámci zkoumání optimalizačních technik hovoří tehdy, máme-li před sebou nějakým způsobem definovanou funkci (zpravidla v praxi netriviální - tím je myšlena skutečnost, že neexistuje žádný analytický či jiný rychlý a spolehlivý způsob, jak analyzovat danou funkci – nalézt maxima, minima, derivaci, charakter atd.) definovanou nad obecně n-rozměrným vektorem hodnot a potřebujeme nalézt její globální extrém či extrémy. Jedná se zpravidla o spojité mnoharozměrné funkce blíže popsané v další sekci.

Formálně vyjádřeno, hledáme [1]:

$$x_{opt} = \arg\min_{x \in S} f(x) \tag{2.1}$$

Kde:

- X … reprezentace instance zkoumané funkce – jedno z možných řešení
- S … prohledávaný prostor možných řešení
- f … optimizovaná objektivní funkce (v textu většinou jako „fitness funkce“)



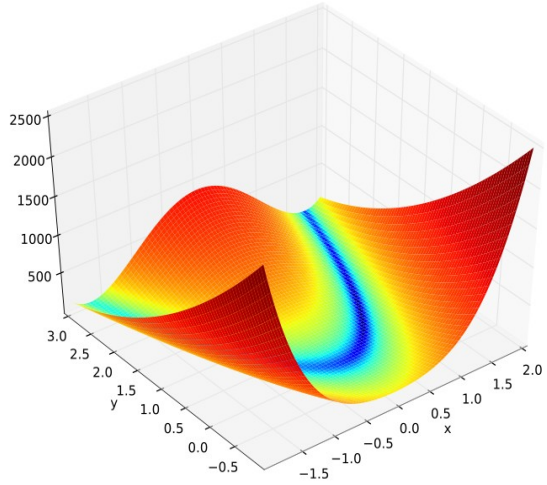
Obrázek 2.2: Ukázka paraboloidu s jasně viditelným extremém, v tomto případě maximem, při numerické optimalizaci, dostupný online na <http://upload.wikimedia.org/wikipedia/commons/6/62/MaximumParaboloid.png>

2.2 Zajímavé funkce

V předchozí kapitole ukázaná funkce je na první pohled ne až tak netriviální. Má přesně jeden extrém a dá se říct, že se ze všech bodů funkce dá jednoznačně určit směr vedoucí k extrému a žádný algoritmus by neměl mít velký problém s jeho nalezením. Pro testování a srovnávání optimalizačních algoritmů se používají funkce daleko složitější [2], definované nad n-rozměrnými vektory a ve výsledku často obsahující větší počet extrémů často zdánlivě náhodně rozesetých po stavovém prostoru. Ty, kterými se budeme dále v textu a především při finálním testování implementace algoritmu zabývat, si ukážeme.

2.2.1 Rosenbrockova funkce

$$f_2(x) = \sum_{i=1}^{n-1} 100 \cdot \left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2 \qquad -2.048 \le x_i \le 2.048 \tag{2.3}$$

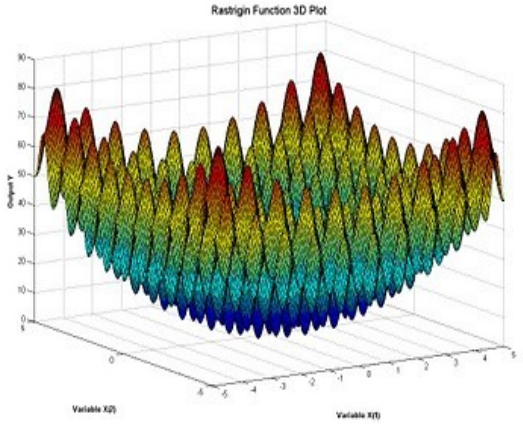


Obrázek 2.4: Ukázka tvaru a průběhu Rosenbrockovy funkce, Dostupný online na <http://upload.wikimedia.org/wikipedia/commons/thumb/3/32/Rosenbrock_function.svg/300px-Rosenbrock_function.svg.png>

Druhá de Jongova aneb také Rosenbrockovo sedlo, Rosenbrockova funkce či banánová funkce je často předmětem testování různých optimalizačních technik. Minimum nalézající se na souřadnicích $(x_1, x_2 \dots x_n) = (1, 1, \dots 1)$ o hodnotě 0 leží v údolí banánovitého tvaru, které samo o sobě je nalezeno vždy poměrně snadno, ale samotné minimum na dně plochého údolí není jedno z těch snáze nalezitelných a proto funkce slouží dobře pro testovací účely.

2.2.2 Rastriginova funkce

$$f_6(x) = 10 \cdot n + \sum_{i}^{n} \left(x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i)\right) \qquad -5.12 \le x_i \le 5.12 \tag{2.5}$$

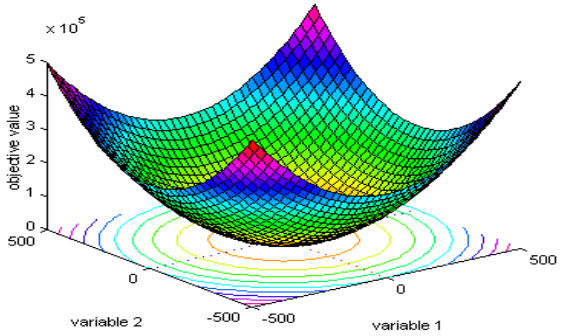


Obrázek 2.6: Ukázka tvaru a průběhu Rastriginovy funkce, Dostupný online na <http://clandestinetalk.blogspot.com/2010/09/optimization-benchmarks.html>

Díky přítomnosti kosinu se objevuje spousta lokálních minim a funkce se stane pro optimalizaci složitě multimodální, i když extrémy jsou alespoň pravidelně rozděleny ve prohledávaném prostoru. Minimum se nachází na pozici $(x_1, x_2 \ldots x_n) = (0, 0, \ldots 0)$ a má hodnotu -200*n (n je počet parametrů, čili dimenze prostoru)

2.2.3 První de Jongova funkce

$$f_1(x) = \sum_{i=1}^{n} x_i^2 \qquad -5.12 \le x_i \le 5.12 \tag{2.7}$$

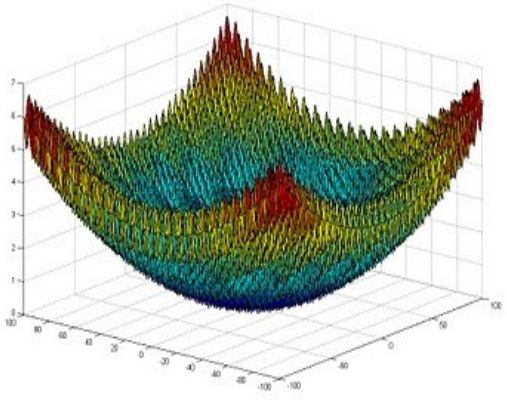


Obrázek 2.8: Ukázka tvaru a průběhu první de Jongovy funkce, Dostupný online na <http://www.karnig.co.uk/ga/4/image007.gif>

Unimodální jednoduchá první de Jongova funkce (v anglické literatuře často nazývaná „sphere function“) se také hojně používá pro testování výkonnosti optimalizačních technik. Globální minimum se nachází na pozici $(x_1, x_2 \ldots x_n) = (0.0, 0.0, \ldots 0.0)$ a má hodnotu 0.0. V této práci bude kvůli testovacím a porovnávacím účelům uvažována definice na intervalu <-600, 600>.

2.2.4 Griewangkova funkce

$$f_8(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \qquad -600 \le x_i \le 600 \tag{2.9}$$

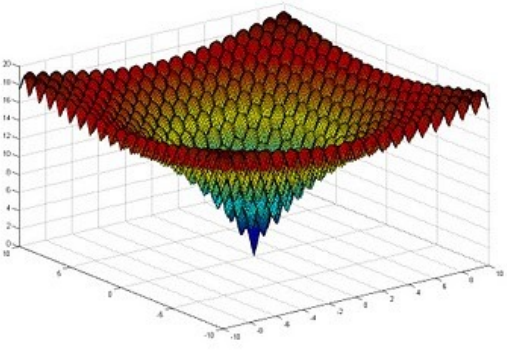


Obrázek 2.10: Ukázka náhledu na Griewangkovu funkci, Dostupný online na <http://clandestinetalk.blogspot.com/2010/09/optimization-benchmarks.html>

Funkce charakterem podobná Rastriginově, nicméně ve skutečnosti o mnoho více matoucí, záleží na rozlišení uvažovaném při náhledu na funkci. Při uvažování plných hranic pro parametry funkce (-600, 600) je funkce velice podobná obrácenému paraboloidu, nicméně čím blíže se na funkci chceme podívat, tím více vyplývá na povrch složitá struktura, jak je znázorněno na Obrázku 2.10. Globální minimum se nalézá na pozici $(x_1, x_2 \ldots x_n) = (0, 0, \ldots 0)$ a má hodnotu 0.

2.2.5 Ackleyho funkce

$$f_{10}(x) = -a \cdot e^{-b\sqrt{\frac{\sum x^2}{n}}} - e^{\frac{\sum \cos(c \cdot x_i)}{n}} + a + e^1 \qquad -1 \le x_i \le 1 \tag{2.11}$$



Obrázek 2.12: Ukázka náhledu na Ackleyho funkci, Dostupný online na <http://clandestinetalk.blogspot.com/2010/09/optimization-benchmarks.html>

Další hojně používaná multimodální testovací funkce s několika lokálními minimy v okolí globálního. Globální minimum se nalézá na pozici $(x_1, x_2 \dots x_n) = (0, 0, \dots 0)$ a má hodnotu 0.

2.3 Optimalizace funkce spojité veličiny

Při řešení obdobných problémů pomocí počítačů se člověk musí také rozhodnout pro způsoby pracování se spojitou veličinou – typ reprezentace hodnot v optimalizačním algoritmu a jeho vhodnost pro další operace.

2.3.1 Možné reprezentace spojité proměnné

Binární zakódování

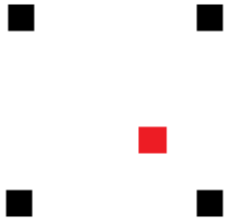
Vzhledem k tomu, že v rámci optimalizace metodou LEM bude místy nutno provádět nad kandidátními řešeními operace křížení či mutace typické pro evoluční přistupy k optimalizaci (více v kapitole 3), které jsou snadno a jasně definovány pro binárně zakódovaná řešení, je vhodné až nutné nějakým způsobem binarizovat spojitou hodnotu. Jednoduchý a postačující způsob popsaný třeba zde [1] se dá použít pro práci se spojitými proměnnými v procesu numerické optimalizace.

$$\alpha = (\alpha_1, \alpha_2 \dots \alpha_k) \in \{0,1\}^k \tag{2.13}$$

$$int(\alpha) = \sum_{i=1}^{k} \alpha_i 2^{k-1} = \alpha_1 2^{k-1} + \alpha_2 2^{k-1} + \dots + \alpha_{k-1} 2 + \alpha_k \tag{2.14}$$

$$x \approx real(\alpha) = a + (b-a)/(2^k - 1)\, int(\alpha) \tag{2.15}$$

Výše uvedené rovnice převzaté z knihy [1] víceméně jen definují převod binárního vektoru na celé číslo v desítkové soustavě. Každému takovému číslu ale můžeme přiřadit odpovídající aproximaci reálného čísla $x \in [a, b]$, přičemž přesnost takovéto aproximace je dána počtem bitů v původním binárním vektoru (v intervalu ohraničeném <a, b> je tedy možno rozlišit celkem 2^k různých hodnot). Volba délky bitového vektoru pro reprezentaci reálného čísla je velmi důležitá. Příliš velké rozlišení pravděpodobně zbytečně zpomalí výpočet (a to až neúnosně), příliš malé rozlišení naopak může dokonce znemožnit nalezení optima – aproximované hodnoty pokryjí spojitý prostor „mřížkou“ bodů a pokud je vzdálenost mezi body větší než dvou sousedících extrémů, může se snadno stát, že některý z ních nebude nikdy uvažován jako potencionální řešení – aproximace jej nevezme v potaz. Také nutno vyzdvihnout, že i když pokud se bod mřížky nalézá v globálním optimu, není zaručeno, že bude snadno nalezeno. Těmto případem se budeme snažit předejít použitím nějaké vhodne adaptivní diskretizační techniky viz další kapitoly.



Obrázek 2.16: Jednoduchá ukázka možného vynechání hledaného optima (červený bod) při moc „hrubé“ diskretizaci prohledávaného prostoru (černé body)

Reálné zakódování

U reálného zakódování není nutné řešit míru přesnosti pokrytí prohledávaného prostoru řešení, což přináší tu výhodu, že každá možná hodnota řešení je dosažitelná. Na druhou stranu se ale také zmenšuje šance, že bude nějaký bod v prostoru vůbec dosažen – obor všech možných hodnot je velký. Co se evolučních operátorů pro reálné proměnné týče, bez problémů jsou definovány také, ale jak již bylo zmíněno, použití reálných hodnot parametrů hledané funkce sebou přináší spíše výpočetní komplikace.

2.3.2 Reprezentace spojité veličiny pro strojové učení

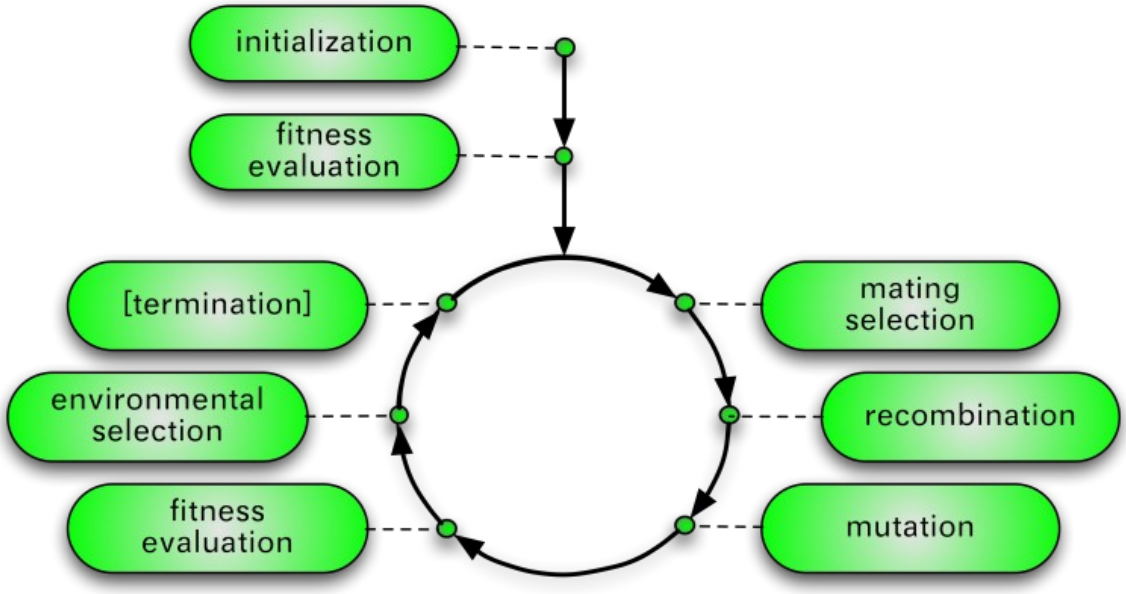
Řada algoritmů strojového učení pracujících na bázi induktivního učení, zobecňování pojmů, tvorbě rozhodovacích stromů (využití se ale najde i u Bayesovského klasifikátoru) pracuje s koncepty, pojmy či celkově jakýmikoliv obecně určujícími atributy zkoumaného objektu. To při použití metod strojového učení pro numerickou optimalizaci nebude překážkou. Pohybujeme-li se v oboru reálných čísel, je samozřejmě také možné, aby nad tímto oborem tyto algoritmy fungovaly jako pro symbolické atributy. Například jak je nastíněno v článku [7], pro získání nějakého souhrnného popisu pro skupinu numerických objektů náležejících stejné třídě můžeme použít tento postup - seřadíme si jednotlivé objekty (čísla) od největšího k nejmenšímu a tím získáme skupiny vedle sebe na číselné řadě ležících objektů. Vytvořené hranice (na půlce vzdálenosti mezi dvěmi skupinami, na půlce vzdálenosti mezi dvěmi prvky atd.) mezi těmito skupinami nám budou tvořit hranice intervalů, které budou používány jako atributy např. rozhodovacího stromu či kterékoliv jiné metody strojového učení, která tradičně pracuje nad symbolickými atributy.

3 Evoluční algoritmy

3.1 Úvod

Problematika evolučních algoritmů netvoří stěžejní část práce, nicméně jedna z nedílných součástí zkoumané LEM metody se skládá z několika kroků pro tradiční evoluční algoritmy typických, a proto si zde popíšeme alespoň základy problematiky a mechanismy EA použité v LEM.

Jak už bylo nastíněno v úvodních kapitolách, evoluční algoritmy jsou optimalizačními algoritmy, které se při hledání globálního extrému řídí zpravidla pouze operacemi inspirovanými v obecných evolučních teoriích – přirozený výběr, mutace jedinců atd. Řešení problému je tedy nějakým způsobem zakódováno/reprezentováno ve formě genotypu – řetězec hodnot reprezentujících nějakou vlastnost řešení a veškeré další operace jsou prováděny právě s takovými řetězci.



Obrázek 3.1: Obecné schéma chodu evolučního algoritmu, dostupný online na <http://shark-project.sourceforge.net/2.1.1/doc/images/evolution.png>

3.2 Princip fungování

Jak je vidět na obrázku 3.1, algoritmus pracuje na principu smyčky, kdy po vygenerování prvotní náhodné populace řešení až do splnění nějaké ukončující podmínky (nalezení řešení, dlouhodobé nezlepšení, daný počet běhů atd.) pořád dokola provádí kroky:

- výběr jedinců – ze současné populace řešení vybere nejvhodnější jedince (hodnota fitness funkce – kvalita řešení) pro další operace
- rekombinace – křížení jedinců mezi sebou a vznik jejich „potomků“
- mutace – případná mutace hodnot takto vzniklých jedinců
- ohodnocení nově vzniklých jedinců

- výběr jedinců pro populaci dalšího kroku (využití předchozí populace/potomků se může lišit pro různé varianty algoritmu)

Pro použití v LEM jsou pro nás důležité vlastně jen kroky selekce, mutace, křížení, a proto si je v následující kapitole stručně představíme – nebudou nás zajímat různé varianty, jen popíšeme funkci těch typů operací, které nás zajímají z hlediska použití v původní verzi algoritmu anebo v mé implementaci.

3.2.1 Selekce

Dá se najít více způsobů, jak vybírat jedince z populace pro další zpracování. Tím základním je výběr jedinců pouze na základě jejich hodnot užitkové (fitness) funkce – jedinci jsou seřazeni a vybrán je určitý počet nejlepších (rank based selection). Tento postup se využije zpravidla v kroku výběru Dále je ještě vhodné v algoritmu použít dva typy selekce a to turnajovou selekci a selekci ruletou (nebo též zvaná proporcionální) např. pro výběr jedinců pro operace křížení. V praxi se použije většinou kombinace obou.

Turnajová selekce

Tento typ operace selekce v každém kroku výběru vybere zpravidla dva náhodné jedince (existují implementace pro obecně N jedinců) a porovná jejich ohodnocení fitness funkcí a zvolí pro další kroky toho, který má ohodnocení lepší/nejlepší. Dále by se dalo ještě rozlišovat, zda tito jedinci můžou být znovu vybrání při dalším výběru nebo jsou vyřazení z možného okruhu řešení pro další krok výběru, ale dále tato problematika diskutována nebude.

Selekce ruletou

Při tomto postupu je také brána v úvahu kvalita fitness funkce pro výběr jedince např. pro fázi rekombinace, ale je zde přidán prvek náhody. Znovu je tedy vybráno 2 .. N jedinců, pro které jsou vypočítány „díly rulety“ - pravděpodobnosti výběru úměrné kvalitě jednotlivých řešení – a následně je náhodně vybrán jeden z nich pro další fázi.

3.2.2 Rekombinace

U možných postupů křížení a mutace si je rozdělíme podle oboru hodnot jedinců – tedy podle aplikovatelnosti na řešení zakódovaná jak binárními vektory, tak reálnými čísly.

Binární zakódování

Uniformní křížení

Při tomto křížení jsou postupně procházeny jednotlivé geny/atributy/bity jednotlivých rodičů a výsledkem jsou dva potomci, kteří mají jednotlivé hodnoty genů nastaveny na hodnoty jednoho z rodičů s pravděpodobností 50%.

Jedno-/vícebodové křížení

U tohoto postupu jsou také výsledkem dva potomci. Hodnoty jejich jednotlivých genů jsou také tvořeny hodnotami buďto z jednoho nebo druhého rodiče, ale je toho dosaženo tak, že se zvolí jeden či více bodů (pozic v genotypu) a každý z jedinců přebírá hodnoty vždy z jednoho z rodičů, které si vždy dočasně vymění právě na zvolených pozicích.

Binární mutace

Triviální operace – záměna jednoho či více náhodných bitů/genů v genotypu jedince.

Reálné zakódování

Aritmetické křížení

Pří tomto křížení reálných hodnot se pro hodnoty obou potomků použijí hodnoty obou rodičů s náhodně zvolenou hodnotou a pro násobení jednotlivých genů viz obrázek níže.

$$g_i^{O_1} = g_i^{P_1} a_i + g_i^{P_2}(1 - a_i)$$
$$g_i^{O_2} = g_i^{P_1}(1 - a_i) + g_i^{P_2} a_i$$

$$g_i \ldots hodnota\ genu\ i$$
$$P_j \ldots rodič\ j = 1,2$$
$$O_k \ldots potomek,\ k = 1,2$$

Obrázek 3.2: Postup při aritmetickém křížení reálných proměnných

Mutace

Znovu velmi jednoduchá operace, kdy buďto k hodnotě genu přičte nebo odečte náhodně zvolená hodnota, který zpravidla nepřesahuje 10% původní hodnoty.

4 Evoluční model s učením (LEM)

4.1 Úvod

Jak už bylo zmíněno, optimalizační algoritmy na bázi v přírodě inspirovaných mechanismů, především principu přirozené evoluce jedinců, se staly obzvláště populárními – často dokáží najít dostatečně dobré řešení dostatečně rychle i pro problémy, o kterých nebyla doposud známá žádná doménová znalost napomáhající jejich analýze a řešení. V roce 1998 přišel Ryszard Michalski s návrhem algoritmu využívajícího jak metodologii evolučních výpočtů, tak principy a postupy klasické umělé inteligence, kterými se do té doby sám zabýval [4]. Nová metoda se ukázala být velice slibná, značně rychlejší než běžné algoritmy téže třídy a tak o dva roky později přišel s inovovanou verzí algoritmu [5] a o šest let později spolu s Januszem Wojtusziakem s zatím nejvyvinutější implementací původního konceptu [10], jejíhož konceptu se budu držet i ve své práci.

Od samých počátků se metoda opírá o učicí jádro založené na AQ algoritmu, který původně navrhl R. Michalski v roce 1969 [11]. Tato technika spadající pod algoritmy induktivního učení se do dneška vyvíjí a zatím poslední verze [12] se stala součásti zmíněné poslední verze - LEM3. Od své druhé verze LEM také využívá již dříve zmíněné speciálně navržené adaptivní vzorkování spojitého prostoru možných řešení ANCHOR [6]. Celkově se metodologie LEM nezaměřuje jen na numerickou optimalizaci, ale na jakýkoliv optimalizační problém i s kategorickými či symbolickými atributy, ale její použití pro numerickou optimalizaci bylo od začátku navrhováno a uskutečňováno samotnými autory a proto se i dá předpokládat, že zde nebude metoda nějak výrazně ztrácet na zajímavosti či efektivitě.

4.2 Současný stav

Metoda neinspirovala vědeckou veřejnost takovou měrou, že by došlo k hojné reimplementaci a pravidelnému publikování různých verzí LEM využívajících různé metody pro strojové učení či jiných vylepšení, i když několik pokusů se objevilo. Na pohled nejtriviálnější byl asi pokus o využití metody klasifikace dle nejbližších sousedů jako LEM-KNN [14] a o dva roky později práce stejných autorů a použití klasického algoritmu ID3 v LEM-ID3 [13] a srovnání s nejlépe pracujícími současnými optimalizačními algoritmy. Výsledky byly v případě LEM-KNN uspokojivé vzhledem k dosavadním výsledkům tradičních evolučních algoritmů a v případě LEM-ID3 bylo dokonce dosaženo několika lepších výsledků v porovnání se současnými optimalizačními technikami na sadě složitých funkcí. Metoda nejen, že dokázala držet krok, ale navíc čím více rostla dimenzionalita problému, tím lépe si vedla. Třetí rozdílná implementace konceptu LEM byl učící algoritmus C4.5 a aplikace na optimalizaci vodní rozvodné sítě [15], která je už specifičtějšího rázu a netýká se oblasti numerické optimalizace.

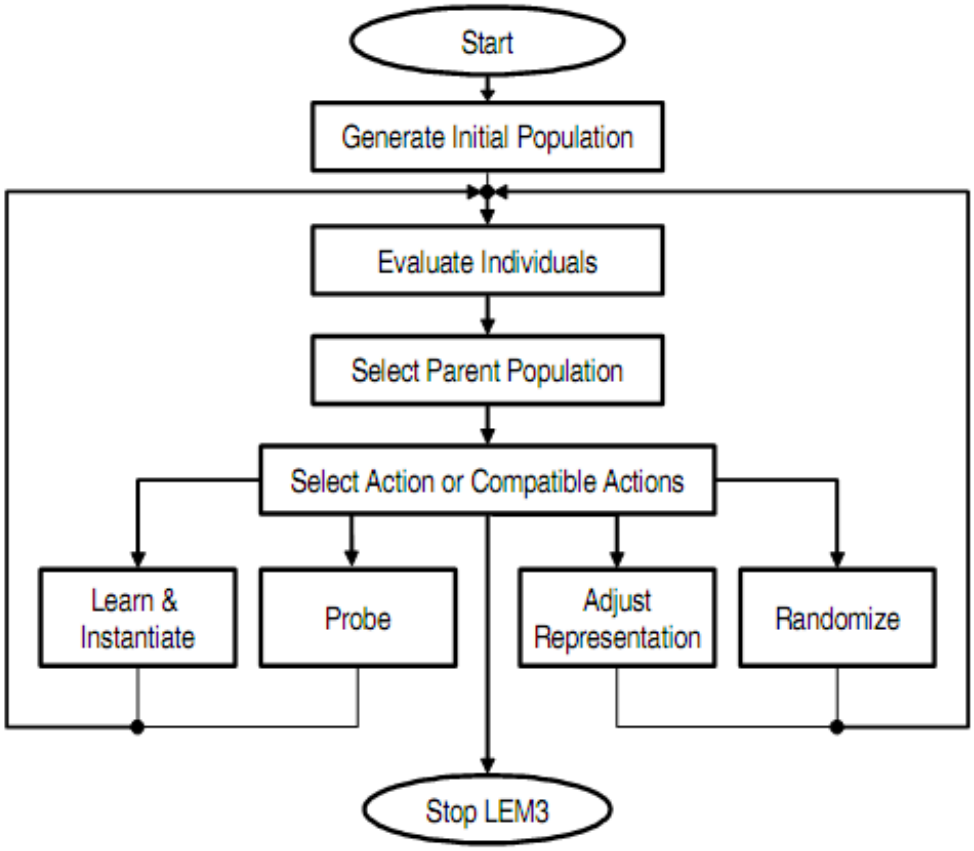
4.3 Přehled a rozbor metody

Zatímco první verze LEM nám dává jen hrubý nástin principu a první optimistické výsledky v porovnání s tradičními EA, s publikací druhé generace metody přišel podrobný rozbor jednotlivých fází algoritmu a v třetí, nejnovější verzi, dojde ještě k dalšímu prohloubení a doladění jednotlivých částí algoritmu. Zajímavé je výkonnostní srovnání jednotlivých generací metody v [10]. LEM2 i přes své starší datum publikace pořád dokáže držet krok a i často předčit svého nástupcen a různých problémech. Pořád se ale nehodí pro problémy o 200 a více dimenzích prostoru, proto se dá LEM3 považovat za současný standard.

Metoda by se dala shrnout do těchto kroků:

1. Vygeneruj startovní populaci
2. *Začni fázi strojového učení*
    - *Najdi extrémy* – z populace separuj dvě skupiny, H-skupinu a L-skupinu, první obsahující dobře ohodnocené jedince, druhá špatně ohodnocené jedince
    - *Vytvoř hypotézu* – za použití metod strojového učení vytvoř si hypotézu, která rozlišuje jedince z H a L skupin
    - *Vytvoř novou populaci* – dle popisu dobře ohodnocených jedinců z předchozích kroků vytvoř jedince nové. Tento krok závisí na zvolené metodě strojového učení (detaily budou blíže rozebírány v další kapitole), případně může být místy tento krok přenechán na klasických genetických operátorech.
    - Pokud není splněna ukončující podmínka pro učení (dokud není vylepšení objektivní funkce nejlépe dosaženého řešení nedostačující), vrať se na začátek fáze strojového učení, jinak:
        - Jestli je splněna podmínka optimalizace – nalezeno optimální řešení, ukonči úspěšně algoritmus
        - Vrať se na krok 1 (*start-over*) nebo pokračuj ke kroku 3
3. *Začni fázi darwinovské evoluce* – vytvářej novou populaci tradičním způsobem operátory mutace, křížení a selekce dokud není splněna podmínka pro ukončení evoluce
4. Nedaří-li se dostatečně vylepšovat řešení:
    - Zpřesni diskretizaci prostoru řešení nebo
    - Přidej náhodné jedince do populace, vygeneruj novou populaci
5. *Střídej* kroky 2, 3 a 4 dokud není splněna podmínka ukončení optimalizace, je nalezeno řešení

Obrázek 4.1: Pseudokód LEM metody



Obrázek 4.2: Vývojový diagram LEM metody [10]

V článku [10] byly navrženy a popsány stěžejní fáze LEM metody a jejich funkce. Podívejme se na ně tedy detailněji.

4.3.1 Probe

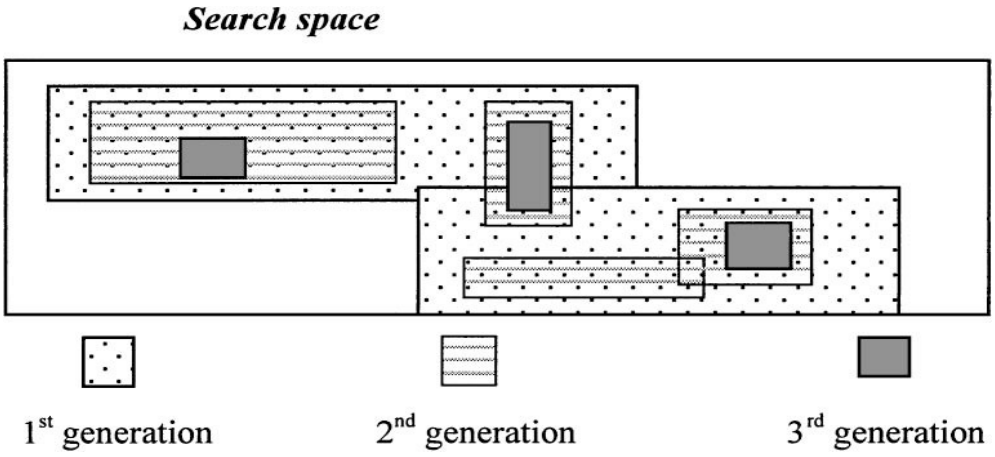
Část nazvaná *Probe* představuje evoluční procesy křížení a mutace aplikované na současná řešení pro získání nové populace. Používá se procesů popsaných v kapitole 3.2 a i když by se mohlo zdát, že není až tak potřebným krokem pro správný chod algoritmu, tak naopak díky použití pořád víceméně náhodné tvorby nových jedinců pomáhá algoritmu vybřednout z často mnoha lokálních optim. Zde není moc prostoru pro vlastní invence.

4.3.2 Learn/Instantiate

Krok *Learn/Instantiate* je fází výběru relevantních řešení, strojového učení a generování nových řešení na základě naučené hypotézy ze zvolených vzorků. Jednotlivé akce se budou lišit dle výsledné formy hypotézy. V kapitole 6 budou diskutována možné alternativy pro učící jádra a související metody generování jedinců. Zde si popíšeme, jak tyto operace probíhají v původním algoritmu.

Nejprve se podíváme na způsob výběru jedinců pro tvorbu hypotézy. Navržený způsob (a není dle mého názoru rozumný důvod jej měnit) je pokaždé vyhradit z celé populace dvě „pozoruhodné“ skupiny řešení (H- a L-skupina), konkrétně skupinu nejlepších jedinců (H) a nejhorších (L). Způsobů výběru je zde více – můžeme považovat všechny jedince, kteří přesahují (nebo naopak nedosahují) určité procento kvality fitness funkce v populaci (20-30%) za reprezentanty dané skupiny, můžeme pokaždé určit skupiny jako pevný počet nejlepších a nejhorších jedinců anebo se nabízí možnost rozdělit si jedinec do více „výkonnostních“ skupin, čímž si napomůžeme k udržování diverzity nejen za pomoci ostatních kroků algoritmu, ale zase si zhoršíme vypovídající hodnotu tvořené hypotézy. Po vytvoření námi zvolených skupin se přejde k fázi *Learn*.

Jako zvolený algoritmus strojového učení je zde použita metoda AQ (blíže popsaná v kapitole 6), která pracuje s intervaly hodnot získaných způsobem popsaným v sekci 2.3.2. Jejím výsledkem je tedy, když vizualizováno, postupné zmenšování intervalů, ve kterých by se mohla nalézat nadějná řešení jak je ukázáno na obrázku 4.3.



Obrázek 4.3: Ukázka postupu segmentace prostoru AQ algoritmem [5]

Výsledkem učení máme tedy vždy nějak popsané intervaly slibných řešení (při více dimenzích jsou ne vždy všechny dimenze pokryty) a úkolem fáze *Instantiate* je z těchto pravidel/intervalů získat nová potencionálně kvalitní řešení. Pokud je pro námi žádanou dimenzi (gen) interval/pravidlo definován, můžeme zvolit náhodný prvek z tohoto intervalu, případně použít nějakého uniformního rozložení na daném intervalu. Pokud pro gen interval/pravidlo k dispozici nemáme můžeme samozřejmě zvolit libovolnou hodnotu s problémové domény, ale tím bychou se často zbytečně zbavovali nadějného řešení – je tedy lepší vzít hodnotu tohoto genu náhodně z existujícího jedince z H-skupiny či z existujícího jedince v celé populaci (ale nesmí být členem L-skupiny).

4.3.3 Adjust representation

Velmi důležitou součástí algoritmu je tato fáze, která odstraňuje nevýhody statické diskretizace popsané v kapitole 2.3.1. Pro tento krok byla navržena metoda navržena tvůrcem algoritmu LEM R. Michalskim ve spolupráci s jeho studentem G. Cervonem [6] a jako taková byla používána

od samých počátků metody LEM ve všech nových implementacích. Bez použití adaptivní binarizační techniky se dá docela spolehlivě říct, že by bylo v mnoha případech těžší dopátrat se v rozumném čase hledaného optima. Metoda dokáže snadno zpřesnit jakékoliv hodnoty, které se ukážou během optimalizace za hodné přesnějšího prozkoumání (získané intervaly H-skupiny). Dosahuje toho prostým vytvořením si dvou polí v paměti, která nám mapují získané hodnoty z genotypů na jiné podle potřeby. Samozřejmě se tím významně snižuje diverzita populace, nicméně v případě, že zpřesňování nevede k lepším výsledkům, je diverzita obnovena jinými kroky algoritmu.

4.3.4 Randomize

V případě, že předchozí operace LEM nevedou ke zlepšení po určitý počet generací, je v algoritmu zakomponována funkce *Randomize*, která buďto nahradí celou populaci novou (začíná se od nuly) nebo jen přidá nové jedince do populace. Zde jsou jedinci generováni buďto náhodně nebo se využívá informací získaných za běhu – jedinci se generují v regionech, kde ještě generování moc nebyli anebo se generují jedinci co nejdále od doposud nejkvalitnějších jedinců (H-skupina).

4.3.5 Action-profiling-function a Select-action

Doposud nezmíněné autory vyzdvihované součásti LEM3 [10]. Jedná se o funkci (APF), která v každé iteraci rozhoduje kolik nových jedinců bude vygenerováno pomocí modulu strojového učení a kolik bude stvořeno pomocí operace Probe – tedy evolučními operátory křížení a mutace.

Ne vždy je po vygenerování nové populace dosaženo lepšího řešení a tehdy přicházejí ke slovu nastavitelné parametry metody (více v návodu na použití v přílohách). Zkráceně se jedná o čítače, které řídí, kolikrát se může generovat nová populace za pomoci APF i když nedojde k žádnému pokroku a následně se provádějí operace mutace, křížení, zpřesnění diskretizace nebo kompletní restart algoritmu dle stavu čítačů. Funkci Select-action si ukážeme podrobněji, protože je vlastně tím druhým řídícím prvkem celého algoritmu.

```
Increment learn-probe-counter
      If learn-probe-counter >= learn-probe
            Learn-probe-counter = 0
            If mutation-probe-counter < mutation-probe
                  Increment mutation-probe-counter
                  Mutate individuals (Probe)
            Else if representation-probe-counter < representation-probe
                  Increase representation-probe-counter
                  Mutation-probe-counter = 0
                  Adjust discretization
                  Mutate individuals
            Else if randomize-counter < randomize-Probe
            Increment randomize-probe-counter
                  Representation-probe-counter = 0
                  Mutation-probe-counter = 0
                  Rollback discretization
                  Add the best-individuals to a list of local optima
                  Randomize
            Else
                  Stop LEM3
```

Obrázek 4.4: *Pseudokód operace Select-action* [10]

Na obrázku 4.4 můžeme tedy vidět jak jednotlivé čítače ovlivňují chod algoritmu. V praxi se ke krokům zpřesnění diskretizace a randomizace přistupuje jen v krajních případech (jak bylo zmíněno výše – při dlouhodobé stagnaci optimalizace) a proto jejich četnosti (max. hodnoty čítačů) bývají menší než u evolučních operátorů či tvorby jedinců z naučené hypotézy.

4.4 Diskuse

Jednotlivé fáze algoritmu nám dávají relativní volnost v jejich implementaci. Nejvíce lákavé možností úpravy se budou samozřejmě výběrem algoritmu strojového učení, který použijeme pro stejnojmennou fázi (o tom ale více v kapitole 6), dalším místem, které nám dává volnost pro experimenty je fáze výběru *H* a *L-skupin* ve fázi Learn/Instantiate. Nabízí se možnost určovat, kteří a kolik jedinců bude tvořit důležité skupiny, průběžně podle průběhu optimalizace, kdy se můžou jednotlivé proporce kvalitních a nekvalitních jedinců často měnit. Může docházet k situacím, kdy je kvalitních jedinců značně více, než je předem stanovená kapacita skupiny nebo naopak podstatně méně a do skupiny pro vytvoření hypotézy by se mohlo dostat větší množství špatně ohodnocených jedinců, což by znehodnotilo do té doby pracně budovanou hypotézu. Co se týče fází zpřesňování reprezentace, přidání náhodných jedinců a generování jedinců za pomocí evolučních operátorů, tak ty jsou poměrně jasně definovány a není třeba nic moc měnit.

Důraz by měl být také kladen na to, aby si algoritmus alespoň do jisté míry pamatoval předchozí výsledky a zahrnoval je do tvorby aktuální hypotézy – jen tak může být získávaná hypotéza čím dál kvalitnější. Zase ale musí člověk dávat pozor, aby budovaný model nebyl aJak už bylo zmíněno, také se může ž moc komplexní, že z něj sice bude možné destat pravidla o H skupině, ale budou neužitečná. Také by se mohlo ukázat jako vhodné nerozlišovat jen dvě skupiny, protože všechna řešení, která se do těchto dvou skupin nedostanou, jsou zahozena, i když by v nich teoreticky také mohly být užitečné informace. Všechny klasifikační metody nastíněné v další kapitole jsou schopny pracovat s více třídami, čili tento postup nebude problém realizovat a vyzkoušet případný přínos. Jako výhodu bych viděl, že při braní v potaz i horších řešení bude udržena relativní diverzita populace řešení, na druhou stranu by takovýto krok pravděpodobně snížil kvalitu naučených klasifikátorů. Nicméně pravděpodobné je, že udržování diverzity řešení v populaci tímto způsobem nebude nijak zvlášť stěžejní – metod pro zpestření genofondu řešení je už tak dost – samotná metoda strojového učení vždy generuje nové jedince s určitým rozptylem a k tomu jsou k dispozici evoluční operátory mutace a křížení.

5 EDA algoritmy

Vzhledem k stále jisté „slepotě“ tradičních evolučních výpočetních technik, jejich tendenci nenalézat i teoreticky snadno dosažitelná optima se na relativní výsluní dostala vlna „Estimation of Distribution“ algoritmů. Koncept EDA by se dal popsat jako kombinace více než dvou rodičů při tvorbě nového kandidátního řešení a to formou postupné tvorby pravděpodobnostního modelu – hypotézy – popisujícího kvalitní řešení, který je následně používám pro generování potomků. Bližší popis, různé varianty a detaily tohoto optimalizačního přístupu jsou dostupné v mnohé literatuře, namátkou třeba [17] [18] [19]. Jen pro úplnost zde předložím pseudokód obecného EDA algoritmu:

1. *Vygeneruj R náhodných jedinců jako počáteční populaci*
2. *Dokud nedojde ke splnění ukončovací podmínky:*
    - Vyber N < R jedinců z populace podle určené metody selekce (výše objektivní funkce)
    - Odhadni pravděpodobnostní rozdělení jevu, že náhodně vybraný jedince bude patřit mezi jedince vybrané v předchozím kroku.
    - Z pravděpodobnostního modelu získaného v předchozím kroku algoritmu vygeneruj novou populaci

Obrázek 5.1: Pseudokód EDA přístupu

Je snadno vidět, že celková motivace na pozadí EDA algoritmů je velmi podobná nosné myšlence metody LEM, a i proto je jedním z cílů pokračování této práce srovnání výsledků různých implementací LEM algoritmu s výsledky dosaženými EDA algoritmy. Nutno doplnit, že k podobnému srovnání už došlo přímo autory algoritmu LEM, ale uvažována byla pouze původní autorská LEM3 metoda, která dosáhla výrazně lepších výsledků na několika funkcích [10].

Pro srovnání bude použito informací a dat z [24], kde máme přehledně otestované různé EDA varianty pro řešení optimalizačních problémů v reálné doméně. Varianty EDA algoritmu jak jsou zmíněné a popsané v článku [24] jsou:

- $UMDA_c$ – pravděpodobnostní profil řešení je počítán za pomoci normálního pravděpodobnostního rozložení
- $MIMIC_c$ – pro pravděpodobnostní rozložení řešení je využíván řetězový model a statistické metody prvního a druhého řádu
- $EGNA_{BIC}$ – využití Gaussovské sítě a lokálního prohledávání pro získání nejlepšího rozložení
- $EGNA_{BGe}$ – podobně jako předchozí jen s využitím Bayesovské metriky
- $EGNA_{ee}$ – jako předchozí s variací na Gaussovské sítě
- $EMNA_{global}$ – využití vícedimenzionálního normálního rozložení
- $EMNA_a$ – kombinace vícedimenzionálního normálního rozložení a evolučního algoritmu

6 Vybrané metody klasifikace pro použití v LEM

Postupně bylo zhodnoceno několik různorodých odvětví metod strojového učení, vhodných jak pro snadnou tvorbu hypotéz z daných množin řešení, tak pro generování nových řešení na základě získaného modelu. U každé metody bude popsán jak její základní princip, tak způsob učení hypotézy rozlišující kvalitní a nekvalitní řešení a také možnou metodu generování nových řešení založených na získaných pravidlech či hypotéze. Snažil jsem se vždy najít vhodnou alternativu k už jednou implementované metodě strojového učení spadající do stejné kategorie algoritmů/přístupů a také aby nebyl problém metody použít pro obecně n rozměrný prostor řešení.

Někdy budou metody, aby fungovaly účinně, vyžadovat veliký počet vzorků a vzhledem k tomu, že velikost populace nebude vždy muset být dostačující, v úvahu může připadat brát jako vstupní vzorky jedince vygenerované za několik generací ať už evolučními operátory či z dosažené současné hypotézy nebo pracovat s velkou populací. Toto bude muset být experimentálně vyzkoušeno, který přístup se ukáže jako nejvhodnější.

6.1 Shlukovací metody

Bez předchozí znalosti původního výběru metody strojového učení pro LEM by asi jako první člověk mohl volit některou z jednoduše implementovatelných technik využívajících myšlenky, že řešení sobě blízká v prohledávaném prostoru si budou blízká i svou kvalitou. Navíc pro numerickou optimalizaci není ani zapotřebí jinak nutné normalizace symbolických atributů, pro vzdálenosti se dá použít jednoduchá euklidovská metrika a metody by mohly být účinné stejnou měrou pro všechny možné testovací funkce – oblasti extrémů budou s vysokou pravděpodobností obsahovat blízké hodnoty extrému i ve svém okolí.

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

Obrázek 6.1: Vztah pro výpočet vzdálenosti v n-rozměrném prostoru

6.1.1 k-nearest neighbour algoritmus

K-nearest neighbour model je jedním z klasifikátorů, které už byly použity v kombinaci s LEM metodou [14], proto jen zběžně nastíníme princip a přínos pro algoritmus. Základem je určit si konstantu r, definující radius prohledávání okolí pro nejbližší sousedy. Poté se dají uvažovat dvě možnosti postupu klasifikace bodu (řešení):

- v rozsahu parametru r hledám nejbližšího souseda a klasifikovanému bodu přiřadím třídu nejbližšího souseda

- v rozsahu parametru r spočítám počty sousedů náležejících do rozdílných tříd a klasifikovaný bod přiřadím do třídy jež měla nejvíce svých zástupců v uvažovaném regionu

Aplikováno při LEM optimalizaci bych si postup představoval tak, že po rozlišení obou význačných skupin řešení (viz. Popis L a H skupiny v kapitole 3) budu postupně generovat náhodná řešení a pokoušet se je klasifikovat pomocí výše zmíněných pravidel. Zjistím-li, že bylo nové řešení přiřazeno k sousedům z L skupiny, řešení zahodím a pokračuji dále dokud nemám dostatek nových potencionálně kvalitních řešení. V rámci zkvalitňování výsledné hypotézy by nemuselo být na škodu nezahazovat žádný z vygenerovaných bodů spolu s jeho příslušností k třídě řešení – to by mělo zajistit čím dál přesnější generování nových dobrých řešení.

6.1.2 Kombinace k-means a hierarchického shlukování

Vzhledem k tomu, že předchozí klasifikační metoda byla už použita v kombinaci s LEM, v potaz jsem vzal principiálně nepříliš vzdálené metody k-means a hierarchické shlukování založené s menšími úpravami na postupu zhora dolů uvedeném zde [21]. Po identifikaci L a H skupin řešení začnu tvořit shluky bodů v prostoru/řešení dělením L a H skupin vždy na dva shluky postupně po jednom nejvzdálenějším bodu k ostatním bodům ve shluku. V okamžiku kdy dosáhne počet shluků určitého námi definovaného počtu, dělení zastavíme a určíme vždy jeden z bodů jako střed svého shluku a budeme pokračovat metodou k-means, která nám bude středy shluků hýbat a postupně se ustálí. Využijeme shluky bodů patřících do H skupiny, - vybereme z nich ten s nejlepším průměrným ohodnocením bodů - pro generování nových kandidátních řešení například tak, spočítáme střed a standartní odchylku a postupně náhodně generujeme řešení využívající těchto statistických parametrů.

Nabízí se také možnost brát v potaz více nejlépe ohodnocených shluků nebo použít jen jednu z výše zmíněných shlukovacích metod – bude-li metoda vybrána, nejlepší postup bude vhodné ověřit experimentálně.

6.2 Induktivní učení

Metody založené na principu induktivního učení byly, jak zmíněno v předchozích kapitolách, skoro výhradně používány jako hlavní metody strojového učení v různých implementacích LEM. Jejich původní zaměření na symbolické atributy není překážkou ani pro numerickou optimalizaci, jak bylo ukázáno v kapitole 2.3 a budou tedy využívat různé subintervaly prostoru řešení jako pravidla. I díky protěžování algoritmu AQ v původní implementaci LEM se tyto přístupy svým charakterem jeví jako vhodné pro použití v LEM, jak bude ukázáno dále.

Výstupem těchto metod strojového učení bude zpravidla vždy množina pravidel (nebo model umožňující nám takováto pravidla snadno odvodit), popisujících koncept kvalitního řešení. Pravidla budou ve tvaru *interval , interval, ... => H skupina* nebo *interval , interval, ... => L skupina* a tedy generace nových jedinců za pomoci takovýchto hypotéz není vůbec složitá. Máme-li n-dimenzionální reprezentaci řešení/jedince, nového jedince vygeneruji z pravidla/pravidel takto:

- pro každou dimenzi, která je v pravidle zastoupena, vygeneruji náhodnou hodnotu z příslušného intervalu na levé straně pravidla

- pokud není dimenze zastoupena ani v jednom pravidle, mám dvě možnosti:
    1. chybějící hodnotu zvolím náhodně
    2. vyberu náhodného jedince z předchozí/stávající generace a hodnotu zkopíruji z něj [10]

6.2.1 Algoritmy ID3, C4.5

Oba algoritmy jsou vlastně variantami strojového učení formou tvorby rozhodovacího stromu, tradičního algoritmu v oboru definovaného např. v [3] takto:

```
function Decision-Tree-Learning(examples, attributes, default) returns a decision tree
if examples is empty then return default
else if all examples have the same classication then return the classication
else if attributes is empty then return Mode(examples)
else
        best << Choose-Attribute(attributes, examples)
        tree << a new decision tree with root test best
        m << Majority-Value(examples)
        for each value vi of best do
                examplesi  << {elements of examples with best = vi
                subtree << Decision-Tree-Learning(examplesi, attributes - best, m)
                add a branch to tree with label vi and subtree subtree
        return tree
```

Obrázek 6.2: *Pseudokód algoritmu Decision-Tree-Learning [3]*

Rozhodovací strom se jeví velice vhodnou klasifikační metodou pro použití v LEM. Po sestavení stromu na základě příkladů z L a H skupin můžeme rychle odvodit pravidla vymezující slibné regiony prohledávaného prostoru (cesty z listů označených H skupina do kořene) a tím pádem rychle vygenerovat dostatek možných nových kvalitních řešení, která leží uvnitř těchto intervalů, chybějící hodnoty zvolit náhodně nebo z náhodně vybraného minulého řešení a pokračovat dokud nemáme dostatek jedinců.

Obě výše zmíněné varianty metody už kdysi byly úspěšně použity v implementaci spolu s LEM [13][15], obě využívají principů teorie informace – entropie - a vlastně se liší pouze v implementaci metody *Choose-Attribute.*

ID3

$$Info(S) = -\sum_{i=1}^{m} p_i \log_2(p_i) \tag{6.3}$$

$$Info(S_j) = -\sum_{i=1}^{m} p_{ij} \log_2(p_{ij}) \tag{6.4}$$

$$Info_A(S) = \sum_{j=1}^{v} \frac{|S_j|}{|S|} Info(S_j) \tag{6.5}$$

$$Gain(A) = Info(S) - Info_A(S) \tag{6.6}$$

Zde je jako nový kořen sestavovaného stromu vybírán atribut s největší hodnotou Gain(A), což v praxi znamená, že je vybíraný atribut, který nám do nějaké cílové třídy klasifikuje statisticky

nejvýznamnější (využittí teorie informace/entropie – bližší výklad není relevantní pro tuto práci) část uvažovaných vzorků - řešení. Finální rozhodovací strom pokryje všechny možné příklady.

C4.5

$$SplitInfo(A) = -\sum_{j=1}^{v} \frac{|S_j|}{|S|} \log_2 (\frac{|S_j|}{|S|}) \tag{6.7}$$

Vylepšení původního algoritmu ID3. Hodnota Gain(A) je normalizována tak, že je spočítána hodnota SplitInfo(A) a původní Gain(A) je touto hodnotou vydělen. Tím je zamezeno zvýhodňování atributů, které jsou unikátní a znevýhodňování atributů, které nabývají velkého počtu hodnot. Nevítězí tedy atribut s nejvyšší hodnotou Gain(A), ale ten s nejlepším poměrem k hodnotě SplitInfo(A). Jinak je metoda naprosto stejná, jako předchozí.

6.2.2 Algoritmus AQ

```
HYPOTHESIS = null
    while not all H-group examples are covered
       Select uncovered positive example e+ and use it as a seed
       Generate star G(e+, L-group)
       Select the best rule, R, from the star according  to a given
        criterion of optimality, and add it to HYPOTHESIS
       Remove examples covered by R from H-group
```

Obrázek 6.8: Pseudokód algoritmu AQ [10]

Algoritmus AQ, jak už bylo zmíněno, se stal stěžejním stavebním prvkem implementace LEM původních autorů. Nebuduje rozhodovací stromy, ale rovnou generuje pravidla – hypotézu – platná pro všechny vzorky. Jak je popsáno v pseudokódu na obrázku 5.8, algoritmus postupně prochází všechny příklady z H skupiny a postupně odstraňuje ty, které jsou pokryty vzniklými pravidly. Pravidla buduje na základě výběru jednoho příkladu z H skupiny, pro který vytvoří sadu nejlépe jej popisujících specifických pravidel, která nekolidují s příklady z L skupiny, a následně vybere nejlepší pravidlo, které se stane součástí finální hypotézy. Dále se mu zde nebudu věnovat, protože v další části projektu nebude implementován.

6.2.3 Algoritmy CN2, PRISM

Z ostatních algoritmů induktivního učení stojí za zmínku kladně hodnocené na pravidlech založené algoritmy strojového učení CN2 [23] a PRISM [20]. Oba na výstupu generují sadu pravidel popisujících H skupinu stejně jako předchozí AQ algoritmus, ale zdají se na první pohled sofistikovanější v postupu generování výsledné hypotézy nebo naopak jednodušší funkci a teoreticky rychlejší. Jak jeden, tak druhý, by mohly mít co nabídnout jako jednak sofistikovanější varianta pro kvalitnější pravidla, tak jako jednodušší varianta pro rychlejší učení.

CN2

```
Let E be a set of classified examples.
Let SELECTORS be the set of all possible selectors.
Procedure CN2(E)
       Let RULE-LIST be the empty list.
       Repeat until BEST.CPX is nil or E is empty:
              Let BEST.CPX be Find-Best.Complex(E).
              If BEST.CPX is not nil,
              Then let E' be the examples covered by BEST.CPX.
                     Remove from E the examples E' covered by BEST.CPX.
                     Let C be the most common class of examples in E'.
                     Add the rule 'If BEST.CPX then the class is C' to the
                     end of RULE-LIST.
       Return RULE-LIST.

Procedure Find-Best.Complex(E)
Let STAR be the set containing the empty complex.
Let BEST.CPX be nil.
While STAR is not empty,
       Specialize all complexes in STAR as follows:
       Let NEWSTAR be the set {x and y|x € STAR, y € SELECTORS}.
       Remove all complexes in NEWSTAR that are either in STAR (i.e., the
       unspecialized ones) or null (e.g., big = y A big = n).
       For every complex Ci in NEWSTAR:
              If Ci is statistically significant and better than
                     BEST.CPX by user-defined criteria when tested on E,
              Then replace the current value of BEST.CPX by Ci.
       Repeat until size of NEWSTAR < user-defined maximum:
              Remove the worst complex from NEWSTAR.
       Let STAR be NEWSTAR.
Return BEST.CPX.
```

Obrázek 6.9: *Pseudokód algoritmu CN2 [23]*

V pseudokódu výše označují *SELECTORS* uvažováné intervaly v prostoru řešení, *BEST:CPX* je nejlepší nalezené jedno pravidlo a *complex* značí momentálně uvažované pravidlo. Jak je vidět, algoritmus CN2 využívá obou přístupů k učení pravidel zmíněných výše, čili jak postupnou specializaci a hledání pravidel ne nepodobně jako v algoritmu AQ, tak ohodnocení kvality nalezené skupiny pravidel pomocí vzorců pro výpočet entropie a informačního přínosu jako u algoritmu ID3 (při porovnávání kvality skupin pravidel). Tento přístup se zdá jako velmi slibný a mohl by být přínosnou obměnou při reimplementaci LEM metody. Výpočetně je asi nejsložitější ze všech uvažovaných metod, ale pokud by se dařilo spolehlivěji identifikovat slibné regiony prostoru řešení, byl by tento nedostatek vyvážen.

PRISM

Algoritmus PRISM nabízí odlišný přístup ke generování pravidel hypotézy. Postupně vytváří prázdná pravidla a snaží se je rozvíjet přidáváním podmínek, které pokrývají co největší počty vzorků dokud nedosáhne pokrytí všech vzorků žádané třídy. Výhodou algoritmu je jeho jednoduchost a rychlost, jako nevýhodu vidím především jeho zaměření pouze na jednu třídu (v našem případě třeba H skupinu) a neuvažování zkreslení výsledku vlivem konfliktních pravidel, která pokrývají i příklady z L skupiny. Nicméně čistě pro účely testování, zda dobrá rychlost nalezení hypotézy nevyváží ne až tak kvalitní výsledná pravidla, bude tento algoritmus uvážen pro implementaci v další části projektu, kde bude i nadále podrobněji rozebrán.

6.3 Neuronové sítě

6.3.1 Restricted Coulomb Energy neuronová síť

Neuronové sítě se ukázaly jako mocný nástroj pro mnoho výpočetních úloh v oblastech, řízení, rozpoznávání vzorů či klasifikace. Z počátku jsem je nepovažoval za vhodné pro tento projekt, ale narazil jsem na variantu, která by se mohla prokázat jako velmi efektivní v identifikaci slibných oblastí v prohledávaném hyperprostoru. Jedná se tedy o variantu neuronové sítě – počítačovému modelu založeném na struktuře lidského mozku [3], která se od ostatních modelů odlišuje svou proměnnou topologií během učení, ale především bude snadno použitelná pro generování nových jedinců, kteří respektují model naučený neuronovou sítí.

Jednotlivé neurony v síti jsou řízeny radiální bázovou funkcí a skokovou aktivační funkcí. Celý algoritmus učení sítě a související informace je popsán v související kapitole 7.1.1.

6.4 Bayesovská klasifikace

Snadno implementovatelné klasifikátory založené na Bayesovském logickém pravidle[3][19] se v mnoha případech ukázaly jako rychlé a efektivní. Pracují s myšlenkou, že při vzorku vstupních dat X a možných třídách $C_i \dots C_m$ je prvek zařazen do té třídy, pro niž je podmíněný pravděpodobnostní výraz $P(C_i|X)$ maximální.

$$P(C_i \mid X) = \frac{P(X \mid C_i)P(C_i)}{P(X)} \tag{6.14}$$

Pokud si napíšeme Bayesův vzorec pro daný problém, můžeme vidět, že ve skutečnosti hledáme maximální podmíněnou pravděpodobnost, že je vybrán daný vzorek, když víme, že patří do třídy C_i vynásobeno pravděpodobností vybrání příslušné třídy. Protože P(X) je konstantní, zajímá nás pouze výpočet čitatele pravé strany vzorce (5.14)

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i) \tag{6.15}$$

Pravděpodobnost pro náležitost do třídy C_i je poměrem všech vzorků z dané třídy a celkového počtu vzorků. A druhý činitel si vypočteme podle vztahu 5.15 jako součin jednotlivých

podmíněných pravděpodobností jednotlivých hodnot atributů. V tomto okamžiku se nám rýsují dva možné další postupy:

1. uvažovaný atribut je diskrétní
    - toho dosáhneme, pokud budeme postupovat podobně jako u předchozích klasifikačních metod, čili rozdělíme si spojity prostor na intervaly podle současně nalezených řešení a ty budeme používat jako atributy
    - budeme hledat počty do třídy zařazených vzorků s konkrétní hodnotou atributu
2. uvažovaný atribut je spojitý
    - toho dosáhneme, pokud neprovedeme předem naznačené kroky binarizace a diskretizace spojité veličiny nebo provedeme, ale nebudeme uvažovat rozdělení dimenzí na intervaly, ale budeme binární hodnoty převádět na spojité
    - budeme počítat s hodnotami gaussovy normální funkce, pro kterou si spočítáme standardní odchylku a průměr pro jednotlivé třídy

Oba tyto přístupy se mi jeví jako případně hodny prozkoumání, implementace a porovnání úspěšnosti. Principem má použití této metody klasifikace hodně blízko k EDA algoritmům, které byly popsány dříve v textu – také se zde bere především ohled na pravděpodobnostní charakter řešení. Po spočítání všech podmíněných pravděpodobností z dostupných dat poté můžeme generovat nová řešení třeba tak, že vybereme pro každou dimenzi vektoru řešení atribut nebo atributy (intervaly) s nejvyššími pravděpodobnostmi a generujeme jedince s parametry náhodně zvolenými z těchto intervalů. Na druhou stranu při numericé optimalizaci se zpravidla setkáváme s komplexními funkcemi v definici obsahujícími závislosti mezi jejich parametry a tam už by základní pravděpodobnostní klasifikátory nemusely stačit pro správné namodelování požadovaných vlastností jednotlivých tříd.

7 Vlastní implementace LEM

Po provedení průzkumu cílové problematiky – numerické optimalizace, evolučních přístupů, EDA algoritmů a metod strojového učení byly identifikovány klíčové aspekty, kterým bude potřeba věnovat pozornost při dalším pokračování práce – zvolená a implementovaná vhodná a nejlépe adaptivní diskretizace spojité veličiny, implementace ideálně všech součástí metody, volba a implementace různých vhodných klasifikačních metod a jejich použití v LEM, otestování na složitých testovacích funkcích a srovnání výkonnosti

V následujícím textu bude obsažen rozbor vlastního návrhu a implementace LEM metody, zvolené přístupy k jednotlivým součástem cyklu algoritmu.

Pro svou verzi Learnable Evolution Model algoritmu jsem zvolil implementační jazyk Java verze 6 bez podpory jakýchkoliv pomocných knihoven (vyjma případných pomocných balíčků pro grafické zobrazení průběhu evoluce, což ale není nezbytnou součástí programu). Program je koncipován pro spuštění na libovolné platformě. Teď ale blíže k detailům implementace.

7.1 Volba modulu strojového učení

Při volbě součásti programu starající se o akce učení se hypotézy rozlišující kvalitní a nekvalitní řešení byly uvažovány možné algoritmy/přístupy z těchto hledisek:

- algoritmus dovoluje rozlišovat různé oblasti v obecně hyperprostoru řešení
- algoritmus nebyl doposud použit pro učení v LEM
- algoritmus poskytuje možnost na základě naučeného klasifikátoru vygenerovat nové řešení

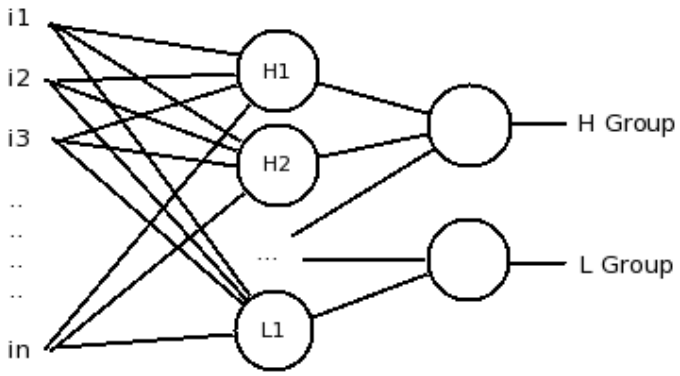
Po vynechání přístupů zmíněných v kapitole 4.2 jsem se rozhodnul z uvažovaných algoritmů nastíněných v kapitole 6 vybrat tyto: RCE neuronovou síť a algoritmus PRISM. Ostatní přístupy nebyly uvažovány z důvodu, že buďto už byly jednou s LEM použity, jevily se příliš podobné už použitým nebo se nezdály vhodné pro jeden z úkonů zmíněných výše.

7.1.1 Neuronová síť Restricted Coulomb Energy

Úvod k tomuto přístupu byl už uveden v kapitole 6.3.1, zde si tedy představíme tuto neuronovou síť více do hloubky. Tato metoda splňuje všechny tři výše uvedené požadavky pro použití v LEM metodě jako modul strojového učení. Nemalou váhu při jeho zvolení měla také jeho relativní exotičnost – ne až tak časté využití a jiný přístup k segmentaci hyperprostoru na intervaly oproti ostatním přístupům – a jednoduchost konceptu.

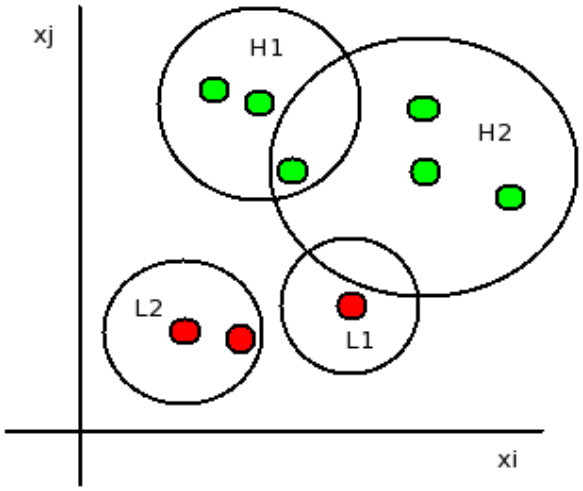
Struktura sítě, výsledná segmentace prostoru

Hotovou síť máme tvořenou na vstupy (jednotlivé dimenze vstupního vektoru) napojenou skrytou vrstvou, kde každý neuron reprezentuje jednu z hyperkoulí pokrývající jí přiřazené body dané třídy a neurony výstupní, reprezentující třídu, které jednotlivé skryté neurony náleží. Nejlépe je to asi vidět na obrázku 7.1 níže.



Obrázek 7.1: *Ukázka struktury RCE sítě*

Jako každý klasifikátor i tato neuronová síť ve výsledku jistým způsobem poskytuje pravidla o určité třídě/segmentuje prostor řešení na regiony náležející třídám. Nejsnáze se to asi ukáže na zjednodušené ilustraci pro dvě dimenze (Obrázek 7.2), kde je vidět, že pravidla pro prvky náležející jednotlivým třídám jsou ve formě neuronů a jejich parametrů v prostoru



Obrázek 7.2: *Ukázka pokrytí řešení neurony RCE sítě*

Učení sítě

Jak už bylo zmíněné v kapitole 6.3.1, tato neuronová síť se vyznačuje radiální bázovou funkcí a skokovou aktivační funkcí. Každý neuron je charakterizován svými vahami, které prakticky odpovídají souřadnicím v kartézském prostoru a poloměrem hyperkoule náležící neuronu, který vlastně představuje práh pro aktivační funkci. Bazová funkce vlastně odpovídá euklidovskému vzorečku pro výpočet vzdáleností dvou bodů v prostoru (rovnice 7.3):

$$u_k = \|\vec{i} - \vec{u_k}\| = \sqrt{\sum_{i=1}^{n} (i_i - u_{ki})^2} \tag{7.3}$$

Co se samotného učení týče, algoritmus je uveden na obrázku 7.4. Pro všechna řešení (vzorky pro učeni) postupně vytváří/upravuje neurony (tedy hyperkoule) v prostoru dokud dochází ke špatnému přiřazování vzorků. Po skončení učení máme k dispozici strukturu takovou, jako byla například ukázána výše v obrázku 7.2.

1. Všechny vrstvy jsou prázdné.
2. Opakuj dokud není hodnota proměnné *modif* rovna *true*:
   - nastav proměnnou *modif* na hodnotu *false*
   - pro všechny vstupní vektory:
     1. nastav proměnnou *hit* na *false*
     2. pro každý neuron:
        - Vypočítej vzdálenost mezi středem hyperkoule zastupované neuronem a vstupním vzorkem.
        - Pokud je vzdálenost menší, než je poloměr hyperkoule a pokud je třída reprezentovaná vzorkem stejná jako třída reprezentovaná nauronem, nastav proměnnou *hit* na hodnotu *true*
        - Pokud je vzdálenost větší, zmenši poloměr hyperkoule tak, aby vzorek nebyl zahrnut do neuronu a nastav hodnotu proměnné *modif* na *true*
     3. Pokud je hodnota proměnné *hit* rovna *false*
        - Vytvoř nový neuron se souřadnicemi vstupního vzorku a stanoveným maximálním poloměrem.
        - Nastav proměnnou *modif* na *true*

Obrázek 7.4*: Algoritmus učení RCE neuronové sítě [22]*

Generování nových řešení

Chceme-li generovat nová řešení na základě námi vytvořeného modelu RCE sítě, nabízí se asi nejjednodušší postup, kdy vybereme jeden z neuronů a na základě jeho parametrů (pozice a poloměr) vygenerujeme náhodný bod v jím představované hyperkouli. Postup je analogický generování náhodného bodu uvnitř dvourozměrné kružnice (nestačí nám generovat náhodné body v rámci poloměru, mohlo by snadno dojít k přesahu v nějaké dimenzi) s tím, že pro každou další dimenzi je jen potřeba dopočítat úhel viz. obrázek 7.5 a rovnice 7.6 a 7.7. Je tedy potřeba náhodně vygenerovat vzdálenost nového bodu R a poté všechny potřebné úhly pro každou dimenzi zvlášť.

$$x_1 = R\cos\theta_1$$
$$x_2 = R\sin\theta_1\cos\theta_2$$
$$x_3 = R\sin\theta_1\sin\theta_2\cos\theta_3$$
$$x_4 = R\sin\theta_1\sin\theta_2\sin\theta_3\cos\theta_4$$
$$\dots$$
$$x_{N-1} = R\sin\theta_1\sin\theta_2\sin\theta_3\dots\sin\theta_{N-2}\sin\theta_{N-1}$$
$$x_N = R\sin\theta_1\sin\theta_2\sin\theta_3\dots\sin\theta_{N-2}\cos\theta_{N-1}$$

Obrázek 7.5: *Postup odvození rovnic 7.6 a 7.7 pro generování náhodných bodů uvnitř hyperkoule*

$$x_k = R\cos\theta_k\prod_{i=1}^{k-1}\sin\theta_i \qquad (7.6)$$

$$x_N = R\prod_{i=1}^{N-1}\sin\theta_i \qquad (7.7)$$

7.1.2 Algoritmus PRISM

Jak již byl zběžně představen v kapitole 6.2.3, jedná se o algoritmus generující na výstupu množinu pravidel, která popisují danou na vstupu danou třídu vzorků. Níže na obrázku 7.9 si můžeme prohlédnout postup algoritmu při vytvážení konečné množiny pravidel.

Nutno dodat, že před každým učícím cyklem je potřeba provést tvorbu atributů pro algoritmus. U problémů se symbolickými atributy tento krok není potřeba, ale u numerické optimalizace nemáme v datech žádné explicitní kategorie pro klasifikaci prvků/řešení. V následující podkapitole si ukážeme, jak si takovéto atributy obstaráme.

Tvorba atributů ze spojité veličiny

Prakticky se dají uvažovat (a používají se) dva způsoby konverze spojité veličiny do formy symbolického atributu (reprezentujícího určitý interval). První způsob byl popsán v sekci 2.3.2, druhý spočívá v rozdělení oboru hodnot dimenze spojité veličiny na pevný počet intervalů nezohledňujíc "šum" uvnitř intervalů jak je ilustrováno v tabulce 7.8. Toto je provedeno pro každou dimenzi vstupní veličiny a s výslednými intervaly, definovanými hraničními hodnotami nebo např. vzdálenostmi mezi sousedícími hodnotami, je pracováno.

12 13 14 15	16	17	18 19 20	21 22	23
H	L	H	L	H	L
12 13	14 15	16 17	18 19	20 21	22 23
H	H	L H	L	L H	L

Tabulka 7.8: *Ukázka obou možných přístupů k tvorbě atributů*

Pro vlastní implementaci byl zvolen druhý přístup i z toho důvodu, aby mohlo být generováno více méně specifických pravidel.

Učení algoritmu

```
for each class C
      initialize E to the instance set
      repeat
            p = true
            create a rule R with an empty left-hand side that predicts
            class C:
                  R=(if p then C)
            repeat
                  for each (attribute, value) pair found in E
                        consider adding the condition A=v to the Rule R
                  find A=v that maximizes the accuracy p/t
                        (break ties by choosing the condition with the
                        largest p)
                  Add A=v to R
            until R is perfect (or there are no more attributes to use)
            Remove the instances covered by R from E
      until E contains no more instances that belong to class C
```

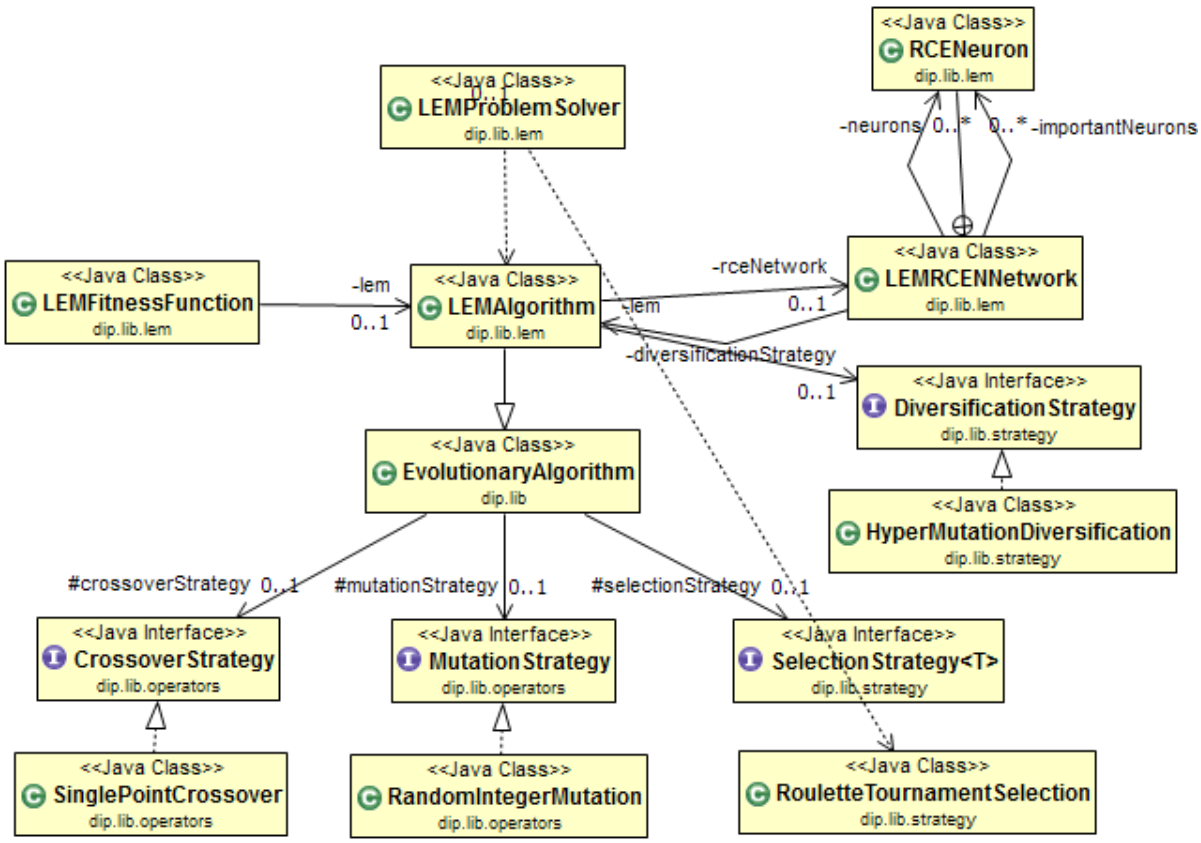
Obrázek 7.9: *Pseudokód algoritmu PRISM [20]*

Jak je tedy vidět, algoritmus postupuje relativně přímočaře – postupně vybírá jeden z atributů (v našem případě intervalů pro každou dimenzi) a postupně přidává do pravidla atributy dokud nepokryje všechny prvky (řešení), které byly pokryty prvotně vybraným atributem. Když se mu to povede, odstraní tyto prvky náležející novému pravidlu z trénovací množiny a pokračuje v hledání dalšího pravidla se zbylými prvky dokud zbývají nějaké prvky, které ještě nejsou pokryté žádným pravidlem.

Generování nových řešení

Nová řešení se v případě PRISM algoritmu budou vytvářet obdobně jako u původní verze LEM metody a jejího algoritmu AQ. Mějme nějakým způsobem vybrané pravidlo, které nám vymezuje jisté slibné intervaly pro určité atributy (dimenze) prostoru řešení. Z těchto intervalů atributů náhodně zvolíme hodnoty a pokud pro některé dimenze řešení není interval v pravidle přítomen, můžeme zvolit náhodné hodnoty anebo, jak se i doporučuje v [10], zvolit hodnoty z již existujících náhodných jedinců v populaci a přiřadit je novému jedinci.

7.2 Struktura programu



Obrázek 7.10*: Diagram hlavních tříd pro finální program*

Na obrázku 7.10 je vidět hlavní struktura implementovaného LEM programu a využití jak tříd s funkčností typickou pro tradiční evoluční přístupy, tak začlenění funkčnosti třídy strojového učení. Nemělo by být potřeba zabíhat do detailů při popisu jednotlivých tříd, všechny možné potřebné informace by měly být na přiloženém CD v dokumentaci.

7.3 Dodatky k implementaci

Nepodařilo se mi spolehlivě/nestihl jsem naimplementovat tyto funkce/součásti LEM metody zmíněné v původních pracech:

- adaptivní diskretizace – místo ní v algoritmu proběhne jen mutace
- dynamická změna počtu generovaných jedinců pomocí učícího jádra/evolučních algoritmů
- podpora více možných tříd kvality jedinců pro učení
- mechanismy pro průběžnou kontrolu fitness profilu populace
- mechanismy pro průběžnou kontrolu stavu učícího modulu – řízené učení z několika generací, používání

Adaptivní diskretizace původně byla plánovanou součástí, proto je pořád ve zdrojovém kódu a programu využíváno mapovacího pole z přirozených číselných hodnot na reálné a stejně tak genotyp jedinců v populacích je tvořen přírozenými čisly pro snadné indexování v mapovacím poli. Stejně tak je v kódu často počítáno s více možnými skupinami dle kvality fitness funkce, ale není toho nikdy využito pro učení/tvorbu nových jedinců. Vynechání spolehlivé implementace adaptivní diskretizace se dle mého názoru podepsalo největší měrou na nízké kvalitě programu. Jak algoritmus PRISM, tak RCE síť se v jistém okamžiku nemají moc kam s generováním nových jedinců pohnout a dochází ke generování ne moc přínosných nových řešení. U přístupu s algoritmem PRISM je navíc problém, že při použití pevně daných intervalů, které zahrnují řešení jak z H, tak z L skupiny, dochází k tvorbě jedinců z vlmi úzce specializovaných pravidel - často taky téměř identických řešení a algoritmus se nemůže z tohoto stavu rozumně dostat (o nějaké resp. může ale po nějaké době se dostane do podobné situace). Generování identických nebo témě identických řešní je také probém RCE sítě v okamžiku, kdy se poloměr nějakého neuronu zmenší na dostatečně malou velikost, kdy neposkytuje dost geometrického prostoru pro vygenerování určitého počtu unikátních jedinců.

Oba zvolené přístupy k strojovému učení se tedy ukázaly jako velice nestálé, na vině jsou z velké míry ústupky oproti popsaným verzím původní LEM metody. Kromě zmíněné diskretizace, pokud nedochází ke kontrole fitness profilu populace (resp. dají li se snadno rozlišovat H a L skupiny a rozdíl mezi jejich průměrnými fitness hodnotami je znatelný) dochází k narušování kvality generovaných hypotéz. Nastávají tak například u RCE sítě situace, kdy je jeden a ten samý neuron střídavě přiřazován jedné i druhé fitness skupině.

8 Experimenty

Veškeré experimenty probíhaly na počítači s dvoujádrovým procesorem 2.2 Ghz a 4GB RAM. Každé měření bylo provedeno desetkrát a byl vzat průměr viz. tabulky v přílohách této práce. Implementtace byla testována jen na maximálně desetidimenzionálním oboru hodnot testovaných funkcí. Během testů pro deset proměnných se ukázalo, jak neuspokojivé výsledky program podává a další testování bylo uznáno za zbytečné.

Pro porovnání bylo využito hodnot naměřených pro různé varianty EDA algoritmů v [24]. Parametry LEM algoritmu byly pevně stanoveny na tyto hodnoty:

- velikost populace - 100, pravděpodobnost křížení - 0.7, pravděpodobnost mutace - 0.1
- mocnosti H a L skupiny - 30%, iterace bez pokroku – 10, počet operací probe před zpřesněním diskretizace (mutací) – 10, počet zpřesnění diskretizace – 4, maximální počet restartů metody – 4

Maximální počet generací byl vždy nastaven tak, aby algoritmus skončil po provedení stejného množství vyhodnocení fitness funkce jako nejlepší EDA řešení z [24] a naměřené hodnoty mohly být snadno porovnány.

Co měříme?

Pro účely srovnání bude měřeno nejlepší nalezené řešetí za daný počet evaluací fitness funkce (je počítáno X generací vynásobeno počtem zavolání výpočtu fitness funkce v každé generaci), což nám dovoluje porovnat i algoritmy původně pracující s jinými velikostmi populace atd. Dále je také pro orientaci změřen i celkový čas optimalizace, nicméně ten je v objektivním měřítku dost závislý na platformě, na které program běží, na implementačním jazyku atd. čili nejedná se až tak o směrodatný údaj pro hodnocení kvality algoritmu.

Dále bude také zohledněna praktická metrika δ-blízkosti řešení. Jedná se o normalizované hodnocení kvality algoritmem nalezeného optima, jak je vysvětleno např. v [10], kdy *opt* značí hodnotu hledaného globálního optima, *v(s)* značí námi dosaženou hodnotu a *init* značí nejlepší hodnotu v první generaci algoritmu. Tato metrika nám říká, jak blízko se náš algoritmus dostal k hledanému optimu a je velmi pěknou měrou k hodnocení optimalizačních algoritmů.

$$\delta(s) = \frac{|opt - v(s)|}{|opt - init|} \tag{8.1}$$

8.1 Měření výsledků LEM metody s RCE sítí

Ve zveřejněných tabulkách v této kapitole bude prezentovno srovnání získaných experimentálních výsledků mnou realizované LEM varianty a výsledků získaných pomocí EDA algoritmů z [24]. Pro kompletní soupis měření doporučují nahlédnout do příloh, zde bude pro zpřehlednění uváděna vždy jen průměrná hodnota po desíti měřeních.

8.1.1 První de Jongova funkce

	Best fitness	Evaluations
UMDAc	6,7360E-06	74164
MIMICc	7,2681E-06	74964
EGNAbic	2,5913E-05	77162
EGNAbge	7,1938E-06	74764
EGNAee	7,3713E-06	73964
EMNAglobal	7,3350E-06	94354
EMNAa	4,8107E+04	301000
LEM-RCE	2,7198E+03	74000

Tabulka 8.1*: Srovnání výsledků LEM / EDA na de Jongově funkci*

Jak se ukázalo, má implementace byla opravdu navržena/naprogramována velice špatně. Kromě jednoho (nicméně $EMNA_a$ se ukázala být jako nejhorší varianta při všech testech) byly všechny uvedené EDA metody mnohem lepší. Dle sesbíraných dat by pravděpodobně má LEM metoda nezlepšila výsledek ani při delší době evoluce. Jedná se přitom a na pohled velmi jednoduchou a k tomu unimodální funkci.

8.1.2 Rosenbrockova funkce

	Best fitness	Evaluations
UMDAc	8,7204E-02	301850
MIMICc	8,7141E-02	301850
EGNAbic	8,8217E+00	268067
EGNAbge	8,6807E-02	164519
EGNAee	8,7366E-02	301850
EMNAglobal	8,7201E-02	289056
EMNAa	3,2630E+03	301000
LEM-RCE	54.863124	164500

Tabulka 8.2*: Srovnání výsledků LEM / EDA na Rosenbrockově funkci*

K obdobných závěrům jsem došel při testování na další z funkcí – znovu jen $EMNA_a$ je horší než má variace LEM metody a ostatní s vyjímkou $EGNA_{BIC}$ jsou znovu o několik tříd dále s kvalitou nalezeného řešení. Je nepravděpodobné, že by se výsledek o mnoho zlepšil při doběhnutí LEM algoritmu do s některými EDA přistupy rovnocenných 301850 evaluací.

8.1.3 Ackleyho funkce

	Best fitness	Evaluations
UMDAc	7,8784E-06	114944
MIMICc	8,8351E-06	114744
EGNAbic	5,2294E+00	229086
EGNAbge	7,9046E-06	113944
EGNAee	7,4998E-06	118542
EMNAglobal	8,9265E-06	119141
EMNAa	1,0885E+01	301000
LEM-RCE	2,1975E-02	114900

Tabulka 8.3*: Srovnání výsledků LEM / EDA na Ackleyho funkci*

Zlehka optimističtější výsledek byl dosažen u Ackleyo funkce, kdy LEM s RCE sítí předvedl lepší výkony než dvě varianty EDA – znovu teddy $EMNA_a$ metoda a přidala se k ní $EGNA_{BIC}$, která víceméně ve všech testech pokulhávala za ostatními, jen zde se to projevilo natolik, aby dosáhla horšího výsledku než má LEM implementace. Nutno dodat, že sice obě jsou na tom o hodně hůře než můj algoritmus, nicméně ostatní varianty EDA jsou naopak tak bližko globálnímu optimu, že znovu se prokazuje velká neefektivnost mé implementace.

8.1.4 Griewangkova funkce

	Best fitness	Evaluations
UMDAc	6,0783E-02	301850
MIMICc	7,3994E-02	301850
EGNAbic	3,9271E-02	301850
EGNAbge	7,6389E-02	301850
EGNAee	5,6840E-02	301850
EMNAglobal	5,1660E-02	301850
EMNAa	1,2941E+01	301850
LEM-RCE	6,7395E-01	300000

Tabulka 8.4*: Srovnání výsledků LEM / EDA na Griewangkově funkci*

Poslední z testovaných funkcí, a asi i ta nejsložitější na optimalizaci, přinesla už asi očekávaný scénář – LEM s RCE prohrála ve srovnání se všemi variantami kromě EMNAa, nicméně rozdíly mezi nalezenými optimy nebyly až tak markantní jako u předchozích pokusů, což je ale pravděpodobně zaviněno vlastním charakterem optimalizované funkce, která má spoustu lokálních optim ne až tak hodnotově vzdálených od sebe.

8.2 Výsledky PRISM metody, souhrn a analýza

8.2.1 Souhrn a diskuse k dalším výsledkům

Naměřeným výsledkům získaným PRISM algoritmem nebude věnován takový prostor – přístup se v mé implementaci neukázal jako vůbec efektivní, jak bude ostatně ukázáno dále. Pro bližší detaily o naměřených hodnotách doporučuji nahlédnout do příloh

Srovnání se stávajícími implementacemi LEM metody nebylo v zadání práce vyžadováno, nicméně jak bylo již zmíněno, v [10] bylo ukázáno, že např. LEM3 metoda nabízí oproti EDA algoritmům výrazné zrychlení – především i ve srovnání s nejvýkonnějším ze zmíněných $EMNA_{global}$ – čili s ním se má implementace už vůbec nemůže rovnat.

Podívejme se teď tedy souhrnně na ostatní naměřené hodnoty v tabulce 8.5:

	PRISM			RCE	
	best found	**δ-close**	**time**	**δ-close**	**time**
Sphere	2.4483E+04	0.053	0.635	0.0062	2.720
Rosenbrock	3.7131E+03	0.015	0.646	0.0003	5.365
Griewangk	6.4405E+00	0.045	1.115	0.0062	9.661
Ackley	4.5146E-01	0.162	1.132	0.0080	4.880

Tabulka 8.5*: Souhrnné zobrazení některých výsledků metod (čas je v minutách)*

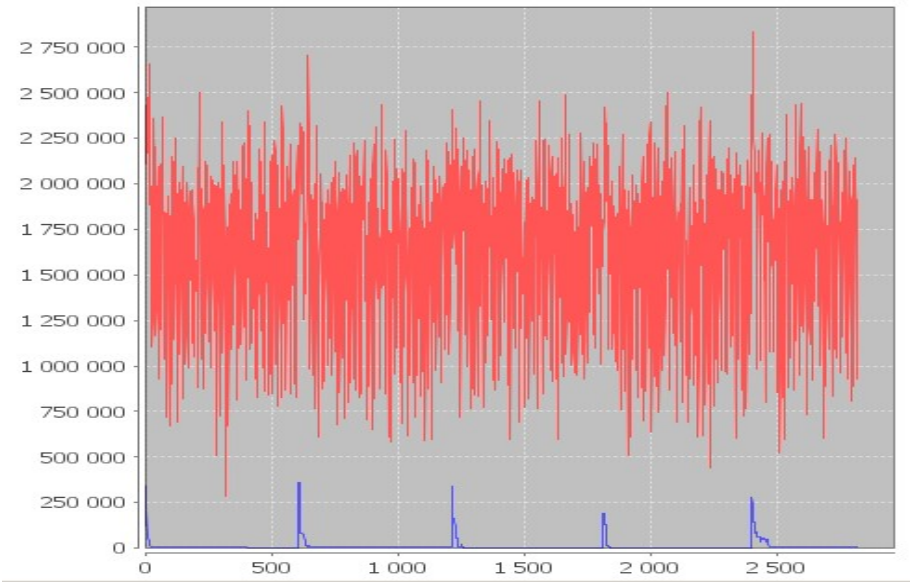
Jak je vidět z tabulky nejlepších dosažených optim za pomoci metody PRISM, dosáhla lepších výsledků než $EMNA_a$ $EGNA_{BIC}$ u Ackleyho funkce, ale jinak dosahovala jen horších výsledků jak v porovnání s LEM s RCE sítí, tak znovu drasticky horších výsledků ve srovnání s ostatními EDA variantami.

Co se časové náročnosti týče, můžeme při srovnání vycházet pouze z tabulky hodnot uvedené v [24] a z toho by vyplynulo, že LEM s PRISM je zdaleka nejrychlejší ze všech variant a i v porovnání s RCE variantou, nicméně zase nedosahuje vůbec slibných výsledků optimalizace, kdy ani v jednom případě nebyl schopný se za daný počet evaluací dostat alespoň do 0.01 δ-vzdálenosti hledanému optimu. Co se časové náročnosti souhrnně týče, RCE a EDA varianty jsou srovnatelné, jen u Griewangkovy funkce trvala optimalizace u mé implementace LEM přibližně dvakrát tak dlouho jako nejdelší časy jiných metod/funkcí. U RCE přístupu jsou výsledky a vyhlidky o mnoho příznivější. Prakticky ve všech porovnáních s EDA algoritmy neuspěl, ale když se podíváme na hodnoty δ, vždy byl algoritmus schopný dosáhnout lepší než 0.01 δ-vzdálenosti od hledaného globálního optima a v případě Rosenbrocka dokonce dosáhl daleko za hranici 0.001 δ-vzdálenosti.

8.2.2 Analýza běhu optimalizace

Jen rychle se podíváme ještě na grafický průběh optimalizace pro oba zvolené algoritmy strojového učení a celkové chování LEM algoritmu na jedné z testovaných funkcí. V grafech na obrázcích 8.6 a 8.8 je červenou barvou znázorněna aktuální hodnota průměrné objektivní funkce v populaci, modrou pak nejlepší jedinec v populaci. Na vodorovné ose je vyvedeno číslo generace, na svislé hodnota objektivní funkce.

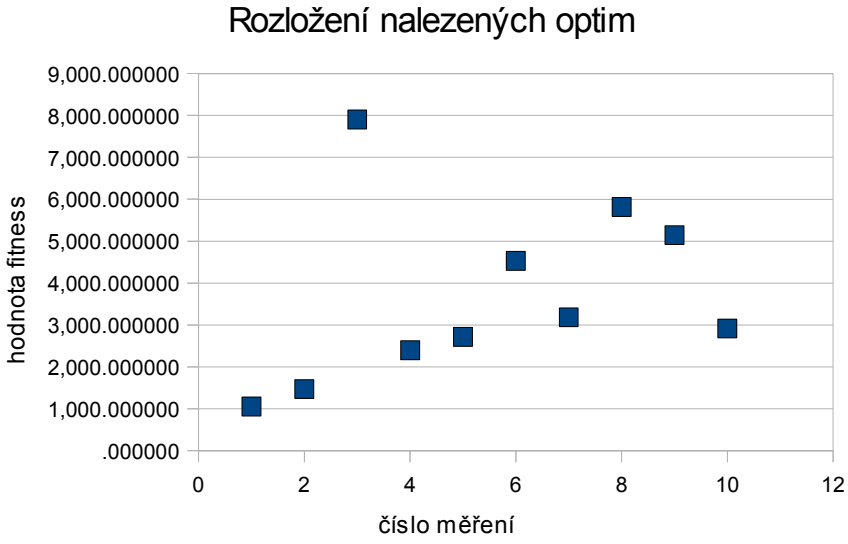
PRISM algoritmus a Rosenbrockova funkce



Obrázek 8.6*: Průběh optimalizace - PRISM*

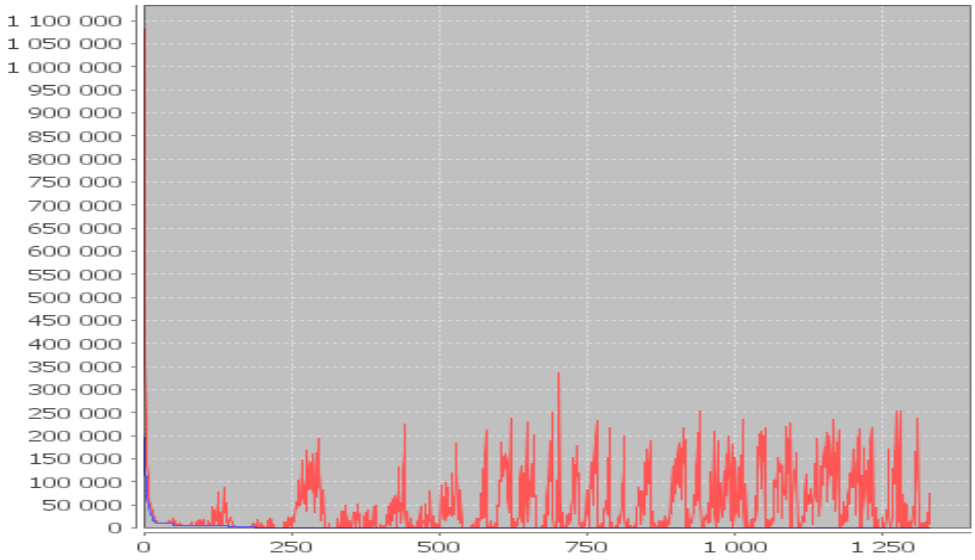
Jak můžeme pozorovat na obrázku 8.6 pří použití PRISM algoritmu rozhodně populace netrpí nedostatkem hodnotové diverzity. Je až s podivem, že s takto charakterizované populace algoritmus dokáže generovat jedince s dobrým fitness ohodnocením. Na druhou stranu se nijak neblíží žádanému globálnímu optimu, po každém z restartů algoritmu se velice brzy dostane do jednoho z lokálních optim a nedokáže z něj vybřednou i přes četné pokusy o mutaci a ostatní metody generování nových řešení včetně častých restartů metody, protože nedochází k pokroku.

Program se zhruba takto choval při každém optimalizačním běhu, nikdy nedocházelo k nějakému řízenějšímu postupu k dosažení lepšího řešení. Jak je vidět i na grafu na obrázku 8.7, nalezená optima ležela někdy hodnotově velice daleko od sebe, což jen podtrhuje nestálou povahu optimalizace za pomoci PRISM.



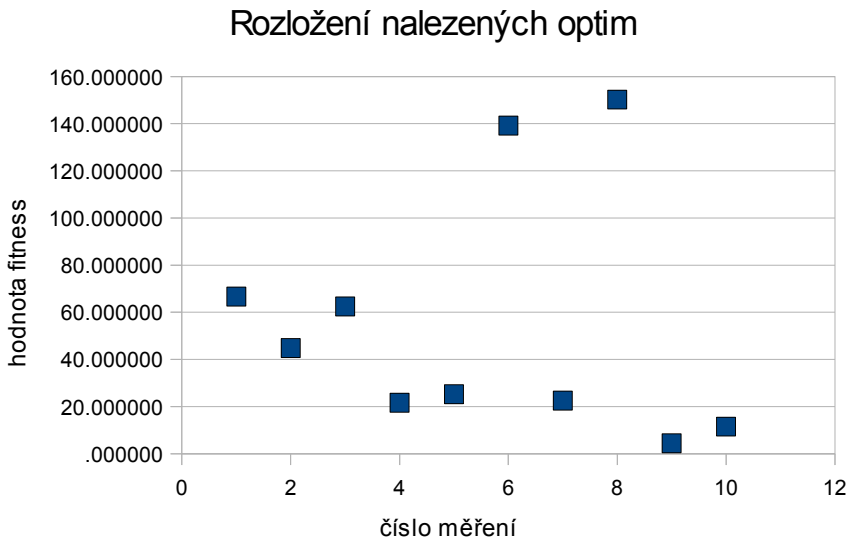
Obrázek 8.7*: Rozložení nalezených optim*

RCE algoritmus a Rosenbrockova funkce



Obrázek 8.8*: Průběh optimalizace – RCE*

Na první pohled jsou hned vidět některé rozdíly oproti metodě PRISM. K nutnému restartu algoritmu nedochází zdaleka tak často, vždy se podaří alespoň nějakým způsobem nalézt lepší řešení. Společná je také počáteční rychlá konvergence a poté už jen velmi pomalý postup, dále a stále k novému lokálnímu optimu, který je u RCE sítě sice velmi pozvolný, ale je tu. Diverzita je udržována poměrně stabilně a i když je z pohledu na obrázek 8.9 vidět také znatelný rozptyl nalezených hodnot, nejedná se už o řád tisíců ale jen desítek, což je sice pořád hodně, ale vzhledem ke některým ústupkům v implementaci se dá akceptovat.



Obrázek 8.9*: Rozložení nalezených optim - RCE*

9 Závěr

V rámci diplomové práce byla nastudována problematika numerické optimalizace a především jeden ze slibně se chovajících algoritmů pro řešení souvisejících problémů – Learnable Evolution Model – založený na kombinaci tradičních přístupech založených na evolučním programování kombinovaných s metodami/algoritmy strojového učení typických spíše pro odvětví klasifikace, rozpoznávání či dolování z dat. Pro implementaci byly vybrány dva doposud v rámci metody a jejích odvozenín neimplementované nadějné metody strojového učení – neuronová síť Restricted Coulomb Energy a algoritmus PRISM.

Algoritmus LEM byl bohužel primárně vinou vlastní nespolehlivosti a zbrklosti nakonec implementován bez několika mechanismů, které byly některé v článcích původních autorů metody zmiňovány – výčet ústupků a nedokončených součástí je v kapitole 7.3. Od kompromisů při vytváření programu se pak nadále odvíjely problémy s laděním částí starajících se o strojové učení a dá se říct, že doteď se nepodařilo uzpůsobit program tak, aby poskytoval kvalitní podklady pro naučení se rozumné hypotézy a následné vygenerování nových přínosných řešení. Nejvíce tím trpěl algoritmus PRISM, který dosahoval velice špatných výsledků v porovnání s RCE sítí, která také na konci nepracovala tak, jak by bylo pro bezproblémový chod optimalizace potřeba.

Po otestování výkonnosti programu na funkcích o desítí proměnných byly výsledky porovnány s hodnotami dosaženými pro obdobné problémy pomocí EDA algoritmů a výsledky dopadly víceméně podle očekávání. Mnou navržené dvě varianty LEM metody dopadly skoro ve všech testech hůře než srovnávané EDA metody. Výsledkem překonaly u všech testovaích funkcí jen jeden z uvažovaných EDA algoritmů, nicméně ten zaostával v úspěšnosti/výkonnosti už v porovnání s podobnými algoritmy, čili se tomuto výsledku nedá přisuzovat jakákolilv míra důležitosti. Když se člověk zabýval ostatními měřítky výkonnosti, situace už nevypadala tak zle, obzvláště varianta LEM využívající RCE síť se dokázala v měřítcích normované vzdálenosti od globálního optima přiblížit hledanému cíli na poměrně dobré vzdálenosti, nicméně v porovnání s původními i odvonými variacemi na LEM jiných autorů se pořád nejednalo o dobrý výsledek. I kvůli tomuto nebylo pokračováno v testování pro problémy s větší dimenzionalitou než 10.

Co se týče možných rozšíření a případných námětů na pokračévání v projektu, asi prvním logickým místem, kde pokračovat, by bylo vrátit se na začátek a znovu pečlivě navrhnout program se všemi náležitými mechanismy, bez kterých, jak se zdá nebude metoda pracovat tak dobře, na kolik by sama měla potenciál. Tedy stálo by za to implementovat kroky pro lepší kontrolu složení populace řešení během optimalizace, aby se v programu neznehodnocovala naučená hypotéza špatným výběrem vzorků. Dále implementovat metodu bez možnosti adaptivni diskretizace slibných intervalů v prostoru řešení také není vhodné, pokud chce člověk dosáhnout kvalitního optimalizačního algoritmu (i když nutno dodat, že ne všechny variace na LEM adaptivní diskretizace využily, nicméně pro numerickou optimalizaci mi teď přijde klíčová vzhledem k charakteru řešených problémů). Co se volby metody strojového učení, také myslím, že LEM ještě neekl poslední slovo a dalo by se na tomto ještě něco vystavět. Co se rozšíření a pokračování přesahujících záběr této práce týče, nejlogičtější mi přijde prozkoumat možnosti paralelizace LEM algoritmu. Přece jen pro generování co nejspolehlivějších hypotéz o kvalitních řešeních je nejlepší, pokud máme k dispozici velké množství testovacích vzorků, což se může v kombinaci s vysokou dimenzionalitou

numerického optimalizačního problému stát reálně nespočitatelné a paralelní návrh a řešení metody by mohlo být přesně to, co by pomohlo tyto problémy překlenout.

Literatura

[1] Kvasnička V., Pospíchal J., Tiňo P.: *Evolučné algoritmy,* Bratislava, STU, 2000, ISBN 80-227-1377-5

[2] Zelinka I. a kolektiv: *Evoluční výpočetní techniky – principy a aplikace,* Praha, BEN, 2008, ISBN 978-80-7300-218-3

[3] Russell, Stuart J., Norvig, Peter: *Artificial Intelligence: A Modern Approach (2nd ed.)*, Upper Saddle River, New Jersey, Prentice Hall, 2003, ISBN 0-13-790395-2

[4] Michalski, R.S.: *Learnable Evolution: Combining Symbolic and Evolutionary Learning*. Proceedings of the Fourth International Workshop on Multistrategy Learning, organized by the University of Torino, 1998, Desenzano del Garda, Italy, June 11-13, pp.14-20.

[5] Michalski, R.S.: *LEARNABLE EVOLUTION MODEL: Evolutionary Processes Guided by Machine Learning*. Machine Learning 38(1-2), 2000, pp. 9-40.

[6] Michalski. R.S. and Cervone, G.: "*Adaptive Anchoring Quantization of Continuous Variables for Learnable Evolution*. Reports of the Machine Learning and Inference Laboratory, George Mason University, 2000, Fairfax, VA

[7] Michalski. R.S., Kubat. M., Bratko I.: "A Review Of Machine Learning Methods." *Machine Learning And Data Mining: Methods And Applications*, John Wiley and sons Ltd., 1996, ISBN 978-0471971993

[8] Pošík P.: *Trends In Real Parameter Optimization*, 25. 4. 2008, Lecture presented at Brno University Of Technology, Faculty Of Information Technologies

[9] Radcliffe, N.J., Surry, P.D.: „Fundamental Limitations on Search Algorithms: Evolutionary Computing in Perspective". Computer Science Today: Recent Trends and Developments, Lecture Notes in Computer Science, Volume 1000, pp. 275-291 (ed J. van Leuwen). 1995.

[10] Wojtusiak J., Michalski R. S.: *The LEM3 Implementation of Learnable Evolution Model and Its Testing on Complex Function Optimization Problems,* Proceedings of the 8th annual conference on Genetic and Evolutionary Computation, July 08-12, 2006, Seattle, Washington, USA

[11] Michalski, R. S.: "On the Quasi-Minimal Solution of the General Covering Problem," *Proceedings of the V International Symposium on Information Processing (FCIP 69) Switching Circuits*, Vol. A3 , Yugoslavia, Bled, pp. 125-128, October 8-11, 1969.

[12] Wojtusiak, J., *AQ21 User's Guid*e, Reports of the Machine Learning and Inference Laboratory, George Mason University, MLI 04-3, Fairfax, VA, 2004.

[13] Sheri G., Corne D.: *Learning-Assisted Evolutionary Search for Scalable Function Optimization: LEM(ID3)*, 2010, IEEE Congress on Evolutionary Computation 2010

[14] Sheri G., Corne D.: *The Simplest Evolution/Learning Hybrid: LEM with KNN*, IEEE Congress on Evolutionary Computation, 2008, Hong Kong, June,1-6, 2008.

[15] L. Jourdan, D. Corne, D. Savic, G. Walters: *Hybridising rule induction and multiobjective evolutionary search for optimizing water distribution systems*, in Proc. of the 4th Hybrid Intelligent Systems conference, published in 2005 by IEEE Computer Society Press. Pp. 434-439, ISBN 0-7695-1916-4.

[16] Cervone, G., Michalski, R. S., Kaufman, K. and Panait, L. A.: *Combining Machine Learning with Evolutionary Computation Recent Results on LEM*, Proceedings of the Fifth International Workshop on Multistrategy Learning (MSL-2000), Guimaraes, Portugal, strany 41-58, June 2000.

[17] Pošík P.: *Estimation of Distribution Algorithms,* Soft Computing and Complex Systems, 2003, strany 119-122

[18] Pošík P.: *Estimation of Distribution Algorithms for Numerical Optimization*, 21. 10. 2010, Lecture presented at Charles University Of Prague, Faculty Of Electrical Engineering

[19] Miquélez T., Bengoetxea E., Larrañaga P.: B*ayesian Classifiers in Optimization: An EDA-like Approach*, Towards a New Evolutionary Computation: Studies in Fuzziness and Soft Computing, 2006, Volume 192/2006, 221-242

[20] Cendrowska J.: *PRISM: An algorithm for inducing modular rules,* International Journal of Man-Machine Studies, Volume 27, Issue 4, October 1987, Pages 349-370

[21] Vomlelová M.: „*Učení bez učitele: Klastrování, analýza nákupního košíku*“, Umělá inteligence II., Univerzita Karlova v Praze, Matematicko-fyzikální fakulta, 2005, Dostupné online na <http://kti.mff.cuni.cz/~marta/sliKlastr.pdf>

[22] Zbořil F.: *Neuronové sítě RBF a RCE. Topologicky organizované neuronové sítě, soutěživé učení, Kohonenovy mapy*, Soft Computing, Přednáška vedená na Vysokém učení technickém v Brně, Fakultě Informatiky, 2010

[23] Clark P., Niblett T.. *The CN2 Induction Algorithm*. Machine Learning, 3(4), 261-283, 1989

[24] Bengoextea, E., Miquelez, T., Larranaga, P., and Lozano, J.A., *Experimental Results in Function Optimization with EDAs in Continuous Domain*, In Pedro Larranaga, Jose A. Lozano, Estimation of Distribution Algorithms, Kluwer Academic Publishers, 2002.

Seznam příloh

1. CD s elektronickou verzí zprávy, dokumentací atd.
2. Listy s tabulkami naměřených hodnot během experimentů