**VŠB-TU Ostrava**

2014

**Evoluční algoritmy**

**Diplomová práce**

**Zdeněk Gold**

Obsah

[Předmluva 4](#_Toc408514639)

[Evoluční algoritmy – nástin 4](#_Toc408514640)

[Evoluční výpočetní techniky (EVT) 4](#_Toc408514641)

[Teorie evoluce 4](#_Toc408514642)

[Historie evolučních algoritmů 5](#_Toc408514643)

[Úvod do teorie optimalizačních algoritmů 5](#_Toc408514644)

[Klasifikace optimalizujících algoritmů 5](#_Toc408514645)

[Evoluční algoritmy 7](#_Toc408514646)

[Stochastic Hill Climbing 7](#_Toc408514647)

[Tabu Search 7](#_Toc408514648)

[Evolutionary Strategy 7](#_Toc408514649)

[Ant Colony Optimization 7](#_Toc408514650)

[Immunology Systém Method 8](#_Toc408514651)

[Memetic Algorithm 8](#_Toc408514652)

[Scatter Search 8](#_Toc408514653)

[Particle Swarm 8](#_Toc408514654)

[Vše nelze spočítat 8](#_Toc408514655)

[Fyzikální limity výpočetních technologií 9](#_Toc408514656)

[Optimalizace a účelov funkce 9](#_Toc408514657)

[Vybrané pojmy z optimalizace 9](#_Toc408514658)

[Účelová funkce 10](#_Toc408514659)

[Geometrie účelové funkce 10](#_Toc408514660)

[Tvorba účelové funkce 10](#_Toc408514661)

[Víčeúčelová optimalizace a Paretova množina 11](#_Toc408514662)

[Paretova množina 11](#_Toc408514663)

[Vybrané základní pojmy z evolučních algoritmů 12](#_Toc408514664)

[Oblasti použitelnosti evolučních algoritmů 12](#_Toc408514665)

[Společné rysy 12](#_Toc408514666)

[Populace 13](#_Toc408514667)

[Jedinci, jejich struktura a reprezentace 14](#_Toc408514668)

[Grayův kód 14](#_Toc408514669)

[Omezení a ošetření krizových stavů 14](#_Toc408514670)

[Formulace problému 15](#_Toc408514671)

[Omezení kladená na argumenty účelové funkce 15](#_Toc408514672)

[Práce s celočíselnými a diskrétními hodnotami 15](#_Toc408514673)

[Genetické programování 16](#_Toc408514674)

[Jako typickou úlohu, která by byla řešitelná touto metodou můžeme považovat například. 16](#_Toc408514675)

[Reprezentace jedince v genetickém programování 16](#_Toc408514676)

[Bibliografie 18](#_Toc408514677)

[Knihovny 18](#_Toc408514678)

[Exp4j 18](#_Toc408514679)

# Předmluva

Evoluční algoritmy vznikly jako řešení a jeden ze způsobů optimalizace v matematických a vědních kruzích. Celkem dlouho byl způsob optimalizace řešen dnes již klasickým matematickým aparátem, který je založen na infinitezimálním počtu, na variačních metodách aplikovaných ve funkcionálním prostoru nebo numerických metodách.

Tímto způsobem však lze nalézt optimální řešení pro problémy jednodušší charakter a pro ty složitější umožňuje nalézt pouze suboptimální řešení.

# Evoluční algoritmy – nástin

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

## Evoluční výpočetní techniky (EVT)

Jsou to numerické algoritmy, které vycházejí ze základních principů Darwinovy a Mendelovy teorie evoluce, jejichž hlavní ideou je předávání rodičovského genomu novým potomkům a následné uvolnění životního prostoru potomkům.

Technologie EVT stojí a padá na existenci tzv. evolučních algoritmů, které v podstatě tvoří většinu EVT. Mimo evoluční algoritmy existují ještě další rozšíření, jako jsou genetické programování, evoluční hardware apod. Vzhledem k faktu, že páteří EVT jsou genetické algoritmy, budeme si blíže specifikovat principy těchto algoritmů.

## Teorie evoluce

Podle klasické Darwinové a Mendelovy teorie evoluce, je uznáváno dogma, podle něhož se jednotlivé druhy vyvíjejí tak, že jsou z rodičů plozeni potomci, kteří podléhají při svém vzniku mutacím. Rodiče a potomci nevhodní pro aktuální životní prostředí vymírají cyklicky po tzv. generacích, čímž uvolňují místo novým rodičům a jejich potomkům.

*Schéma evolučního algoritmu*

**Postup evolučního algoritmu:**

1. Vymezení parametrů evoluce – jako je stanovení kritéria ukončení (počet cyklu, vhodnost řešení,…), stanovení účelové funkce, případně tzv. vhodnosti. Účelovou funkcí se rozumí obvykle matematický model, jehož minimalizace/maximalizace vede k řešení.
2. Generování prvopočáteční populace (obecně matice M x N, kde M je počet parametrů jedince a M je počet jedinců v populaci). Jedincem se rozumí vektor čísel s takovým počtem složek, kolik je optimalizovaných parametrů. Složky jsou nastaveny nahodile a každý jedinec představuje jedno možné řešení.
3. Všichni jedinci se ohodnotí přes definovanou účelovou funkci a každému z nich se přiřadí: a) buď přímá hodnota vrácená účelovou funkcí, nebo b) vhodnost, což je upravená hodnota účelové funkce.
4. Nastává výběr rodičů podle jejich kvality
5. Křížením rodičů se tvoří potomci. Proces křížení je u každého algoritmu odlišný.
6. Každý potomek je zmutován
7. Každý jedinec se ohodnotí stejně jako v kroku 3.
8. Vyberou se nejlepší jedinci
9. Vybraní jedinci zaplní novou populaci
10. Stará populace je zapomenuta a na její místo nastupuje populace nová. Dále se pokračuje krokem 4.

Evoluční algoritmy nejsou populární jen proto, že jsou moderní a odlišné od klasických, ale hlavně pro fakt, že v případě vhodného aplikování jsou schopny nahradit člověka.

## Historie evolučních algoritmů

Začátek se obvykle datuje do poloviny 70. let, kdy se poprvé objevily genetické algoritmy, případně do poloviny let 60., kdy byly poprvé s úspěchem použity tzv. evoluční strategie.

# Úvod do teorie optimalizačních algoritmů

Optimalizační algoritmy jsou mocným nástrojem pro řešení mnoha problémů inženýrské praxe. Obvykle se používají tam, kde je řešení daného problému analytickou cestou nevhodné či nereálné.

Většina problémů inženýrské praxe může být definována jako optimalizační úlohy, např. nalezení optimální trajektorie robota či optimální tloušťka tlakové nádoby. Řešení takových problémů obvykle vyžaduje práci s argumenty optimalizovaných funkcí.

Algoritmy této třídy mají svůj specifický název, a to „evoluční algoritmy“. Řeší problémy tak elegantně, že se staly velmi oblíbené a používané v mnoha inženýrských oborech. Z hlediska nejobecnějšího členění patří evoluční algoritmy k algoritmům heuristickým. Heuristické algoritmy můžeme rozdělit na deterministické a stochastické. Algoritmy druhé skupiny se liší v tom, že některé jejich kroky využívají náhodné operace, a to znamená, že výsledné řešení, které jimi získáme, se v jednotlivýh bězích programu mohou lišit.

Stochastické heuristické metody poskytují pouze obecný rámec a vlastní operace algoritmu je třeba zvolit (operace křížení, mutace, …). Protože tyto metody se často inspirují přírodními procesy, jsou také nazývány evoluční algoritmy. Podle jejich strategie je lze rozdělit do dvou tříd:

1. Metody založené na bodové strategii (simulované žíhání, horolezecký algoritmus, zakázané prohledávání). Základem těchto algoritmů je prohledávání sousedství aktuálního řešení, v němž se hledá řešení lepší.
2. Metody založené na strategii populace. Sem patří genetické algoritmy.

Tyto metody se liší od klasických gradientních metod tím, že připouštějí přijetí horšího řešení do další iterace, a tím se snaží vyhnout uváznutí v lokálním optimu.

Obecně lze říci, že optimalizační algoritmy slouží k nalezení minima dané účelové funkce tak, že hledají optimální numericnou kombinaci jejich argumentů.

## Klasifikace optimalizujících algoritmů

Jednotlivé třídy algoritmů představují obecně způsoby řešení daného problému metodami s různým stupněm efektivity a složitosti. Podle jejich vlastností dělíme algoritmy na tyto kategorie:

* Enumerativní

Algoritmus provede výpočet všech možných kombinací daného problému. Tento přístup je vhodný pro problémy, u nichž jsou argumenty účelové funkce diskrétního charakteru a nabývají malého množství hodnot, protože pro obecnější použití by mohl potřebovat čas, který by představoval nepřekonatelnou bariéru.

* Deterministické

Tato skupina je postavena na rigorózních metodách klasické matematiky. Algoritmy obvykle vyžadují omezující předpoklady, které těmto metodám umožňují efektivní výsledky. Např. se může jednat o:

* Problém je lineární
* Problém je konvexní
* Prohledávaný prostor možných řešení je malý
* Prohledávaný prostor možných řešení je souvislý
* Účelová funkce je, pokud možno, unimodální (jeden extrém)
* Mezi parametry účelové funkce nejsou nelineární interakce
* Jsou dostupné informace o gradientu
* Problém je definován v analytickém tvaru

Výsledkem deterministického algoritmu je pak jediné řešení

* Stochastické

Algoritmy tohoto typu jsou založny na využití náhody. Jde v podstatě o čistě náhodné hledání hodnot alrgumentů účelové funkce s tím, že výsledkem je vždy to nejlepší řešení, jež bylo nalezeno během náhodného hledání. Algoritmy tohoto typu jsou obvykle:

* Pomalé
* Vhodné jen pro malé prohledávané prostory možných řešení
* Vhodné pro hrubý odhad
* Smíšené

Algoritmy této třídy představují rafinovanou směs metod deterministických a stochastických, které ve vzájemné spolupráci dosahují překvapivě dobrých výsledků. Poměrně silnou podmnožinou těchto algoritmů jsou zmíněné evoluční algoritmy. Algoritmy smíšeného charakteru jsou:

* Róbustní (velmi často najdou kvalitní řešení)
* Efektivní a výkonné (jsou schopny nalézt kvalitní řešení během relativně malého počtu ohodnocení)
* Odlišné od čistě stochastických metod (díky přítomnosti deterministických postupů)
* Mají minimální nebo žádné požadavky na předběžné informace
* Jsou schopné pracovat s problémy typu „černá skřínka“ (netřeba analytického popisu problému)
* Jsou schopny najít více řešení během jednoho spuštění

Shrnutí je takové, že:

* Enumerativní a stochastické optimalizace nejsou vhodné na problémy, u nichž se prohledává rozsáhlý prostor možných řešení
* Deterministické optimalizace pracují dobře s problém, u nichž není prostor možných řešení příliš rozsáhlý
* Smíšená optimalizace je vhodná u problémů, u nichž nejsou kladeny žádné omezení na velikost prostoru možných řešení

# Evoluční algoritmy

## Stochastic Hill Climbing

Stochastický horolezecký algoritmus je verze tzv. horolezeckého algoritmu obohaceného o stochastickou složku. Patří mezi gradientní metody, tzn. že prohledávajíprostor možných řešení ve směru největšího spádu. Díky své gradientní povaze velmi často uvázne v lokálním extrému. Vždy se vychází z náhodného bodu v prostoru možných řešení. Pro momentálně navržené řešení se pomocí konečného souboru transformací navrhne určité okolí a daná funkce se minimalizuje jen v tomto okolí. Získané lokální řešení se pak použije jako střed pro výpočet nového okolí.

Celý proces se pak iterativně opakuje a zaznamenávají se nejlepší nalezená řešení, které po ukončení slouží jako nalezené optimum. Nevýhodou tohoto algoritmu je to, že v něm může za určitých podmínek dojít k zacyklení a řešení tak uvízne v lokálním extrému.

## Tabu Search

Je vylepšená verze horolezeckého algoritmu, u něhož vylepšení spočívá v tom, že je do horolezeckého algoritmu zavedena tzv. krátkodobá paměť, jejimž úkolem je pamatovat si ty transformace, které sloužily pro vypočítání aktuálního středu. To má v konečném důsledku ten efekt, že se algoritmus nezacyklí, díky zakázanému použití těchto transformací.

Na rozdíl od algoritmu horolezeckého, Tabu Search tak často neuvízne v lokálním extrému.

## Evolutionary Strategy

Evoluční strategie patří mezi první úspěšné stochastické algoritmy v historii. Byl navržen počátkem šedesátých let Rechenbergem a Schwefelem. Vychází z principů přirozeného výběru podobně jako genetické algoritmy. Na rozdíl od jiných stochastických algoritmů pracuje sevoluční strategie přímo s reálnými hodnotami. Jejím jádrem je práce s řešením ve formě vektoru x, které je mutováno pomocí vektoru náhodných čísel.

## Ant Colony Optimization

Optimalizace mravenčí kolonií je algoritmus, jehož činnost napodobuje chování mravenců v kolonii. Princip je následovný: Nechť existuje zdroj mravenců (mraveniště) a cíl jejich snažení (potrava). Když jsou mravenci vypuštěni, tak po nějaké době dojde k tomu, že všichni mravenci se pohybují po kratší (optimální) cestě mezi zdrojem a cílem. Tento efekt, kdy mravenci najdou optimální cestu je dán tím, že si svou cestu značkují feromonem. Pokud dorazí prní mravenec k rozcestí dvou cest, které vedou ke stejnému cíli, pak je jeh rozhodnutí, po které cestě se vydá, náhodné. Ti, kteří najdou potravu, začnou cestu značkovat a při návratu jsou díky těmto značkám při rozhodování ovlivnění ve prostěch této cesty. Při návratu ji označkují podruhé, což opět zvyšuje pravděpodobnost rozhodnutí dalších mravenců v její prospěch. Tyto principy jsou použity v ACO algoritmu.

Feromonová váha je aditivní, což umožňuje přidávat další feromony od dalších mravenců. V ACO algoritmu je zohledněn i fakt vypařování feromonů tak, že váhy jednotlivých spojů s časem slábnou. To zvyšuje robustnost algoritmu z hlediska nalezení globálních extrémů.

ACO byl úspěšně použitý na optimalizování problému Obchodního cestujícího nebo při návrhu telekomunikačnich sítí.

## Immunology Systém Method

Metoda imunitního systému je, jak už název napovídá, algoritmus, který je založen na principech fungování imunitního systému v živých organismech. Nahlíží se na něj jako na multiagentní systém, kde jednotliví agenti mají svůj specifický úkol. Tito agenti mají různé pravomoci a schopnosti komunikovat s jinými agenty. Na základě této komunikace a určité svobody v rozhodování jednotlivých agentů vzniká hirearchická struktura schopná řešit komplikované problémy. Může například jít o použítí antivirové ochrany u velkých rozsáhlých počítačových sítí.

## Memetic Algorithm

Memetický algoritmus. Tento pojem představuje širokou třídu metaheuristických algoritmů. Klíčovou charakteristikou těchto algoritmů je použití různých aproximačních algoritmů, technik lokálního vyhledávání, speciálních rekombinačních operátorů apod. V podstatě mohou být memetické algoritmy charakterizovány jako strategie soutěže a spolupráce projevující atributy synergetiky. Jako příklad memetického algoritmu lze uvést hybridní kombinaci genetikých algoritmů a simulovaného žíhání či paralelní lokální prohledávání.

S úspěchem byly použity na řešení takových problému, jako problém obchodního cestujícího, učení neuronové sítě, plánovaní údržby, nelineární celočíselné programování a další.

## Scatter Search

Rozptýlené prohledávání se svou podstatou liší od standardních evolučních algoritmů a je dost podobný algoritmu Tabu Search. Je to vektorově orientovaný algoritmus, který má za úkol generovat nové vektory na základě pomocných heuristických technik. Při startu se vychází z řešení získaných pomocí vhodné heuristiké techniky. Poté jsou generována nová řešení na základě podmnožiny nejlepších řešení ze startu. Z těchto nově nalezených řešení se opět vybere množina těch nejlepších a celý proces se opakuje. Tento algoritmus byl použit k řešení problémů, jako je řízení dopravy, učení neuronové sítě, optimalizace bez omezení, atd.

## Particle Swarm

Rojení častic je založeno na práci s populací jedinců, jejichž pozice v prostoru možných řešení je měněna pomocí tzv. rychlostního vektoru. Podle popisu nedochází v základní verzi mezi jedinci k vzájemnému ovlivňování. To je odstraněno ve verzi s tzv. sousedstvím. V rámci tohoto sousedství dochází k vzájemnému ovlivňování tak, že jedinci patřící do jednoho sousedství putují k nejhlubšímu extrému, který byl v tomto sousedství nalezen.

# Vše nelze spočítat

K demonstaci složitosti optimalizačních problémů lze uvést příklad typického reprezentanta těchto problémů a tím je SAT. Jedná se o problém z oblasti logiky, který je reprezentován složitější logickou funkcí s velkým počtem logických proměnných, který obsahuje např. 100 proměnných a cílem je nalézt takové hodnoty jednotlivých argumentů funkce tak, aby výsledná hodnota vztahu byla TRUE. Vezmeme-li v potaz, kolik je všech ohodnocení pro 100 proměnných (2^100), zjistíme, že i kdybychom vyhodnotili jednu kombinaci za 10^-13s, tak bychom stejně potřebovali 3,17x10^9 let pro všechny hodnoty.

Dalším nedostatkem SAT problému, který znemožňuje použít evolučních technik je to, že nelze u řešení vyhodnotit kvalitu. Dvě po sobě nalezené řešení tak nelze mezi sebou porovnat, zda jsi jsou sobě blízké. Pramen problému pocházi z toho, že funkce nám vrácí pouze dvě možné odpovědi: TRUE, FALSE. Nezjistíme tak, jestli je řešení lepší než jiné se stejnou hodnotou.

Dalším, teď už praktičtějsí a všednější problém, je problém obchodního cestujícího. Jde o úlohu, kdy musí obchodník navštívit N měst a vrátit se do počátečního tak, aby jeho trasa bylo co nejkratší. Problém obchodního cestujícího se dá popsat grafem. Počet všech možných kombinací je roven n! Složitost tohoto problému rychle roste s počtem měst. Už pro n>6 je v úloze více kombinací, než je v problému SAT.

## Fyzikální limity výpočetních technologií

Existují limity, které omezují výkonnost jakéhokoliv sebelepšího počítače. Tyto limity plynou z kvantově-mechanické povahy hmoty. Tyto limity omezují jak výkon počítače, tak jeho paměť.

Základním omezením v tomto směru je tzv. Bremermannova limita, která tvrdí, že v každém kilogramu hmoty nelze zpracovat více než bitů za sekundu. Tento limit se zdá dostačující, dokud nevezmeme v úvahu reálné příklady. Příkladem nám mohou být šachy, ve kterých lze nalézt kolem kombinací. Nebo třeba buněčná mřížka 100 x 100, jejiž buňky nabývající černé nebo bílé, obsahuje kombinací.

Limita je definovaná tak, že aby bylo možné informaci měřit, musí být uložena na nějakém fyzickém nosiči (elektromagnetické vlnění, papír, laserový paprsek, …). Pro měření informace je nutné, aby byly tyto informace na nosiči namodulované tak, aby byo možné rozlišit jednotlivé stavy nosiče. Tyto stavy pak reprezentují hodnotu informace. Nejmenším rozlišitelné stavy jsou kvantové stavy hmoty. Jejich rozlišitelnost je zespodu ohraničená tzv. Heinsenbergovou relací neurčitosti.

John von Neumann nazýval pozorované veličiny, které mohly rozlišovat stavy informace jako „markery“. Obecně může být „markerem“ např. přítomnost nebo absence díry děrného štítku, stav 0, 1 v paměti počítače, pixel na obrazovce s 256 odstíny šedé.

Matematicky lze „markery“ popsat jako diskrétní náhodou veličiny , která nabývá hodnot s pravděpodobností . Měřením této veličiny lze získat bitů informace. Tato veličina se nazývá Shannonova míra informace, případně entropie. Je jasné, že jako funkce n proměnných má jediný extrém a tím je hodnota . V tomto případě může „marker“ reprezentovat maximálně bitů.

# Optimalizace a účelové funkce

## Vybrané pojmy z optimalizace

Úlohy vedoucí k výpočtu extrémů funkce jsou obvykle úlohy prakticé činnosti člověka a vyžadují analytické a mnohem častěji numerické výpočty extrémů funkcí více proměnných. Existují však komplikace, které ztížují optimalizaci někerých problémů, mající tyto zdroje:

* Prostor možných řešení je příliš velký. To může být například u numerativního přístupu (procházení všech možností) dosti velkým problémem
* Samotný problém je natolik složitý, že jeho matematický model použitý při optimalizaci vrací výsledky, které sice odpovídají příslušnému modelu, ale nesouhlasí se skutečně řešeným problémem. Model je vůči realitě nepřesný.
* Účelová funkce použitá k měření kvality aktuálního nalezeného řešení může podléhat šumu nebo se může měnit v čase. Proto nestačí pouze jedno řešení, ale množina řešení reprezentující vývoj toho nejlepšího.
* Množina řešení podléhá striktnímu omezení, že nalezení optimálního řešení je extrémně složitý problém.

Historický původ optimalizačních úloh sahá do antiky. Z té doby známe úlohy, jako je Didonina úloha, která měla od krále přislíbenou zemi, kterou by ohraničila volskou kůží. Didonina kůži nařezala na tenké pásky a ohraničila území, které se stalo základem Kartága.

Impulz pro rozvoj optimalizačních metod po druhé světové válce přinesly úlohy z oblasti ekonomie, které lze shrnout pod název problému optimalizace výrobních programů. Typicky se v nich předpokládá, že podnik má technické možnosti pro výrobu n druhů výrobků a hledá výrobní program x. Matematicky toto můžeme zapsat jako , kde udává počet výrobků tohoto druhu. Takový výrobní program přináší podniku užitek . O této funkci mluvíme jako o funkci účelové. Podnik však musí pro konkrétní program respektovat řadu omezení (výrobní zdroje) a rovněž omezení odbytu výrobků.

Jinou skupinou optimalizačních úloh pak může být problém úloh z dopravního hospodáství. V těchto problémech jde o to, aby byl navržen plán přepravy zboží s míst výroby do míst spotřeby. Respektují se omezení na množství přepravovaného zboží, požadavyk spotřebitelů a příp. další. Výsledkem by pak měl být plán na omezení nákladů na přepravu a nebo čas přepravy.

*Ještě vypsat nějaké příklady*

## Účelová funkce

Výrazem „účelová funkce“ budeme rozumět funkci, jejiž optimalizace povede k nalezení optimálních hodnot jejich argumentů. Označuje se jako nebo také . Funkce má v bodě lokální maximum, jestliže existuje v okolí bodu takové, že platí pro všechna z tohoto okolí. Funkce má v bodě ostré lokální maximum, jestliže existuje okolí bodu takové, že pro všechna z tohoto okolí, vyjma .

## Geometrie účelové funkce

Na každou účelovou funkci lze nahlížet jako na geometrický problém, v jehož rámci se hledá nejnižší či nejvyšší pozice na ploše ležící v (N+1) – rozměrném prostoru, pro kterou se někdy používá výraz „hyperplocha“ či „prostor možných řešení“ daného problému. Počet dimenzí N je dán počtem optimalizačních argumentů účelové funkce. Má-li optimalizovaná funkce např. šest argumentů, pak se hledá extrém na šestirozměrné ploše v sedmirozměrném prostoru, kde sedmá dimenze je návratová hodnota účelové funkce.

Často je situace mnohem složitější. Daná účelová funkce může obsahovat více stejných globálních extrémů o stejné hodnotě na různých souřadnicích, nebo ještě hůře, je těchto extrémů nekonečně mnoho.

## Tvorba účelové funkce

Definice účelové funkce je jedním z nejkritičtějších krků v rámci optimalizačního procesu. Jeho správné provedení může citlivě ovlivnit kvalitu výsledků. Třídá problémů pro definici optimalizační funkce je nepřeberná, neexistuje kuchařka konkrétních postupů, jak účelovou funkci sestavit. Dají se pouze naznačit obecné principy. Při tvorbě je nutné vědět, čeho se má dosáhnout a z čeho lze vycházet.

Jako příklad může posloužit prediktivní řízení, které v základní myšlence hledá množiny akčních zásahů, které vedou výstup k žádané hodnotě.

Jiným příkladem může být odhad parametrů tzv. ARMA modelu pro predikci časových řas ekonomického, technického a jiného charakteru. V takovém případě je účelová funkce postavena na základě modelu s N parametry. Při optimálně nastavených parametrech modelu by pro každý vstup dat měl být rozdíl na výstupu minimální.

Další příkladem jak sestrojit účelovou funkci je tzv. inverzní fraktální problém (IFP).

# Víčeúčelová optimalizace a Paretova množina

## Paretova množina

Víceúčelová optimalizace (MOOP) je založena na optimalizaci dvou a více funkcí, které mají být minimalizovány, příp. maximalizovány. V obecně formě definujeme MOOP takto:

Proměnná je vektor . Funkce a jsou omezující funkce a poslední podmínka reprezentuje omezení kladená na argumenty účelové funkce. Řešení, které nesplňuje podmínky je nepřípustné řešení. Množina řešení, které podmínky splňuje se nazývá přípustná řešení. Díky restrikcím omezujících podmínek, je prostor možných řešení obecně nesouvislý a je tvořen izolovanými množinami přípustných řešení.

V případě jednoúčelové optimalizace je řešení reprezentováno jako bod v prostoru přípustných řešení, kde x a y reprezentují možné hodnoty argumentů účelové funkce a z hodnotu účelové funkce. U víceúčelové optimalizace je taková grafická reprezentace nemožná, a mimo jiné i proto se zavádí dva typy grafů, které umožňují přehledně vizualizovat připustné řešení, ale i hodnoty více funkcí najednou. První graf je označován jako „decision space“, které budeme pro potřeby této publikace raději označovat jako prostor přípustných kombinací (PPK) a druhý (objective space) jako prostor hodnot funkcí (PHF). V každém bodu v PPK je přiřazen jeden bod v PHF. Jde tedy o zobrazení z n rozměrného prostoru PPK do m rozměrného prostoru PHF.

Paretovou hranici lze definovat jako množinu bodů, které reprezentují takové kombinace, že nelze snížit žádnou hodnotu účelové funkce, aniž by se nezvýšila hodnota nkterých jiných funkcí.

Úlohy víceúčelové optimalizace se často převádějí na optimalizační s jednou účelovou funkcí. Výsledkem je pak PHF. Nejčastěji používané rozšířené úpravy jsou: vážení a následná sumace více účelových funkcí, nebo zahrnutí funkcí do omezujících podmínek. První metoda spočívá v tom, že se jednotlivé funkce vynásobí zvolenými váhami a sečtou se. Tak můžeme dosáhnout toho, že se víceúčelový problém převede na optimalizaci jediné funkce. Nevýhodou těchto úprav je, že váhy jsou většinou vybírány na základě úsudku, který může poškodit vítězství. Druhá úprava je založena na výběru jedné účelové funkce a přesunutí ostatních do omezujících podmínek. I zde obvykle dochází ke zkreslení.

Pro potřeby víceúčelových optimalizací bývají nadefinovány čtyři typy vektorů: Ideální vektor – IOV, utopický vektor UOV, nadir vektor NOV, nejhorší vektor WOV. Při optimalizacích je IOV poměrně důležitý objekt, protože reprezentuje tzv. ideální řešení, kterého nelze dosáhnout. Je to bod v PHF, jehož složky jsou nejlepším jednotlivých účelových funkcí. Vezmeme-li složky každé účelové funkce optimalizované samostatně, pak získáme složky IOV. Neméně důležitý je UOV, který je používán některými algoritmy jako nositel informace o striktně nejlepších hodnotách. NOV je v podstatě inverze IOV. Zatímco IOV ohraničuje Paretovou množinu zdola, NOV ji ohraničuje shora.

# Vybrané základní pojmy z evolučních algoritmů

## Oblasti použitelnosti evolučních algoritmů

K dnešnímu datu existuje velké množství algoritmů, jež spadají do třídy tzv. evolučních algoritmů nebo se do této třídy algoritmů dají za určitých podmínek zařadit. Příkladem mohou být ACO, Imunitní systém (ISM) Scatter search či rojení částic.

Nestačí mít pouze dobrý algoritmus, ale je mnohdy životně důležité vědět, s jakou třídou problému je vůbec daný algoritmus schopen pracovat. Znamená to tedy stanovit oblast použitelnosti daného algoritmu.

Na každý optimalizační problém lze nahlížet jako na problém geometrický, u něhož je cílem je najít nejnižší či nejvyšší bod na N rozměrné ploše.

Pro zmíněné testovací funkce platí.

1. Graf funkce nemá fraktální charakter
2. Jsou definované na reálných, celočíselných nebo diskrétních argumentech
3. Jsou multimodální (jeden či více extrémů)
4. Mají různá omezení (kladené na argumenty či hodnoty účelové funkce)
5. Jsou silně nelineární
6. Představují problémy typu jedhla v kupce sena
7. Nalezení jejich globálních extrému evolučními algoritmy je NP složité

Navíc také může platit, že:

1. Funkce je separabilní, což znamená, že se nadá rozložit na několik jednodušších funkcí, jež lze optimalizovat nezávisle
2. Počet promnných je vysoký
3. Prostor možných řešení může být rozsáhlý i nesovislý

## Společné rysy

Evoluční algoritmy mají několik společných rysů:

1. Jednoduchost, protože tyto algoritmy lze naprogramovat obvykle velmi jednoduše.
2. Hybridnost čísel, se kterými algoritmus pracuje. Bez jakýchkoliv problémů lze kombinovat čísla typu integer, real, případně jen vybrané množiny čísel.
3. Používání dekadických čísel. Jedinec se nemusí převádět do binárního kódu, který je běžně používán u genetických algoritmů. U binárního zápisu mohou totiž mutace způsobit skokovou změnu čisla, což nemusí být dobrý dopad na průběh evoluce. Tyto nerovnoměrnosti se sice dají odstranit použitím Grayova kódování, nicméně práce s reálnými čísly je stále výhodnější.
4. Rychlost. Díky své relativní jednoduchosti, zvláště při porovnání s klasickými metodami, lze říci, že požadované řešení naleznou mnohem rychleji.
5. Chopnost nalézt extrém i u funkcí, které jsou v grafickém slova smyslu ploché a extrém je jen dírou v této rovině. Lze to označit jako hledání jehly v kupce sena. Ovšem u těchto problémů je účinnost jakýchkoliv algoritmů, včetně evolučních, velmi nízká.
6. Schopnost dát vícenásobné řešení.Výsledkem evoluce je nejlepší jedinec – jedno řešení. Pokud je ovšem v globálních extrémů více, pak lze očekávat, že budou rovněž evolučním procesem nalazeny.

Jinými slovy, evoluční algoritmy jsou vhodné pro hledání extrémů funkcí trpících takovými patologiemi, jako např. šum, vsoký počet dimenzí, „multimodalita“

## Populace

Typickým rysem evolučních algoritmů je, že jsou založeny na práci s populací jedinců. Populace může být znázorněna jako matice NxM, kde sloupce představují jednotlivé jedince. Každý jedinec představuje aktuální řešení danéího problému. S každým jedincem je navíc spojena hodnota účelové funkce, která říká, jak vhodný je jedinec pro další vývoj populace.

Hlavní činnosti evolučních algoritmů je cyklické vytváření nových populací, tedy náhrada starách populací novými. To vše pomocí přesně definovaných matematických pravidel.

K vytvoření populace je třeba nadefinovat tzv. vzor, podle kterého se generuje celá počáteční populace. Ve vzoru jsou pro každý parametr konkrétního jedince definovány tři konstanty, a to typ proměnné a hranice intervalu, v němž může pohybovat hodnota parametru. Volba hranice je velmi důležitý krok, protože při jejich nevhodném zvolení se může stát, že budou nalezena řešení , která nebudou možné fyzikálně realizovat nebo nebudou mít opodstatnění.

Další neméně důležtý význam hranic souvisí se samotným evolučním procesem. Může se stát, že daný problém bude reprezentován plochou, která bude nabývat lokálních extrémů stále větších hodnot se vzrůstající vzdáleností od počátku. To způsobí, že evoluce bude nacházet stále nová řešení až do nekonečna. Je to způsobeno tím, že evoluční proces směřuje do stále hlubších a vzdálenějších extrémů.

Populace je na základě vzorového jedince vygenerována podle vzorce.

Tento vztah zajišťuje, že všchny parametry jedinců budou náhodně vygenerovány uvnitř povolených hranic prostoru možných řešení.

Zobrazení o tom, jak kvalitně proběhla evoluce, se provádí pomocí tzv. historie vývoje hodnoty účelové funkce ve formě jednoduchého grafu. Na něm je vykreslena závislost vývoje účelové funkce na aktuálním počtu jejich ohodnocení. Jde o sekvenci nejhorších a nejlepších řešení z jednotlivých populací. Výhodnější je však zobrazení závislosti hodnoty účelové funkce na aktuálním počtu jejich ohodnocení. To proto, že u evolučních cyklů se provádí u jednotlivých algoritmů různý počet ohodnocení účelové funkce. U prvního případu může být pomalejší konvergence hodnoty účelové funkce zobrazena jako rychlejší a naopak. Skutečná informace o kvalitě evoluce je pak zkreslená. U druhého způsobu můžeme objektivně porovnávat různé typy algoritmů bez ohledu na jejich vnitřní struktuře.

Kromě vývoje nejlepšího jedince je pak vhodné zobrazovat vývoj i nejhoršího jedince z populace, a to do jednoho grafu.

Pokud má nejlepší a nejhorší jedince stejnou hodnotu účelové funkce, vysvětlením může být:

1. Populace je rozprostřena ve více extrémech o stejné hodnotě účelové funkce
2. Celá populace je v jednom extrému, což je pravděpodobnější, neboť většina problémů je reprezentována funkcí s jedním globálním extrémem.

V případě číslo 1 je šance, že evoluce poběží dále, zatímco v příadě číslo 2 je jakákoliv další evoluce zbytečná.

Vývoj populace **musí** být vždy konvergentní k lepším hodnotám, což znamená, že nemůže nikdy vykazovat divergenci. V daném algoritmu funguje tzv. „elitismus“, který slouží jako jakýsi jednosměrný filtr, jenž **propouští do nové populace pouze ta řešení, která jsou lepší či stejně dobrá jako ta ze staré populace**.

## Jedinci, jejich struktura a reprezentace

Historicky nejstarší je binární reprezentace. Jedinec je tvořen sekvencí 0 a 1, které se říká chromozom. Tato reprezentace má své historické kořeny a používá se dodnes zejména u genetických algoritmů. Má však svoje nevýhody. Zejména se jedná o skokovou změnu struktury chromozomů, při spojité změně odpovídající reálné hodnoty. Aby se předešlo tomuto chování, používá se Grayův kód. Jde opět o binární kód, je však zprostěn výše uvedeným skokovým změnám.

Další možností, jak lze jedince v algoritmu reprezentovat, je pomocí reálných nebo celých čísel (případně jejich kombinací). Speciální repezentací je jedinec, který obsahuje i nenumerické hodnoty.

Poslední speciální formou repezentace je tzv. strom. Ten umožňuje vizualizovat jedince jako stromovou strukturu.

## Grayův kód

Pro genetické algoritmy se často využívá Grayovo kódování, při mutaci binárního jedince tak nedochází k velké zěně reálného čísla, které odpovídá příslušné binární sekvenci.

Jedna z variant konstrukce Grayova kódu je pomocí tzv metody „zrcadlení“. Spočívá v tom, že se vezme n bitový kód, ten se rozšíří o svou vlastní zrcadlovou kopii. K původní části se pak přidá 0, zatímo k zrcadlové kopii 1. To vše se opakuje tak dlouho, dokud nemáme požadovaný m bitový řetězec.

Výhoda Grayova kódování se projevuje v rovnoměrnější mutaci jedinců a obecně rychlejší konvergenci algoritmu ke globálnímu optimu. Někteří spíše tvrdí, že Grayův kód genetický algoritmus zpomaluje kvůli procesu konverze, druzí jsou naopak zastánci použití Grayova kódu.

# Omezení a ošetření krizových stavů

Při práci s evolučními algoritmy je prakticky na denním pořádku, že jedinci mají tendenci se nejen shlukovat okolo možných řešení, ale také nastává situace, při nichž opouštějí tyto oblasti a mohou nalézat řešení, která jsou nesmyslná či neopodstatněná. K zamezení takových situací existují metody, které jsou součástí evolučního algoritmu a které mají roli pastevce udržujícího své stádo ve vymezeném prostoru možných řešení.

## Formulace problému

Evoluční algoritmy mohou být použity k řešení optimalizačních problémů s celočíselnými, diskrétními či reálnými argumenty účelové funkce. Tyto problémy lze formulovat následovně:

Nalézt takové , které minimalizuje funkci vzhledem k funkčním omezením a k omezením argumentů

## Omezení kladená na argumenty účelové funkce

Při optimalizaci je obvykle nutné zajistit, aby hodnoty jednotlivých argumentů účelové funkce ležely uvnitř povolených hranic.

Každý jedinec, který překročí povolenou hranici je nahrazen nově náhodně vygenerovaným jedincem, který leží v povoleném intervalu. To má mimo jiné také vliv na zvýšení diverzibility populace, protože v okamžiku náhrady jedince v geometrickém slova smyslu přesune na novou pozici, která není výsledkem křížení jedinců.

Druhý způsob – zastavení jedince na kranici, který nám nabízí prostá logika, se ukazal být nevyhovující. Jedinci se za určitých podmínek mohou shluknout na hranici. Evoluční procesy mají s takovým ošetřením hranic nižší diverzibilitu, což vede v konečném výsledku k delší době potřebné k nalezení globálního extrému. Případně nemusí být extrém vůbec nalezen.

Penalizací funkce se rozumí záměrná úprava hodnot účelové funkce ve vybraných oblastech argumentů. Pokud jsou tyto oblasti jednoznačně zakázány, pak se penalizace označuje jako hard-constraints. V tomto případě se hodnota účelové funkce nemodifikuje a jedinec, který by se nacházel v zakázáne oblasti, je zrušen a náhodně nahrazen jedincem ležícím v oblasti povolené. Druhou možností je tzv. soft-constraints. V tomto případě jedinec, který leží v zakázané oblasti, není zrušen, ale je nezvýhodněn modifikací hodnoty účelové funkce.

Tato omezení mají svůj původ v reálném světe, kde některé hodnoty účelové funkce mohou představovat řešení fyzikalně nerealizovatelné nebo nevýhodné. Penalizace má pak efekt ten, že je vybraným jedincům, kteří se nacházejí v zakázené oblasti, znepříjemněn pobyt. Z geometrického hlediska si lze toto znepříjemnění představit jako lokální deformaci hyperplochy směrem k opačným extrémům. Výsledný efekt je pak takový, že jedinci, kteří se díky evolučnímu procesu dostanou do takových oblastí, je buď rychle opustí, nebo nepostoupí do další populace.

//Dodělat

Velmi výraznou výhodou penalizačního přístupu soft-constraints je, že prostor možných řešení zůstává souvislý.

Za zmínku ještě stojí tzv. adaptivní penalizace. Její mírná modifikace je zde zastoupená vztahem. Suma v tomto vztahu reprezentuje penalizační faktor. Jeho velikost je ovlivněná dvěma heuristicky stanovenými koństantami.

## Práce s celočíselnými a diskrétními hodnotami

Evoluční algoritmy jsou schopny pracovat nejen s reálnými, ale také s celočíselnými a diskrétními hodnotami argumentů. Ošetření celočíselných hodnot může být dvojího druhu. V prvním případě se argument účelové funkce jednoduše zaokrouhlí přímo v populaci a ve druhém případě se zaokrouhlení v populaci neprovede, ale provede se až před dosazením do účelové funkce. Tento nepatrný rozdíl má zásadní význam. Při zaokrouhlení v populaci jdou do další generace jedinci, jejichž pohyb je omezen na diskrétní množinu pozic na prohledávané hyperploše. Při zaokrouhlení ořed dosazením do funkce zůstávají v populaci jedinci s neceločíselnými hodnotami, což má za následek dopad na diverzibilitu a robustnost.

# Genetické programování

V předchozích odstavcích jsme se postupně dostali z genetických algoritmů, řešenými binárními chromozomy, až k hybridním genetickým algoritmům, které byly navrženy pro konkrétní účely řešení problémů. Nyní se dostáváme ke kapitole zabývající se problematikou generování počítačových programů za pomocí evoluce (konkrétně genetickým programováním).

Genetické programování lze považovat za upravenou verzi genetických algoritmů, kde namísto číselných polí s pevnou délkou používáme stromovou strukturu s objekty (funkce, podprogramy, struktury,… atd.)

Za otcem genetického programování stojí John Koza, který v roce 1998 navrhl modifikaci klasického genetického algoritmu, ale už ne pro práci s binárními hodnotami (resp. obecně číselnými), ale pro tvorbu tzv. programů s použitím symbolické regrese.

Za příklady úloh, které by se daly řešit metodou genetického programování uveďme například hledání programu složeného z několika instrukcemi, které by zajišťovaly bezpečný průchod robota prostředím s překážkami.

Populace tedy neobsahovala čísla, ale struktury programů, symboly, funkce. Kdyby se ale tyto struktury musely projít procesem křížení a mutací, byly by výsledné zdrojové kódy nesmyslné a neproveditelné proto se kód reprezentuje S-výrazem, který lze aplikovat na jakýkoliv programovací jazyk.

GP používá pro reprezentaci jedince nikoliv pole hodnot, ale stromovou strukturu.

obrázek stromové struktury

Implementace GP a stromové struktury.

# Jako typickou úlohu, která by byla řešitelná touto metodou můžeme považovat například.

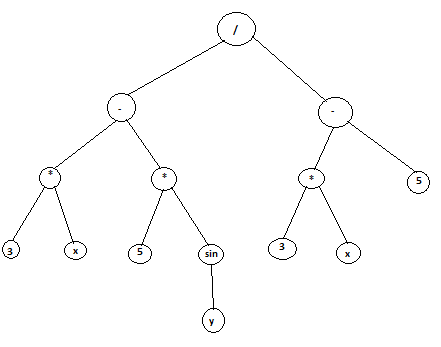
Doplnit příklad řešení genetickým programováním

## Reprezentace jedince v genetickém programování

Víme-li, že genetické programování pracuje s hierarchicky uspořádanou strukturou programů, funkcí, příkazů, musíme být schopní reprezentovat jakoukoliv variantu takto vytvořených programů nějakou univerzální datovou strukturou. Tímto zásadním problémem se zabýval také Koza a dospěl k řešení, které mu poskytl jazyk LISP. Programy zde psané odpovídají zápisem S-výrazům[[1]](#footnote-1). S-výraz je v podstatě syntaktický strom příslušného programu.

Syntaktický strom popisuje strukturu programů za pomocí neterminálů (funkce, procedury, příkazy,…) a terminálů (konstanty, proměnné). Pro konkrétní problémy se potom definuje množina funkcí a množinou terminálů .

Příkladem takového syntaktického stromu nějakého smyšleného programu může být toto schéma, reprezentované jako výraz níže popsaný:



Obrázek Syntaktický strom pro daný výraz

Dalšími důležitými požadavky pro množiny terminálů a neterminálů je to, aby splnili podmínky uzavřenosti[[2]](#footnote-2) a postačitelnosti[[3]](#footnote-3). Zabraňuje se tak chybám při generování takovýchto struktur. Máme-li například množinu terminálů obsahující goniometrické funkce a množinu terminálů, obsahující reálná a celá čísla, budou obě podmínky splněny. Pokud však bude množina neterminálů obsahovat navíc funkci dělení, podmínky splněny nebudou, protože dělení nulou není definovatelné a bude se tak muset funkce opravit tak, aby tento chybný stav korektně opravila. Možným řešení těchto problémů a jejich odpovědí můžou být gramatiky[[4]](#footnote-4).

Příklad

Požadavek postačitelnosti.je hůře popsatelný.

# Implementace

## Struktura aplikace

Class diagram

## Start optimalizačního programu (algoritmu)

Jelikož je vyvíjený framework[[5]](#footnote-5) pro využití evolučních optimalizačních algoritmů bude nejčastěji nasazován spolu s nějakým GUI pro snazší nastavení parametrů algoritmu, je už proto model běhu navrhnut tak, aby podporoval oddělený běh aplikace v nezávislých vláknech. Zkrátka a dobře, každá třída nějakého optimalizačního algoritmu obsahuje metodu start(), která je tzv. neblokující a vykonává svoji práci ve vlastním vlákně. Výhodou takovéhoto návrhu je potom to, že se programátor nemusí starat o paralelní běh GUI a výpočetní metody.

I když výpočet běží v jiném vlákně, je možné reagovat na změny ve vnitřním stavu objektu algoritmu, které se projeví u několika komponent GUI a ty si postupně načtou nová data.Uplatňuje se zde návrhový vzor Observer (Pozorovatel).

# Testování

## Rozdíl mezi kódování chromozomu

Udělat test na rozdílné kódování

# Bibliografie

1. **Wikipedie.** Formální gramatika. *Wikipedie, otevřená encyklopedie.* [Online] 28. listopad 2014. http://cs.wikipedia.org/wiki/Form%C3%A1ln%C3%AD\_gramatika.

2. **Poli, Riccard a Langdon, William B.** *A Field Guide to Genetic Programming.* 2008.

3. **Zelinka, Ivan.** *Evoluční výpočetní techniky, principy a aplikace.* místo neznámé : BEN.

4. **Hynek, Josef.** *Genetické algoritmy a genetické programování.* Praha : Grada Publishing, a.s., 2008. ISBN 978-80-247-2695-3.

5. **Wikipedie.** Framework. *Wikipedie - Otevřená encyklopedie.* [Online] 26. srpen 2014. http://cs.wikipedia.org/wiki/Framework.

# Knihovny

## Exp4j

<http://www.objecthunter.net/exp4j/apidocs/overview-summary.html>

Knihovna pro výpočet výrazu zadaného v textové podobě.

1. Jde o zápis vnořeným seznamem (např. stromovou strukturou) a lze jim zapisovat program i data [↑](#footnote-ref-1)
2. Funkce je schopná přijmout jakýkoliv argument z množiny terminálů a výsledek jakékoliv funkce z množiny neterminálů [↑](#footnote-ref-2)
3. [↑](#footnote-ref-3)
4. Struktura, která popisuje formální jazyk. Skládá se z pravidel, pomocí nichž může být každé slovo vygenerováno z předem daného počtu symbolů. [↑](#footnote-ref-4)
5. Softwarová struktura jako opora při programování [↑](#footnote-ref-5)