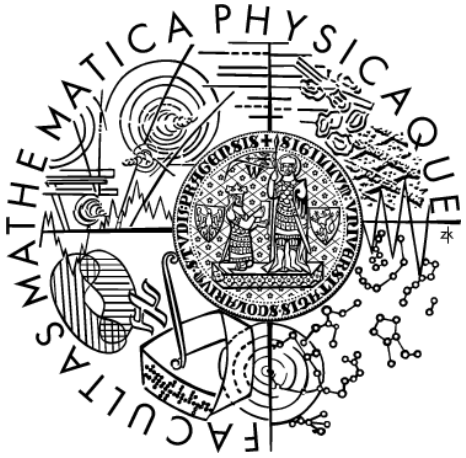
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Martin Štefan

Studium a srovnávání hlavních typů evolučních algoritmů

Ústav Informatiky Akademie Věd ČR

Vedoucí diplomové práce: doc. RNDr. Ing. Martin Holeňa, CSc.
Studijní program: Informatika
Studijní obor: Teoretická informatika

Praha 2011

Na tomto místě bych rád poděkoval svému vedoucímu diplomové práce doc. RNDr. Ing. Martinu Holeňovi, CSc. za vedení a podnětné impulzy při psaní této práce. Zároveň bych poděkoval rodičům a manželce, kteří mě po celou dobu podporovali a vytvořili vynikající zázemí.

Prohlašuji, že jsem tuto diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 5. 12. 2011. Podpis autora

Název práce: Studium a srovnávání hlavních typů evolučních algoritmů

Autor: Martin Štefan

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: doc. RNDr. Ing. Martin Holeňa, CSc., Ústav Informatiky Akademie Věd ČR

Abstrakt: Evoluční algoritmy patří mezi nejmladší a zároveň nejprogresivnější metody řešení obtížných optimalizačních úloh. Tyto algoritmy si získali svou velkou oblibu díky dobrým experimentálním výsledkům na složitých optimalizačních úlohách, jednoduchosti implementace a také vysoké modularitě, tedy možnosti úpravy pro účely řešení různých problémů. Mezi nejpoužívanější evoluční algoritmy patří Genetické algoritmy, Diferenciální evoluce a Evoluční strategie. Tyto algoritmy a jejich varianty je možné aplikovat jak na úlohy spojité, diskrétní tak i smíšené optimalizace. Předmětem této práce je srovnání chování tří základních typů evolučních algoritmů na katlystické optimalizační úloze se smíšenými proměnými, lineárním omezením a experimentálně počítanou fitness funkcí.

Klíčová slova: evoluční algoritmy, smíšená optimalizace, lineární omezení, genetický algoritmus, diferenciální evoluce, evoluční strategie s adaptací kovarianční matice a dynamickým shlukováním.

Title: Study and Comparison of the Main Types of the Evolutionary Algorithms

Author: Martin Štefan

Department: Department of Software Engineering

Supervisor: doc. RNDr. Ing. Martin Holeňa, CSc.

Abstract: Evolutionary algorithms belongs among the youngest and the most progressive methods of solving difficult optimization tasks. They received huge popularity mainly due to good experimental results in optimization, a simplicity of the implementation and a high modularity, which is an ability to be modified for different problems. Among the most frequently used Evolutionary algorithms belongs Genetic Algorithm, Differential Evolution and Evolutionary Strategy. It is able to apply these algorithms and theirs variants to both continuous, discrete and mixed optimization tasks. A subject of this theses is to compare three main types of algorithms on the catalyst optimization task with mixed variables, linear constraints and experimentally evaluated fitness function.

Keywords: evolutionary algorithms, mixed optimization, linear constraints, genetic algorithm, differential evolution, evolutionary strategy with covariance matrix adaptation and dynamic niching

Obsah

1. Úvod

1.1 Cíle práce

Cílem této práce je porovnání tří základných typů evolučních algoritmů, kterými jsou Genetické algoritmy, Diferenciální evoluce a Evoluční strategie s adaptací kovarianční matice a dynamickým vytvářením nik. Srovnávané algoritmy by měly být schopny řešit smíšenou optimalizaci [1] a pracovat s lineárním omezením. Prostřekem pro porovnání zmíněných algoritmů nám posloužila reálná optimalizační úloha tzv. *katalytická optimalizace*.

1.2 Katalytická optimalizace

Rozhodovací (prohledávací) prostor této úlohy je poměrně rozsáhlý a často vystoupá až na dimenzi 20-50, což společně s experimentálně počítanou objektivní funkcí dělá z této úlohy obtížný problém smíšené optimalizace. Optimalizační funkce[2] vyjadřuje v tomto případě jistou mírou efektivity katalýzy vzhledem k vstupním prvkům a ostatním rozhodovacím parametrům. Příkladem takové míry efektivity reakce může být například rychlost konvergence reakce a mnoho dalších charakteristik. Kvůli experimentální povaze této funkce a nedostatečné teoretické znalosti o ní, není možné pro nalezení optima využít standardní efektivní matematické metody využívající znalosti první, nebo druhé derivace. Existují tedy dva možné přístupy jak danou úlohu řešit. Za prvé se jedná o metody nevyužívající znalosti derivací optimalizační funkce. Do této kategorie spadají například simplexová metoda, genetické algoritmy nebo simulované žíhání. Druhou možností řešení je místo stávající optimalizační funkce využít od ní odvozenou analyticky vyjádřitelnou aproximaci, u které již je možné využít některou standardní gradientní metodu. Pro nalezení takových aproximačních funkcí se osvědčují neuronové sítě zvláště pak vícevrstvé perceptronové sítě [1].

1.3 Požadavky na katalytickou optimalizační úlohu

Katalytická optimalizační úloha patří do kategorie smíšené optimalizace, tedy obsahuje jak diskrétní, tak i spojité proměnné. Součástí jejího popisu jsou komponenty vstupující do katalýzy, dále pak jejich množství, počet a další vlastnosti. Aby byla úloha validní, musí veškeré tyto složky splňovat určité podmínky.[2]

1. *Stromová struktura komponent* - Všechny vstupní komponenty jsou uspořádány v hierarchické stromové struktuře, která popisuje víceúrovňové dělení komponent na podkomponenty dle svého významu v reakci. Některé komponenty nabývají určitou hodnotu vybranou z konečné množiny buď

[1] Tedy optimalizaci jak ve spojitých, tak i diskrétních proměnných

[2] Ekvivalentní název je cílová, nebo také fitness funkce

celočíselných hodnot, nebo dalších komponent. Součástí popisu stromové struktury jsou i určité vlastnosti komponent. Příkladem vlastnosti, kterou mohou mít komponenty je tzv. *proporce* (Proportion). Proporce může být buď molární, nebo hmotnostní a může se týkat buď samotných komponent, nebo celých skupin komponent. U *proporce* může být specifikována také pravděpodobnostní distribuce přípustných hodnot. V případě, že je *proporce* reprezentována spojitou komponentou, pak příklady takových rozdělení jsou: *rovnoměrné*, *lognormální* a *exponenciální rozdělení*. *Přesnost* (Precision) je vlastnost komponenty, která určuje na kolik desetinných míst se proporce zvolené komponenty bude ukládat do databáze. Zbývající desetinná místa jsou zanedbána.

2. *Lineární omezení* - vazby mezi počty komponent případně poměry mezi komponentami popisují lineární rovnosti a nerovnosti. Zachycují například poměry mezi proporcemi komponent, zdola i zhora omezují počty komponent, podkomponent, nebo celých skupin komponent. Linearní omezení a hranice také zpravidla oboustranně omezují *proporce* komponent.

3. *Diverzita populace* - udržování odlišných jedinců v populaci Evolučních algoritmů je nezbytný předpoklad ze dvou důvodů. Zaprvé je fitness funkce v reálné aplikaci počítána experimentálně, a tedy je velice časově i finančně náročné opakovaně vyhodnocovat stejné nastavení chemické reakce. Druhý důvod je podstatný pro samotnou studii. Diverzita populace totiž významně ovlivňuje rychlost konvergence algoritmů.

1.4 Struktura práce

V následující kapitole budou připomenuty jednotlivé algoritmy, nebo lépe řečeno třídy algoritmů, které byly v této práci použity. U algoritmů bude uveden základní popis, jejich vlastnosti, možnosti a výhody oproti ostatním algoritmům. Pro podrobnější studii ovšem doporučuji nahlédnout do odkazovaných materiálů. Třetí kapitola bude již věnována vlastním algoritmům pro řešení naší reálné optimalizační úlohy. Budu se zde věnovat modifikacím algoritmů a také metodám zachování přípustnosti řešení vůči lineárním omezením. V závěru kapitoly bude popsán hlavní algoritmus, který vytváří jakýsi jednotný rámec pro porovnání zbývajících algoritmů. Čtvrtá kapitola bude popisovat vlastní implementaci celého algoritmu. V páté kapitole bude na dvou optimalizačních strukturách, provedeno několik testů porovnání algoritmů s různým nastavením kontrolních parametrů. V závěrečné kapitole budou zhodnoceny výsledky experimentů a shrnutí naplnění cílů ale také nedostatky a možnosti dalšího vývoje uvedených metod.

2. Použité evoluční algoritmy

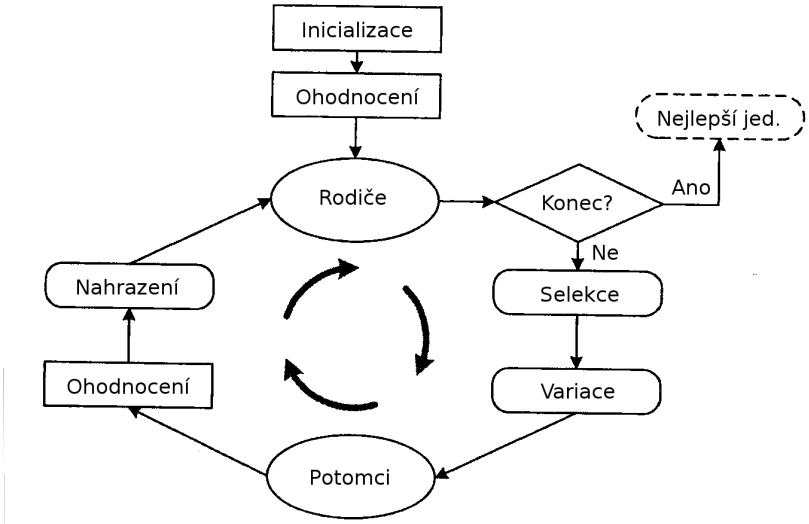
2.1 Evoluční algoritmy obecně

Evoluční algoritmy (EA) jsou třída algoritmů sloužících pro řešení náročných optimalizačních úloh [4]. Jejich název není náhodný, existuje zde totiž jistá paralela s *Darwinovou evoluční teorií*, neboli vývojem druhů v přírodě. Dle této teorie se organizmy v přírodě z generace na generaci vyvíjí a přizpůsobují se tak svému okolí, aby obstáli v konkurenci ostatních druhů. V řeči matematické optimalizace lze říci, aby maximalizovali šanci na přežití. Stejného principu jako evoluce druhů využívají i *evoluční algoritmy* pro řešení různých optmalizačních úloh.

Pro popis *evolučních algoritmů* je vhodné zavézt několik pojmů opět převzatých z biologické terminologie. Přípustné řešení se v řeči evolučních algoritmů nazývá *jedinec* (n-tice spojitých, případně reálných hodnot prohledávacího prostoru). Skupina více jedinců se označuje *populace*. Optimalizační (objektivní) funkce, na které se hledá globální maximum, nebo minimum a která ohodnocuje kvalitu jednotlivých jedinců je nazývána *fitness funkce*.

Algoritmy obecně vychází z náhodně vygenerované množiny jedinců, kteří jsou průběhu mnoha generací mezi sebou kříženy a náhodně modifikovány - souhrnně řečeno rekombinovány tak. aby v následující generaci byli jedinci kvalitnější, tedy mající lepší hodnotu fitness funkce. Tento proces běží iterativně, dokud není dosaženo jisté koncové kritérium. Takovým kritériem může být například maximální počet generací, maximální počet vyhodnocení funkce fitness, dosažení minimální změny fitness funkce nejlepšího jedince ve dvou, nebo více po sobě jdoucích generacích a mnoho dalších. Pro podrobnější studium ukončovacích kritérií bych čtenáře odkázal na článek [3]. Volba správného koncového kritéria je jednou z klíčových otázek správného návrhu evolučních algoritmů a je specifická pro každý problém. Jednotné schéma EA je zachyceno na obrázku 2.1.

Nedílnou součástí všech evolučních algoritmů jsou tzv. *kontrolní parametry*. Jsou to konstanty, které modifikují běh algoritmu a které jsou klíčové pro dosažení relevantních řešení dané úlohy. Navíc ke každé úloze je vhodné jiné nastavení kontrolních parametrů a jejich optimální nastavení je opět optimalizační úloha. Kontrolní parametry mají ještě jeden význam a tím je vyvážení vhodného průběhu konvergence algoritmu ke globálnímu optimu. Jedná se o vyvážení dvou protichůdných procesů *explorace* a *exploatace*. *Explorace* se podobá stochastickému procesu náhodné procházce a garantuje prohledání celého rozhodovacího prostoru. Zatímco *exploatace* je proces opačný, který garantuje rychlé nalezení optimálního řešení. Pokud je nastaven příliš velký podíl *exploatace*, potom algoritmu inklinuje k nalezní pouze lokálního extrému. Druhým pólem je nastavení nadměrné míry *explorace*, která má za následek nahodilé procházení prohledávaného prostoru a tedy opět nenalezení globálního extrému. Pro úspěšnou konvergenci algoritmu je proto důležitá jistá míra rovnováhy mezi *explorací* a *explolatací*. Žádná z oněch vlastností nesmí dominovat nad tou druhou. Není totiž žádoucí ani příliš rychlá ani příliš pomalá konvergence, která vede často k nalezení pouze lokálního.



Obrázek 2.1: Jednotné schéma běhu evolučních algoritmů – nejprve se inicializuje populace, ohodnotí se fitness funkcí a následně probíhá evoluční cyklus, dokud není splněno koncové kritérium.

2.2 Genetické algoritmy

2.2.1 Obecně

Genetický algoritmus (GA) je zřejmě nejznámější algoritmus patřící do rodiny evolučních algoritmů. GA lze používat jak pro optimalizaci spojitého, tak i diskrétního prostoru. Jeho původní návrh však počítá pouze s jedinci reprezentovanými binárními řetězci, tedy diskrétní optimalizací [5]. V této práci bude, pro potřeby smíšené optimalizace, GA využit v obou zmíněných případech. Genetický algoritmus zapadá do schématu běhu evolučních algoritmů (2.1)a ve své základní podobě využívá operátory *selekce*, *mutace* a *křížení*. V některých variantách se může objevit i tzv. elitizmus, což je zaručený postup nejlepších jedinců z jedné generace do druhé.

2.2.2 Selekce

Selekce je operace, která produkuje rodiče (jedinci, na které jsou následně aplikovány operace křížení a mutace) pro následující generaci. Základní selekční operátory jsou:

Ruletová selekce - *Ruletová selekce* je operátor, jenž vybere rodiče pro křížení na základě fitness jednotlivých jedinců. Pro případ maximalizace mějme populaci Pop s jedinci $\{\vec{x}_1, ..., \vec{x}_n\}$, fitness funkci $F : Pop \to \langle 0, \infty))$, pro kterou si označme $F_i \equiv F(\vec{x}_i)$. Potom

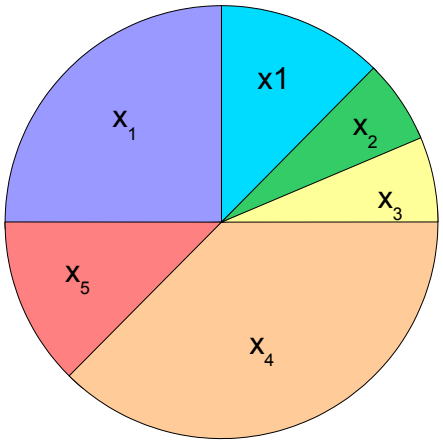
$$P(\vec{x_i}) = F_i / \sum_{j=1}^{n} F_j \tag{2.1}$$

je pravděpodobnost, že i-tý jedinec bude mezi vybranými rodiči. Je tedy

zjevné, že čím větší je ohodnocení fitness jedince, tím má větší pravděpodobnost, že se stane rodičem, a tedy jeho potomek bude součástí příští generace. V případě minimalizace stačí do vztahu 2.1 za F_i dosadit $-F_i$. Pravděpodobnostní rozdělení fitness funkcí je znázorněno na obrázku

Turnajová selekce - Další možností volby selekčního operátoru je *turnajová selekce*. Jejím vstupem jsou dva parametry: t - velikost turnaje, *lev* - počet úrovní turnaje. Náhodně se z populace vyberou skupinky jedinců o velikosti t, které se zúčastní turnaje. Jedinci v jednotlivých turnajích se spolu utkají [1] s tím, že do další úrovně turnaje postoupí vždy lepší ze dvojice jedinců. Opakováním tohoto postupu až do úrovně *lev* se vytvoří množina rodičů požadovaného počtu.

Pro modifikaci poměru *explorace* a *exploatace* je možné použít ještě tzv. *elitismus*, jenž zajišťuje postup k in nejlepších jedinců z aktuální generace do té následující. Pokud se tedy nastaví příliš vysoké k, algoritmus příliš rychle zkonverguje a zastaví se v některém lokálním extrému. Je proto pro získání relevantních výsledků algoritmu vhodně nastavit kromě jiných i tento parametr.



x_i	$F(x_i)$	$p(x_i)$
[0 1 0 1]	4	1/8
[0 0 0 1]	2	1/16
[1 0 0 1]	2	1/16
[1 1 1 0]	12	3/8
[1 0 0 1]	4	1/8
[0 0 0 1]	8	1/4

Obrázek 2.2: Ruletová selekce

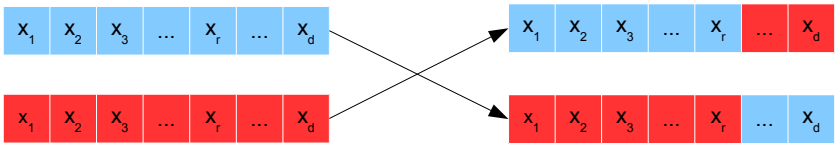
2.2.3 Křížení

Křížení je operace, která zajišťuje exploataci, neboli konvergenci jedinců k optimálnímu řešení. Základním parametrem této operace je P_{cross}, neboli pravděpodobnost křížení. Tento parametr specifikuje s jakou pravděpodobností se na rodiče z aktuální populace aplikuje operátor křížení. Existuje mnoho typů toho operátoru použitelných pro různé typy optimalizačních úloh. Patří sem kromě jiných například jednobodové, vícebodové, nebo aritmetické křížení.

Jednobodové: Nechť d je doména problému, neboli počet genů v jedinci. Mějme dva rodiče $\vec{x_1} = \{x_{1,1}, \ldots, x_{1,d}\}$ a $\vec{x_2} = \{x_{2,1}, \ldots, x_{2,d}\}$, potom se náhodně

[1] Utkáním se myslí porovnání fitness ohodnocení obou jedinců

zvolí index $r \in \{1, \ldots, d-1\}$, potom jejich dva potomci budou v následujícím tvaru: $\{x_{1,1}, \ldots, x_{1,r}, x_{2,r+1}, \ldots, x_{1,d}\}$ a $\{x_{2,1}, \ldots, x_{2,r}, x_{1,r+1}, \ldots, x_{1,d}\}$. Schématicky je tato operace jednobodového křížení znázorněna v obrázku 2.3.



Obrázek 2.3: Jednobodové křížení – ze dvou jedinců (rodičů) vzniknou dva potomci.

2.2.4 Mutace

Na rozdíl od křížení operátor mutace garantuje exploraci prostoru řešení. Principiálně se jedná o náhodnou změnu genomu jedince. Základním parametrem operace mutace je P_{mut}, který udává pravděpodobnost, že na daného jedince (rodiče vybraného selekčním operátorem), se bude aplikovat operace mutace. Stejně jako u operace křížení existuje u této operace několik variant pro spojitý i diskrétní prostor. Nejjednodušší diskrétní mutace funguje tak, že se pro každého mutovaného jedince $mut \in \{1, \ldots, n\} : \{x_{mut,1}, \ldots, x_{mut,d}\}$ náhodně zvolí index $r \in \{1, \ldots, d\}$ a jedinci se na této pozici náhodně změní hodnota. Výsledný zmutovaný jedinec tak má tento tvar: $\{x_{mut,1}, \ldots, x_{mut,r-1}, y, x_{mut,r+1}, \ldots x_{mut,d}\}$, kde y je náhodná nová hodnota. 2.4. K této mutaci lze vymyslet mnoho modifikací jako například změna na více pozicích apod.

Obrázek 2.4: Operace mutace

Další možností mutace, tentokrát ve spojitém prostoru, je tzv *Gaussovská mutace* tzn., že jedinec je „rozostřen" normálním rozdělením $\mathbf{N}(0, \Sigma)$.

2.3 Diferenciální evoluce

2.3.1 Obecně

Diferenciální evoluce (DE) je evoluční algoritmus vyvinutý v letech 1994-1995 Kenneth Pricem a Rainer Stornem [6]. DE vzešel z algoritmu Genetického žíhání (Genetic Anealing), jenž byl postupně modifikován pro spojitý prostor. K takto upravenému algoritmu byla doplněna operace diferenciální mutace, která se stala

Algorithm 1 Jednoduchý genetický algoritmus - Simple genetic algorithm

Require: $P_{mut}, P_{cross}, SelType, CrossoverType, MutationType, Elitism, MaxGen$

```
Pop^0 ← RandInit
Fitness^0 = Eval(Pop^0)
g ← 0
while g ≤ MaxGen do
  for i = 1 to NPdiv2 do
    ind_{1,2} ← Select(Pop, Fitness, SelType)
    new_ind_{i,i+NPdiv2} ← Recombination(Pop^g, P_cross, CrossoverType)
    new_ind_{i,i+NPdiv2} ← Mutation(Pop^g, P_mut, MutationType)
  end for

  if Elitism == true then
    Pop^{g+1} = EliteReplacement(Pop^g, Pop^new)
  else
    Pop^{g+1} = Replacement(Pop^g, Pop^new)
  end if
end while
```

hlavním rysem nového algoritmu – *Diferenciální evoluce*. Jeho následnou modifikací a zobecněním vznikl tzv. *Moderní diferenciální evoluce*. Tento algoritmus zahrnuje mnoho různých variant a strategií, z nichž některé budou popsány níže.

DE je primárně určen pro optimalizci spojitého prostoru, ale lze ho snadno modifikovat na celočíselné, nebo diskrétní problémy. Kromě toho umožňuje řešit *multimodální* i *multiobjektivní* optimalizaci. Pro některé úlohy lze aplikovat tzv. *hybridizaci*, což je kombinace DE s jinými prohledávacími metodami (lokální prohledávání apod.).

Stejně jako u ostatních evolučních algoritmů jsou důležitým atributem DE kontrolní parametry, jejichž správné nastavení může mít klíčový dopad na konvergenci a správný výsledek algoritmu. DE má následující kontrolní parametry:

CR - pravděpodobnost křížení

F - diferenciační (mutační) konstanta

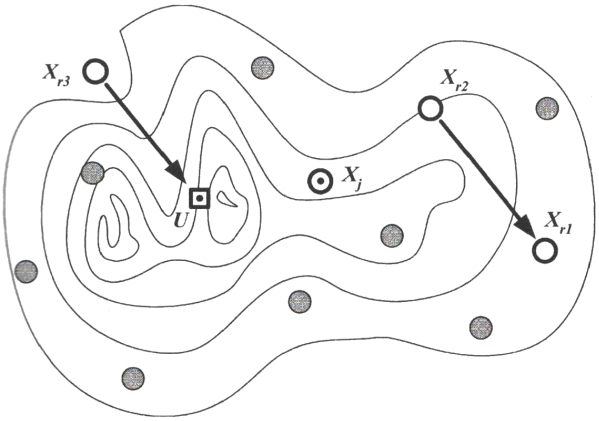
NP - počet jedinců v populaci

GEN - maximální počet generací

D - doména problému; tj. pro spojitý prostor každý jedinec X leží v prostoru $\mathbb{R}^D$

L, H - dolní a horní hraniční omezení; tj platí $\forall X :\in \mathbb{R}^D L \leq X \leq H$

Mezi kontrolní parametry je možné zařadit také volbu prohledávací strategie (RAND, RAND/DIR, RAND/BEST, RAND/BEST/DIR), protože i jeho správná volba může zásadním způsobem ovlivnit výstup algoritmu. DE pro prohledávání prostoru řešení využívá dvě evoluční operace - *diferenciální mutace* a *křížení*.



Obrázek 2.5: Princip diferenciální mutace DE

2.3.2 Původní diferenciální evoluce

Diferenciální evoluce (Differential Evolution – DE) byl prvním algoritmem spadajícím do kategorie diferenciální algoritmů. Pro jeho demonstraci předpokládejme spojitou maximalizační úlohu. Nechť x^* je optimální řešení úlohy a $F : M \subseteq \mathbb{R}^D \to \mathbb{R}$ objektivní funkce. Potom pro hledané optimální řešení x^* platí $\forall x \in M : f(x^*) \geq F(x)$, kde M je omezená kompaktní množina.

Běh algoritmu DE zapadá do schématu evolučních algoritmů. Vychází se tedy z náhodně zvolené populace, která je iterativně upravována opracemi diferenciální mutace a křížením, dokud není splněno koncové kritérium. Schéma celého algoritmu je na obrázku 2. Diferenciální mutace a křížení pro DE probíhá v podstatě jako jedna operace:

$$x_i = \begin{cases} x_{ir_3} + F \cdot (x_{ir_2} - x_{ir_1}) & \text{když} \quad rand_{ij} \geq CR \vee rnd = i \\ x_{ij} \cdot (x_1 - ind) & \text{jinak} \end{cases} \tag{2.2}$$

Kde $rand_{ij}$ resp. rnd je náhodně vygenerované číslo z intervalu $\langle 1, D \rangle$ resp. $\langle 1, D \rangle$ a CR (pravděpodobnost křížení) je jeden ze vstupních kontrolních parametrů.

2.3.3 Moderní Diferenciální evoluce

Moderní Diferenciální evoluce (Modern differetnial evolution - MDE) vychází z DE algoritmu a v podstatě se jedná o jeho zobecnění. Prvním rozdílem je oddělení operace mutace od křížení do dvou oddělených operací. To nám umožňuje obě operace nezávisle na sobě zkoumat a zjišťovat vliv na konvergenci algoritmu. Druhou změnou je zobecnění operace diferenciální mutace na toto schéma:

$$\omega = \beta + F \cdot \delta \tag{2.3}$$

kde místo náhodného vektoru x_{r1} z algoritmu SDA je použit tzv. bázový vektor β. Parametr δ zastupuje místo diferenciálního posunu vektorů $(x_{r2} - x_{r3})$. Konkrétním naplněním parametrů β a δ vznikají různé prohledávací strategie, které se mohou chovat odlišně na specifických optimalizačních úlohách.

Algorithm 2 Moderní Diferenciální evoluce

Require: NP - velikost populace, F - difernciační konstanta, CR - pravděpodobnost křížení, $MaxGen$ - maximální počet generací, D, L, H - dolní a horní mez, $Strategy$ - volba strategie, Fit - fitness funkce

$Pop^0 \leftarrow InitPop(L, H)$
$Pop^0 \leftarrow EvalPop(Pop^0, Fit)$
$g \leftarrow 0$
while $g < MaxGen$ **do**
  for $i = 1$ to NP **do**
    $\pi \leftarrow SelectRandIndividuals(Pop^g, Strategy)$
    $\omega \leftarrow DiffMutation(Pop^g, Strategy, \pi, F)$
    $\omega \leftarrow Crossover(Pop^g, Strategy, \pi, CR)$
    $\omega \leftarrow CheckBoundries(\omega, L, H)$
    $Pop_i^{g+1} \leftarrow Replacement(\omega, Pop_i^g)$
  end for
  $g \leftarrow g + 1$
end while

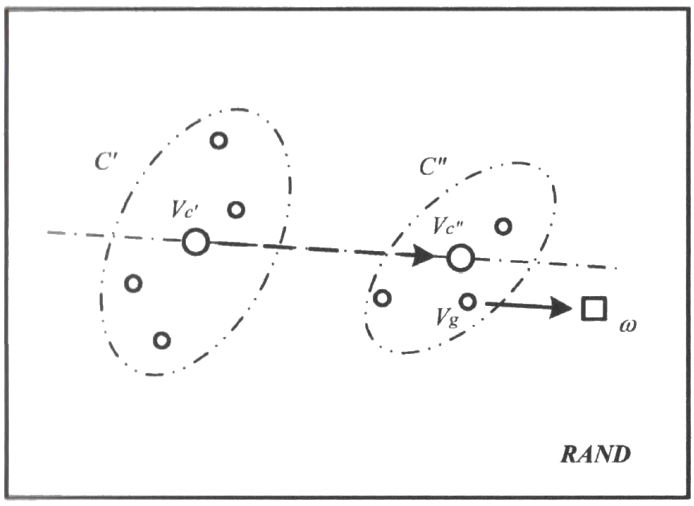
2.3.4 Prohledávací strategie

Již K. Price a R. Storn ve své knize o Diferenciální evoluci [6] definovali určitá schémata prohledávání tzv. prohledávací strategie. Jedná se o dosazení za parametry β a δ do vztahu 2.3:

1. strategie DE/RAND/1 - $\omega = x_1 + F \cdot (x_2 - x_3)$
2. strategie DE/RAND/2 - $\omega = x_5 + F \cdot (x_1 + x_2 - x_3 - x_4)$
3. strategie DE/BEST/1 - $\omega = x_{best} + F \cdot (x_1 - x_2)$
4. strategie DE/BEST/2 - $\omega = x_{best} + F \cdot (x_1 + x_2 - x_3 - x_4)$
5. strategie DE/RAND-TO-BEST/1 - $\omega = x_{best} + F \cdot (x_1 + x_2 - x_3 - x_4)$

Zmíněné strategie lze trochu přehledněji klasifikovat do čtyř kategorií podle toho, jestli se využívá znalosti objektivní funkce nebo ne.

1. RAND je skupina strategií, kde nově vzniklý jedinec vzniká bez znalosti objektivní funkce. Obecným schématem této skupiny je $\beta = V_g$ a $\delta = \left(V_C'' - V_C'\right)$, kde C', C'' jsou dvě náhodně vybrané disjunkní skupiny jedinců, $V_C = \frac{1}{n}\sum_{i=1}^{n} x_i$, jsou jejich barycentra, a V_g může být zvoleno buď jako náhodný prvek monžiny $C' \cup C''$, nebo mimo ní. Konkrétními příklady strategií typu RAND jsou (viz také obrázek 2.6):
   - RAND1 $\omega = x_1 + F \cdot (x_1 - ind)$, kde x_1 je náhodně zvolený jedinec a ind je původní jedinec, který je mutován.
   - RAND2 se oproti RAND1 strategii liší pouze tím, že jedinec ind je nahrazen druhým náhodným jedincem z populace x_2. Tedy dostáváme schéma $\omega = x_1 + F \cdot (x_1 - x_2)$
   - ...



Obrázek 2.6: Ukázka strategie RAND

2. RAND/DIR je skupina strategií, která využívá fitness funkce pro volbu správného směru δ. Tato skupina se snaží napodobovat gradientní metody prohledávání. Nechť C_+ a C_- jsou opět dvě disjunktní množiny náhodných jedinců z populace, pro něž platí $F(x_i) \leq F(x_j) \forall x_i \in C_+, x_j \in C_-$[2].

   Následně se spočítají posuny uvnitř tříd C_+, C_-

$$V_s^{\pm} = \lambda\dot{(}X_{C_\pm}^{min} - X_{C_\pm}^{max})V_S = (V_s^+ + V_s^-)/2 \tag{2.4}$$

   Obecné schéma těchto strategií je potom

$$\omega = V_{C_+} + F.(V_{C_+} - V_{C_-} + V_S) \tag{2.5}$$

   Konkrétními příklady strategií typu RAND/DIR jsou (viz také obrázek 2.7):

   - RAND1/DIR1

$$\omega = \begin{cases} x_1 + F \cdot (x_1 - ind) & \text{když} f(x_1) < f(ind \\ ind + F \cdot (x_1 - ind) & \text{jinak} \end{cases}$$

   - RAND2/DIR1

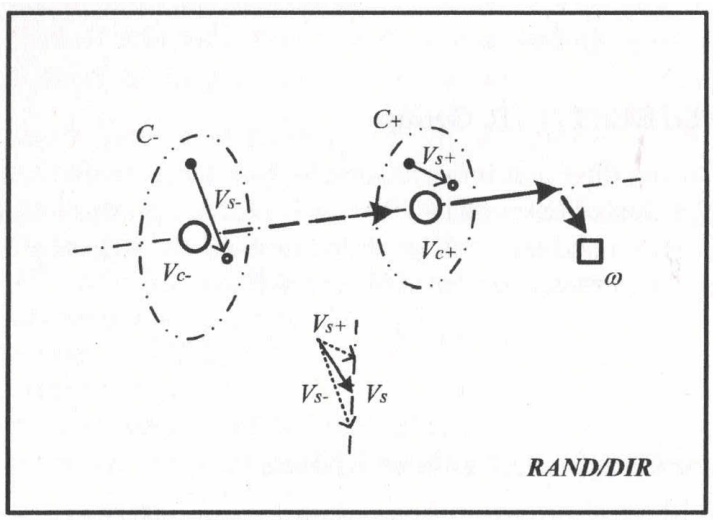
$$\omega = \begin{cases} x_1 + F \cdot (x_1 - x_2) & \text{když} f(x_1) < f(x_2) \\ x_2 + F \cdot (x_2 - x_1) & \text{jinak} \end{cases}$$

   - viz další strategie v [6]

3. RAND/BEST je skupina strategií, které volí jako bázový vektor nejlepšího jedince $V_b = x_{best}$. Schématem této skupiny je

$$\omega = V_b + F \cdot (V_C'' - V_C') \tag{2.6}$$

[2] v případě minimalizace



Obrázek 2.7: Ukázka strategie RAND/DIR

kde $V_C^{'}, V_C^{''}$ jsou definovány stejně jako u strategií RAND.

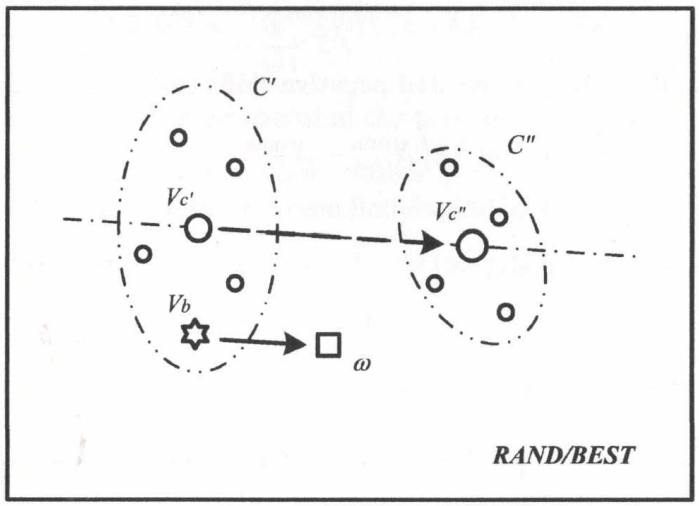
Konkrétními příklady strategií typu RAND/BEST jsou (viz také obrázek 2.8):

- RAND1/BEST je kombinace RAND1 strategie s aplikací nejlepšího jedince x_{best} na místo bázového vektoru. Diferenciální mutace se tedy spočítá
$$\omega = V_b + F.(V_C^{''} - V_C^{'}) \tag{2.7}$$
- ...

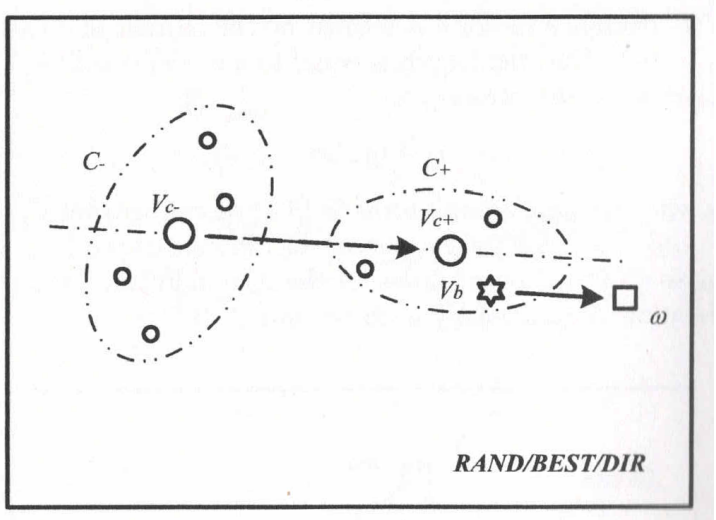
4. RAND/BEST/DIR je skupina strategií, která vzniká kombinací strategií ze skupin RAND/BEST a RAND/DIR. Schéma této skupiny proto vypadá následovně:
$$\omega = V_b + F.(V_{C_+} - V_{C_-} + V_S) \tag{2.8}$$
Ukázka konkrétní strategie RAND/BEST/DIR je na obrázku .

2.3.5 Shrnutí Diferenciální evoluce

Z náčrtu algoritmu 2 je zjevné, že tento algoritmus zapadá do obecného schématu evolučních algoritmů. Inicializace a evaluace je ponechána beze změny. Co se týká vlastní mutace a křížení populace v MDE je na rozdíl od algoritmu DE realizována sekvenčně na všechny jedince, čímž se zajistí udržování jisté diverzity populace během prohledávání. Výhodou algoritmu je jednoduchost, aplikovatelnost na spojitý i diskrétní prohledávací prostor a široká modulárnost. V neposlední řadě je nutné zmínit důležitou vlastnost algoritmu, kterou je *autoadaptivita* prohledávacích parametrů. *Autoadaptivita* je způsobena faktem, že s rostoucím počtem generací a tedy s postupnou konvergencí algoritmu ke globálnímu optimu se diferenciační krok δ zmenšuje, a tím zamezuje velkým skokům v pokročilém stádiu prohledávání. Díky této vlastnosti se za běhu také mění poměr explorace a exploatace.



Obrázek 2.8: Ukázka strategie RAND/BEST



Obrázek 2.9: Ukázka strategie RAND/BEST/DIR

Na začátku je diferenciální krok velký, tedy algoritmus inklinuje spíše k exploraci prohledávaného prostoru. Na druhou stranu v závěru, kdy se diferenciální krok zmenšuje, převažuje spíše exploatace.

2.4 Evoluční strategie s adaptací kovarianční matice a dynamickým vytvářením nik

2.4.1 Evoluční strategie

Evoluční strategie (ES) je další optimalizační algoritmus který byl vyvinut v 70.letech 20.století. Jejími autory jsou pánové I. Rechenberg a P. Schwefel [7]. Tento algoritmus slouží pouze pro spojité optimalizační problémy. Pro iteraci populace z jedné generace do druhé zde slouží operátory selekce, mutace a nahrazení (replacement). Operace selekce je nejčastěji reprezentována náhodným vzorkováním jedinců z rovnoměrného rozdělení. Kromě rozhodovacích parametrů jednotlivých jedinců se zde objevují i tzv. strategické parametry. Jsou to σ parametry [3] Gaussova normálního rozdělení, které slouží pro rozostření jedinců při aplikaci operce mutace. Jednotlivé σ parametry jsou v každé iteraci adaptovány podle úspěšnosti nové populace, díky tomu mají evoluční strategie důležitou vlastnost a totiž samoadaptaci kroku algoritmu v průběhu konvergence. Pro adaptaci těchto parametrů je aplikováno pravidlo 1/5. Nahrazení rodičovské generace generací potomků se dělí na dvě základní strategie podle toho, jací jedinci mohou postoupit do následující generace na (μ, λ) a $(\mu + \lambda)$, kde μ je počet rodičů, λ je počet potomků a $\mu \leq \lambda$. U (μ, λ)-ES algoritmů se z μ rodičů navzorkuje λ potomků, z nichž se vybere do nové generace μ nejlepších jedinců. Může proto nastat situace, že se nejlepší jedinec z předchozí generace nepřenese do té následující. Na druhé straně u $(\mu + \lambda)$-ES se pro následující generaci vybírají nejlepší jedinci ze sjednocení μ rodičů a λ jejich potomků. Někdy se proto této strategii říká elitistická. Pro více podrobností o evolučních strategiích naleznete v článku [7].

2.4.2 Evoluční strategie s adaptací kovarianční matice

Algoritmus *Evoluční strategie s adaptací kovarianční matice* (Evolutionairy Strategies with Covariance Matrix Adaptation - ES-CMA) je speciální variantou ES, který byl úspěšný díky zachycení závislostí mezi proměnnými. Adaptace kovarianční matice je metoda, jak adaptovat mutační parametry algoritmu. Dále budeme pracovat s variantou $(1, \lambda)$-ES-CMA, tedy z jednoho rodiče vznikne pro novou generaci λ potomků. Nejlepší jedinec z těchto λ jedinců postoupí do nové generace, na jejímž základě se adaptuje kovarianční matice.

Základní vlastností této metody je propagace informace získané z předchozích úspěšných mutací. Předpokládejme, že máme jedince v úvodní populaci $\vec{x}^0$. Nová populace vznikne tak, že se mutačním operátorem vytvoří λ nových potomků, z nichž se vybere pouze ten nejlepší jedinec. Generování nových jedinců probíhá podle následujícího vztahu:

[3] V jednorozměrném normálním rozdělení to je rozptyl, ve vícerozměrném pak kovarianční matice

$$\vec{x}^{g+1} = \vec{x}^{g} + \sigma \cdot B \cdot \vec{z} \tag{2.9}$$

kde g je aktuální generace, σ je krok algoritmu, B je matice vlastních vektorů kovarianční matice C vícerozměrného normálního rozdělení a $\vec{z} \approx \mathbf{N}(0, I_n)$ je náhodně vybraný vzorek z vícerozměrného normálního rozdělení se střední hodnotou 0 a jednotkovou kovarianční maticí I_n. Tímto způsobem se navzorkují noví potomci, ale klíčovou součástí celého algoritmu je *adaptace kovarianční matice* C. Více o této metodě naleznete v článku [8].

2.4.3 Evoluční strategie a dynamické vytváření nik

Jednou z prvních metod, které spadají do schématu algoritmu ES je tzv. *Evoluční strategie s dynamickým vytvářením nik* (ES-DN) [9]. Tato metoda je vhodná zejména při řešení tzv. *multimodální optimalizace*, protože si algoritmus udržuje jistou diverzitu mezi jedinci [4] tak, aby nezkonvergovaly do jednoho lokálního optima. Tím je zajištěno jednak pokrytí prohledávacího prostoru, ale také rozdělení populace do několika epicenter, kde může probíhat optimalizace nezávisle. Výhoda diverzity populace je markantní i při jednocílové optimalizaci, v níž je více niky zaručeno prohledání větší části prohledávaného prostoru. Pro vzorkování nových jedinců a adaptaci rozhodovacích parametrů je v tomto algoritmu použita obdobná metoda jako je tomu u algoritmu ES (tedy adaptace pravidlem pětin).

Vstupem algoritmu je parametr q, který označuje odhadovaný nebo očekávaný počet *nik*, do nichž se bude populace dělit. Vhodná hodnota musí být odhadnuta pro každou úlohu zvlášť. Evoluční cyklus algoritmu je zahájen náhodnou mutací populace, která je následně ohodnocena fitness funkcí a rozdělena do q nik pomocí funkce DPI [5] . Jako metrika vzdálenosti dvou jedinců je použita *eukleidovská metrika*. Každá nika má svůj vrchol. Každé dva vrcholy musí splňovat podmínku, že jejich vzdálenost musí být větší než $2 \cdot \rho$, kde ρ je poloměr nik a jeho výpočet je popsán ve článku [9]. Splněním této podmínky se zajistí požadovaný invariant tj. jistá míra diversity populace. Následně se pro každou niku vyberou jedinci, kteří se rekombinují a postoupí do následující generace. Má-li jedna nika μ jedinců, pak vygenerujeme λ potomků následujícím postupem. První rodič vznikne turnajovou selekcí a druhý je pak vzat jako nejlepší jedinec z niky, který je jiný, než první jedinec. Takto vznikne λ párů rodičů, kteří jsou poté vzájemně zkříženi: *strategické parametry* jsou kříženy průměrováním (crossover intermediate) a rozhodovací parametry pak diskrétním křížením, čímž dostaneme λ potomků. Z těchto potomků vybereme μ jedinců do nové generace tak, že vezmeme η nejlepších jedinců z λ potomků společně s nejlepšími $\mu - \eta$ jedinci z původní niky. Stručně je algoritmus naznačen ve schématu . Více podrobností o této optimalizační metodě naleznete ve článku [9].

[4] Ve smyslu tohoto algoritmu se takoví jedinci nazývají vrcholy (peaks). Jedná se o jedince, kteří jsou uvnitř nik a mají nejlepší ohodnocení fitness.

[5] Dynamic Peak Identification - dynamická identifikace vrcholů

Algorithm 3 ES-DN

Require: q - odhadovaný počet vrcholů, ρ - minimální vzdálenost mezi dvěma vrcholy (poloměr), NP, $MaxGen$

```
Pop^0 ← InitRand(...)
g ← 0 {Evoluční cyklus}
while g ≤ MaxGen do
  for i = 1 to q + 1 do
    NewPop ← GenerateOffspringCMA(λ)
  end for
  Fitness ← EvaluateSort(NewPop)
  DPS ← DynamicPeakIdnetification(NewPop, Fitness, q, ρ)
  for all Peak ∈ DPS do
    SetAsNewSearchPoint(Peak)
    UpdateCMASet()
  end for
  N ← size(DPS)
  if N < q then
    GenerateNewSets(q − N)
    UpdateCMASet()
  end if
  g ← g + 1
end while
```

Algorithm 4 DPI - Dynamic Peak Identification

Require: Pop - vstupní populace, Fit - fitness funkce, NP - velikost populace, q - odhadovaný počet nik, ρ - poloměr nik {Funkce DPI vrátí seznam vrcholů nově vzniklé populace. Funkce zajistí, že žádné dva vrcholy nebudou od sebe vzdálené méně než ρ}

```
Pop ← Sort(Pop, Fit, DESC)
I ← 1
NumPeaks ← 0
DPS ← []
while NumPeaks = q‖I = N + 1 do
  if Pop_i není zahrnut v ρ vzdálenosti od exitujících DPS then
    DPS ← DPS ∪ {Pop_i}
    NumPeaks ← NumPeaks + 1
  end if
  I ← I + 1
end while
return DPS
```

2.4.4 Evoluční strategie s adaptací kovarianční matice a dynamické vytváření nik

Evoluční strategie s adaptací kovarianční matice a dynamické vytváření nik (Dynamic Niching in Evolution Strategies with Covariance Matrix Adaptation - ES-CMA-DN) je algoritmus, který vychází z algoritmu ES-DN, ovšem místo použití evolučních operací z ES se zde používají operace z algoritmu $(1, \lambda)-$ES-CMA. Algoritmus ve stručnosti funguje následovně. Mějme q odhadnutý počet nik. Během algoritmu se udržují tzv. $q+1$ CMA-množin, kde jedna CMA-množina zahrnuje veškeré proměnlivé proměnné, nebo parametry ES-CMA algoritmu jako je kovaranční matice, velikost kroku apod. q CMA-množin je bráno pro každý vrchol a $q+1$. je počítáno pro tzv. ne-vrchol, tedy jedince, který je v každé generaci náhodně inicializován. Tento jedinec zajišťuje prohledání celého prostoru. V evolučním cyklu je v jedné generaci z každého vrcholu navzorkováno λ potomků na základě jeho distribuce definované v příslušné CMA-množině. Po ohodnocení fitness funkcí nových $\lambda \cdot (q+1)$ jedinců proběhne klasifikace do nik pomocí DPI funkce z algoritmu ES-DN 4. Vrcholy z nově vzniklých nik jsou vzaty jeko jedinci do nové populace. Pokud funkcí DPI se nepodaří nalézt q vrcholů, pak zbylé vrcholy pro novou populaci jsou vygenerovány náhodně. V každém případě náhodně přegenerován $q+1$. vrchol. Pro ilustraci algoritmu následuje schéma algoritmu .

Algorithm 5 ES-CMA-DN

Require: $q, \rho, NP, PXover, MaxGen, Dim, L, H, Strategy$ {q je odhadovaný počet vrcholů, ρ je minimální vdálenost mezi vrcholy }

```
Pop^0 ← InitRand(...)
g ← 0 {Evoluční cyklus}
while g ≤ MaxGen do
  for i = 1 to q + 1 do
    NewPop ← GenerateOffspringCMA(λ)
  end for
  Fitness ← EvaluateSort(NewPop)
  DynamicPeakSet ← DynamicPeakIdnetification(NewPop, Fitness, q, ρ)
  for all Peak ∈ DynamicPeakSet do
    SetAsNewSearchPoint(Peak)
    UpdateCMASet()
  end for
  N ← size(DynamicPeakSet)
  if N < q then
    GenerateNewSets(q − N)
    UpdateCMASet()
  end if
  g ← g + 1
end while
```

3. Modifikace algoritmů pro účely optimalizace katalýzy

Některé algoritmy zmíněné v kapitole 2 nesplňují včechny požadavky, které vyžaduje naše optimalizační úloha popsaná v 1.3. Proto je nutné zmíněné algoritmy modifikovat, tak aby splňovaly tří základní požadavky vyplývjící ze zadání:

1. Smíšená optimalizace
2. Lineární omezení
3. Diversita populace

V této kapitole budou nejprve popsány všechny zmodifikované algoritmy Genetický algoritmus pro diskrétní a spojitou část, Diferenciální evoluce pro spojitou část a konečně Evoluční strategie s adaptací kovarianční matice a dynamické vytváření nik. Závěrem bude jako důsledek těchto úprav navržen pseudoalgoritmus, který je určen pro řešení celé komplexní optimalizační úlohy a který nastiňuje aplikační rozhraní celého optimalizačního algoritmu.

3.1 Genetický algoritmus pro diskrétní prostor

Jako diskrétní algoritmus byl pro naše testování použit modifikovaný Genetický algoritmus. Vstupem tohoto algoritmu je populace přípustných diskrétních jedinců. Každý diskrétní jedinec reprezentuje neprázdný mnohostěn. Obecně řečeno v diskrétním algoritmu se v podstatě řeší křížení a mutace mnohostěnů. Body uvnitř těchto mnohostěnů reprezentují spojité proměnné, jenž optimalizovány pomocí některého z algoritmů pro spojitý prostor. Naší snahou při návrhu tohoto algoritmu je zachování jisté diverzity populace. Diverzitou se zde rozumí odlišnost všech jedinců jak v diskrétní, tak i ve spojité části. Udržování diverzity populace je klíčové pro konvergenci celého algoritmu, ale také je nezbytná pro efektivní výpočet fitness funkce, která se vyhodnocuje experimentálně. Při našem testování jsme ovšem vycházeli z aproximace fitness funkce pomocí RBF sítí, a proto není nutné dobu vyhodnocení ani finanční hledisko brát v potaz. Cílem úpravy tohoto Genetického algoritmu bylo navrhnout genetické operace *mutace* a *křížení* nad diskrétní částí populace tak, aby splnňovaly všechny výše zmíněné požadavky. Vzhledem k složité stromové struktuře komponent a podkomponent, není možné pouze aplikovat existující operátory, jenž pracují nad čistě diskrétním prostorem.

Jako operátor *mutace* je zde použit náhodný vyběr jedince z množiny všech přípustných diskrétních jedinců tedy jedinců reprezentujících neprázdné mnohostěny. Pravděpodobnostní rozdělení náhodného výběru zachovává pravěpodobnosti existence jednotlivých komponent i jiných diskrétních parametrů definujících katalytickou reakci. Toto rozdělení je rovnoměrné, pokud optimalizační struktura nedefinuje žádné pravděpodobnosti komponent.

Operace *křížení* je upravena, tak aby zachovávala princip rekombinace, tedy dědičnosti některých vlastností rodičovské generace do generace potomků. Kvůli stromové struktuře jednotlivých diskrétních komponent, není možné aplikovat

například aritmetické, nebo vícebodové křížení rodičů z genetického algoritmu. Nebyl by totiž zajištěn invariant, že všichni jedinci reprezentují přípustné řešení úlohy, protože aplikací standardních operátorů by mohl vzniknout jedinec nesplňující lineární omezující podmínky. Navrhovaný operátor křížení využívá předpočítanou datovou strukturu zachycující pro všechny jedince z existující populace seznam nejbližších sousedů z téže populace. Jako vdálenost dvou jedinců je použita Hammingova metrika restringovaná na diskrétní část rozhodovacích parametrů. Tato struktura je zobrazena na obrázku [Obr predpocitana struktura pro krizeni]. Potomek dvou rodičů operací křížení je potom definován jako jedinec, který je obsadžen v jednom ze seznamů nejbližších sousedů a zároveň má minimální vzdálenost vůči všem jedincům ze seznamu sousedů druhého rodiče. Tímto způsobem jsou garantovány oba požadované invarianty algoritmu.

Pro zlepšení konvergence je možné v navrhovaném diskrétním genetickém algoritmu aplikovat *elitismus*, která garantuje převod nejlepších jedinců z jedné generace do druhé.

3.2 Genetický algoritmus pro spojitý prostor

Prvním algoritmem pro spojitý prostor je opět genetický algoritmus. V tomto případě byla využita existující implementace algoritmu v prostředí Matlab, které umožňuje genetické operace s lineárním omezením a jím definovaným mnohostěnem přípustných řešení. Operátory, které zachovávají přípustnost v rámci jednoho mnohostěnu jsou: *mutationAdaptFeasible* pro operaci mutace a *crossoverScattered*, c, nebo *crossoverAritmetic* pro křížení, což jsou funkce implementované v GADS toolboxu. Více informací o genetickém algoritmu naleznete v oficiální dokumentaci příslušného Toolboxu na [12]. Tento genetciký algoritmus byl také modifikován tak, aby produkoval pouze odlišné jedince.

3.3 Diferenciální evoluce

Druhým algoritmem, který slouží pro optimalizaci spojitého části katalytické optimalizace je diferenciální evoluce. Algoritmus vychází z existujícího algoritmu Moderní diferenciální evoluce nastíněného v kapitole 2.4. Příslušný algoritmus ovšem nezahrnuje práci s lineármím omezením rozhodovacích parametrů, a tudíž pro naše účely musí být upraven. Operátor aritmetidkého křížení nemusí být modifikován, protože zachovává přípustnost řešení. Na druhé straně operátor diferenciální mutace upraven být musí. Pro připomenutí je výpočet zmutovaného potomka v Moderní diferenciální evoluci definován vztahem 2.3. Parametr δ má v tomto vztahu význam směrového vektoru, tedy směr posunu od jiného bázového vektoru β. Mutace je proto jednoznačně určena přímkou procházející bázovým vektorem β a směrovým vektorem δ. Modifikace tohoto operátoru pro zachování lineárního omezení má proto charakter nalezení takového bodu na přímce, který je uvnitř mnohostěnu definovaného omezeníml. K tomu je zapořebí pomocí lineárního programování nalézt průsečíky přímky s mnohostěnem. Oba dva hledané průsečíky je možné spočítat ze vztahu 3.1

$$t_{max} = \max_{i:[A\cdot\vec{v}<0]} \frac{[b - A\cdot c]_i}{[A\cdot\vec{v}]} // t_{min} = \min_{i:[A\cdot\vec{v}>0]} \frac{[b - A\cdot c]_i}{[A\cdot\vec{v}]} \tag{3.1}$$

kde $A \cdot \vec{x} \leq b$ je soustava lineárních nerovnic definujících mnohostěn a $c + t\vec{v}$ je parametrické vyjádření přímky se směrovým vektorem $\vec{v}$, parametrem t a procházejícím bodem uvnitř mnohostěnu c. Výsledkem obou rovnic je hodnota parametru t v rovnici přímky. Dopočítání souřadnic průsečíků spočívá pouze v dosazení do rovnice přímky. Uvědomme si že nemusíme řešit, žádné speciální případy, kdy průsečík přímky s mnohostěnem neexistuje $A \cdot \vec{x} = b$, protože v našem případě je mnohstěn ve všech souřadnicích omezený a bod c je uvnitř mnohostěnu. Více informací o dané problematice naleznete na [10].

Postup získání nového jedince modifikovanou diferenciální mutací je takový, že nejprve se zkusí zpočítat nový jedinec pomocí operace a pokud je výsledný jedinec nepřípustný, tak se jako jeho přípustný ekvivalent použije průsečík přímky s mnohostěnem lineárního omezení.

3.4 Evoluční strategie s adaptací kovarianční matice a dynamickým vytvářením nik

Třetím algoritmem určeným pro řešení spojité části optimalizační úlohy je upravený algoritmus ES-CMA-DN, který je popsaný v kapitole 5. Opět se jedná o úpravu té části algoritmu, která způsobuje vygenerování nových jedinců v populaci. V tomto případě to je ta část algoritmu, kdy se náhodně vzorkují noví jedinci z vícerozměrného normálního rozdělení $\mathbb{N}(\mu, C)$ se střední hodnotou μ a kovarianční maticí C, rodičovské populace do nové. Cílem úpravy této mutační operace je zajistit, aby navzorkovaná populace obsahovala přípustné jedince, tedy jedince splňující lineární omezení. To ovšem není možné s jistotou zaručit u žádného jedince.

První navržené řešení bylo pomocí vzorkování jedinců, u nichž je „dostatečně" velká pravděpodobnost, že je jejich potomek bude přípustný. Tuto pravděpodobnost není snadné spočítat přesně jako integrál hustoty θ rozdělení vícerozměrného normálního rozdělení $\mathbb{N}$ přes celý mnohostěn přípustnosti, nýbrž jsme pouze schopni spočítat její dolní odhad jako integrál tytéž funkce přes menší množinu obsaženou v mnohostěnu a tou je Čebyševova koule. U ní je zaručeno, že je celá obsažena uvnitř mnohostěnu. Jak se ovšem později ukázalo, je tato metoda nevhodná pro řešení dané úlohy, protože ve většině případů je objem Čebyševovy koule vůči objemu celého mnohostěnu příliš malý, a tudíž výpočet přes takto malou množinu nemá vypovýdající hodnotu. Navíc nebylo garantováno, že Čebyševova koule je umístěna v blízkosti těžiště mnohostěnu. Tyto důvody nás přiměly od tohoto postup odstoupit.

Jako nejvhodnější a zároveň nejjednodušší řešení se ukázalo navzorkování více jedinců z rozdělení N, než bylo zapotřebí, a mezi potomky rodičovské populace se vybraly pouze ty přípustné. Pokud se opakovaně nezdaří navzorkovat dostatečný počet jedinců, pak se generace potomků doplní některými jedinci z populace rodičovské.

3.5 Hlavní algoritmus

Výsledkem úprav diskrétního a spojitých algoritmů vznikl návrh celkového pseudoalgoritmu řešícího smíšenou optimalizaci s lineárním omezením a podporujícím

zpracování optimalizační struktury popisující katalytickou reakci. Je to tedy jakýsi rámec, který nám umožní porovnatelné srovnání všech použitých algoritmů.

Algorithm 6 Pseudoalgoritmus

Require: *Optimization* - optimalizační struktura, F - fitness funkce, *Options* - vstupní parametry algoritmu

{Předzpracuje optimalizační strukturu}

$(Optimization, CL, ProbCL) \leftarrow PreprocessOptimization(Optimization)$

{Najde množinu tříd jedinců majících ekvivalentní matice omezení FCL a jejich pravděpodobnosti ProbFCL jako podmnožinu množiny všech existujících tříd CL}

$(FCL, ProbFCL) \leftarrow FindFeasibleClasses(CL, ProbCL)$

{Náhodná inicializace populace funkce RandPopulation(N, FCL). V jedné iteraci se vygeneruje vždy jeden jedinec}

for $i = 1$ to N **do**

$FCL_{rnd} \leftarrow RandDist(FCL, ProbFCL)$ {Náhodně vybere jednu třídu}

$Pop_i \leftarrow FindRandInd(Optimization, FCL_{rnd})$ {Nalezene se nového jedince ve třídě FCL_{rnd}}

end for

{Ohodnocení úvodní populace}

$State \leftarrow EvalPopulation(Pop, F, Optimization)$

{Inicializace algoritmu}

$State \leftarrow InitAlg(Options)$

{Hlavní cyklus algoritmu}

while $EndAlg = false$ **do**

$State \leftarrow DiscGeneration(State, Pop, CL, Optimization, Options)$ {Provedeni jedné iterace diskrétního algoritmu}

$State \leftarrow DiscGeneration(State, Pop, CL, Optimization, Options)$ {Provedeni jedné iterace spojitého algoritmu}

$(State, Pop) \leftarrow EvalPopulation(Pop, F, Optimization)$ {Ohodnocení nové populace}

$State \leftarrow PopulationStats(State, Pop, Optimization, Options)$ {Výpočet statistik nové populace}

$State \leftarrow PopulationPlot(State, Pop, Options)$ {Výpis statistik nové populace na konzoli}

$(EndAlg, ExitValue) \leftarrow EndCriterium(State, Pop)$ {Výpočet koncového kritéria}

$State.Generation \leftarrow State.Generation + 1$ {Posun na novou generaci}

end while

$returnExitValue$

4. Implementace

V úvodu této kapitoly bude popsáno vývojové prostředí použité pro vývoj programů a jeho nezbytných součástí. Dále bude následovat popis vstupních dat algoritmu a podrobnější popis funkcí použitých v porovnávacím schématu. Důraz bude kladen hlavně na tzv. globální funkce, které implementují navržený hlavní algoritmus 6.

4.1 Vývojové prostředí

Jako vývojové prostředí pro porovnání algoritmů a řešení katalytické optimalizační úlohy byl použit program Matlab od společnost MathWorks. Jedná se o nástroj vhodný mimo jiné pro řešení matematických úloh, úloh dobývání znalostí z dat, modelování neurononových sítí a vizualizaci dat. Tento program je distribuován jako komerční software, který je možné rozšiřovat doplňkovými moduly tzv. *toolboxy*, které vždy řeší specifickou třídu úloh. Mezi nejpoužívanější *toolboxy* patří *Gentic Algorithm and Direct Search Toolbox* (u novějších verzí *Global Optimization Toolbox*), *Neural Network*, *Statistics* a mnoho dalších. Pro vývoj našich algoritmů byl použit Matlab verze R2007b rozšířený o *Gentic Algorithm and Direct Search Toolbox* (GADS), *Optimization Toolbox*, *Statistics Toolbox* a volně dostupný *Multi-Parametric Toolbox (MPT)* [11].

4.2 Struktura programu

Aby bylo možné všechny tři zadané algoritmy porovnat bylo nutné navrhnout jakýsi jednotný rámec (viz algoritmus 6), do něhož by bylo možné vkládat jednotlivé algoritmy, jako moduly. Druhý požadavek, který plyne ze zadání úlohy je, aby onen rámec umožňoval řešit katalytickou optimalizační úlohu.

4.3 Vstupní data algoritmu

4.3.1 Struktura Optimization

Struktura `Optimization` má čtyři položky (`KnownIdentifiers`, `Constraints`, `GlobalParameters`, `ExternalFunction`), z nichž pro naše účely byly použity pouze první dvě položky:

- KnownIdentifiers - pole struktur popisujících jednotlivé identifikátory optimalizační úlohy

  Identifier - název identifikátoru.

  QuantityNumber - [číslo] číslo větší než nula, pokud je identifikátor omezen lineárním omezením, jinak rovno 0.

  KnownComponent - [0/1] rovno 1, pokud je známá komponenta, jinak rovno 0.

InProportion - [0/1] rovno 1, pokud má identifikátor definovanou proporci.

Proportion - [číslo/identifikátor] číslo, nebo identifikátor definující množství.

PreparedUsing - [0/1] rovno 1, pokud má identifikátor definovanou přípravu.

Preparation - [identifikátor] identifikátor popisující přípravu.

Evolvable - [0/1] rovno 1, pokud se identifikátor účastní evoluce, jinak rovno 0.

Count - [0/1] rovno 1, pokud má identifikátor definovanou proporci, jinak rovno 0.

ComposedOf - [0/1] rovno 1, pokud je identifikátor složen z dalších identifikátorů, jinak rovno 0.

ComposedOfRoot - [0/1] rovno 1, pokud je identifikátor kořenem stromu, jinak rovno 0.

FromAmong - [0/1] Je rovno 1, pokud se u identifikátoru vybírá jen jistá podmnožina komponent, jinak rovno 0. Týká se pouze identifikátorů, které mají `ComposedOf` = 1.

HowMany - [číslo, identifikátor] velikost vybíraných podkomponent. Týká se pouze identifikátorů, které mají `FromAmong` = 1.

Subcomponents - [seznam identifikátorů v případě, že má identifikátor položku `ComposedOf` nastavenou na 1, pak tato položka určuje identifikátory, které jsou jeho podkomponentami.

FromInterval - [0/1] rovno 1, pokud identifikátor nabývá spojitých hodnot z omezeného intervalu, jinak rovno 0.

LowerBound, UpperBound - [číslo, identifikátor] horní a dolní mez spojitého identifikátoru.

Precision - [mocnina deseti] - určuje počet desetinných míst na která se zaokrouhluje spojitý idnetifikátor.

FromIntervalParameters - [řetězec] - určuje spojité náhodné rozdělení spojitého identifikátoru. Pokud je prázdné, bere se jako defaultní hodnota rozdělení rovnoměrné.

IsPrimitive - [0/1] rovno 1, pokud je to list stromové struktury.

OneOf - [0/1] rovno 1, pokud je identifikátor diskrétní a jeho hodnota se vybíra ze seznamu hodnot z pole `Choices`.

Choices - [seznam hodnot/řetězců] viz výše.

DistributedAs - [0/1] rovno 1, pokud je identifikátor `OneOf`=1 a pravděpodobnostní rozdělení jednotlivých výběrů v Choices není rovnoměrné.

Distribution - [seznam čísel] tato položka určuje pravděpodobnostní rozdělení jednotlivých výběrů v `Choices`.

SavedIn - [0/1] rovno 1, pokud je identifikátor uložen v databázi.

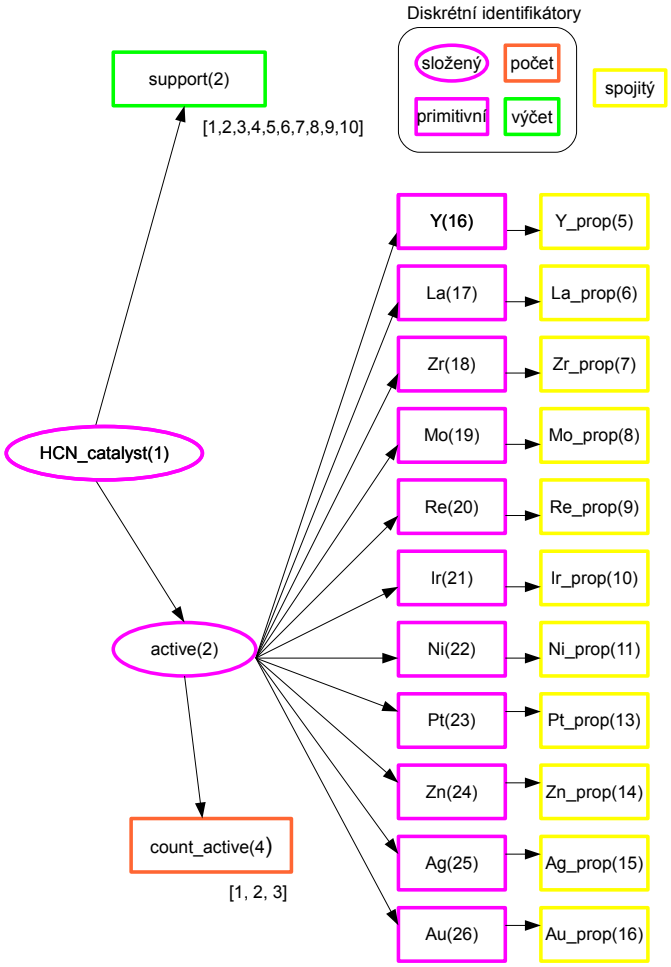
- Constraints - struktura, která popisuje lineární omezení spojitých a některých diskrétních identifikátorů.

  Aeq, Beq - soustava lineárních rovností.

  Aineq, Bineq - soustava lineárních nerovností.

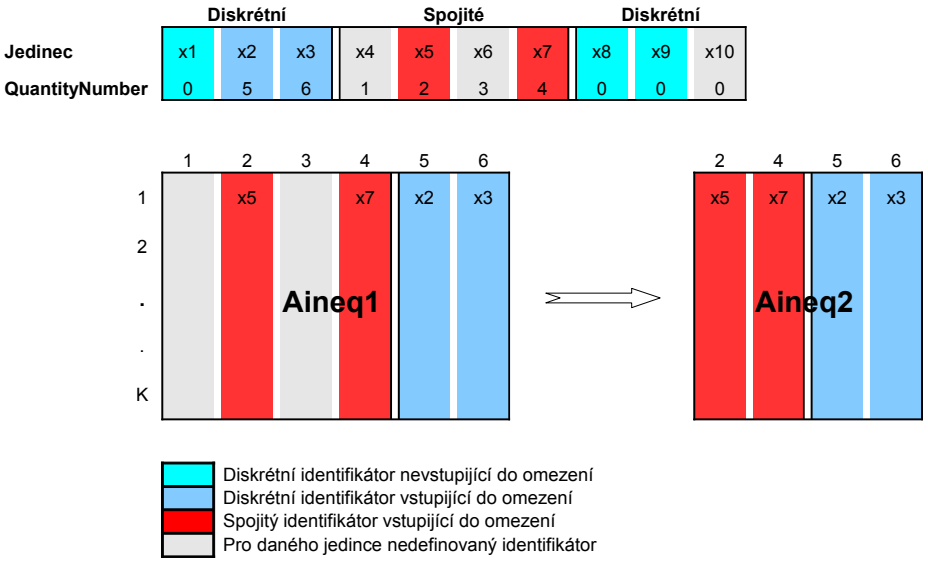
  Quantities - názvy identifikátorů, v pořadí, které koresponduje jednotlivým sloupcům výše zmíněných soustav.

  Precisions - zaokrouhlovací přesnosti jednotlivých identifikátorů.



Obrázek 4.1: Příklad stromové struktury identifikátorů (komponent) optimalizační struktury HCN_catalyst.

- `options` - options je struktura uchovávající veškeré konstantní nastavení celého algoritmu jako je velikost populace (`PopulationSize`), maximální počet generací algoritmu (`MaxGenerations`), maximální počet ohodnocení fitness funkce (`MaxFunEval`), volba spojitého resp. diskrétního algoritmu (`ContAlg` resp. `DiscAlg`) a jejich nastavení evolučních parametrů



Obrázek 4.2: Příklad redukce vstupní matice omezeni Aineq1 definované v struktůře Constraints a výběru sloupců podle identifikátorů definovaných v konkrétním jedinci.

(`ContParams` resp. `DiscParams`). Dále je možné nastavit parametry ladícího režimu `DebugMode`, `Verbosity` a `PlotInterval`. `DebugMode` je přepínač mezi ladícím a produkčním režimem běhu algoritmu. Pokud je `DebugMode` roven 1, tak jsou v pruběhu algoritmu volány některé kontrolní funkce začínající `check_*`, jenž kontrolují konzistenci algoritmu za běhu. `Verbosity` určuje úroveň ladících výpisů a nabývá čtyř hodnot: 0 - ladicí tisky jsou úplně vypnuty, 1 - ladící tisky pro produkční běh algoritmu, 2 - podrobnější ladící tisky, 3 - nejpodrobnější výpis.

- `info` - `info` je struktura předpočítaných hodnot z `Optimization` struktury. Obecně obsahuje vektorizované booleovské masky, určitých vlastností identifikátorů definovaných v `Optimization` struktuře (např.: `cont_mask` je maska všech spojitých identifikátorů)

4.4 Běh programu a popis pseudoalgoritmu

4.4.1 Konvence a základní struktura programu

Vývojové prostředí Matlab umožňuje jak objektový, tak i procedurální přístup k programování. Pro účely implementace hlavního algoritmu 6 a jednotlivých porovnávaných algoritmů byl použit druhý (procedurální) způsob. Celý program je proto členěn do funkcí, které jsou dle konvence programování v Matlabu umístěny v souboru se stejným názvem. Funkce v tomto programu se dají rozdělit na následující tři kategorie:

1. kategorie základních funkcí algoritmu jsou pojmenovány `XX_název_funkce`, kde XX je EA - obecné funkce algoritmu, GA - funkce týkající se genetic-

kého algoritmu, DE funkce týkající se diferenciální evoluce, nebo ES funkce týkající se evoluční strategie.

2. kategorie ostatních globálních funkcí.

3. kategorie lokální, nebo private funkce.

Spuštění běhu algoritmu zajišťuje základní funkce `eacompare2()`, jejímž vstupem jsou dva parametry `Optimization`, `options`. `Optimization` je vstupní optimalizační struktura, která reprezentuje reálnou katalytickou, smíšenou, optimalizační úlohu tj. popisuje formát prohledávaného prostoru a také experimentální fitness funkci. Připomeňme, že fitness funkce má experimentální charakter, a proto je její vyhodnocení v praxi velice časově a finančně náročné. Proto pro účely testování optimalizace byla využita její aproximace pomocí RBF sítí. [1]. Druhým parametrem je struktura options 4.3.1, která je inicializována funkcí `EA_create_options()`. Pokud chceme načíst RBF model fitness funkce, je nutné zavolat `load_fitness_data()`, kde se inicializují modely `final_model`, `model_type`, `glm_model`, `global_model`. Tyto modely je možné pro zeefektivnění běhu algoritmu a jeho opakování předpočítat. V takovém případě se předinicializované struktury vloží do `load_fitness_data()` jako parametr.

Funkce `eacompare2()` by se dala rozdělit do dvou částí inicializace a evoluční cyklus.

1. Inicializace - v inicializaci se provede několik klíčových kroků pro běh celého algoritmu. Pro správné vygenerování náhodné populace je nezbytné, aby všichni jedinci byli přípustní a aby jejich náhodný výběr respektoval prvděpodobnostní rozdělení definované strukturou `Optimization`. Hlavní kroky které proto proběhnou v této fázi jsou: předzpracování `Optimization` struktury [1], nalezení přípustných tříd jedinců viz 4.4.3 a na závěr se provede vygenerování náhodné populace.

2. Evoluční cyklus - cyklus během něhož se střídají dva kroky optimalizace, nejprve se provede iterace diskrétního algoritmu `EA_disc_generation()`, poté se provede iterace algoritmu spojitého `EA_cont_generation()`. Následně se ohodnotí nová populace, spočítají se statistiky pro porovnání `EA_statistics()` a vypíše se průběh algoritmu na výstup `EA_plot_pop()`. Celý cyklus běží, dokud není splněno koncové kritérium `EA_end_criterium()`. Pak algoritmus skončí a vrátí návratový stav obsahující informace o průběhu algoritmu [2]. Jako koncové kritérium je použito buď dosažení maximálního počtu generací (options.MaxGenerations), nebo ohodnocení fitness funkcí (options.MaxFunEval).

4.4.2 Lineární omezení ve struktuře Optimization

Lineární omezení je pro naší optimalizaci definováno ve struktůře Optimization.Constraints trojím způsobem:

[1] Jedná se o předzpracování položek Constraints a KnownIdentifiers

[2] konvergence skóre v průběhu generací

1. Pomocí soustavy lineárních rovnic

$$A_1 \cdot \vec{x} = \vec{b_1} \tag{4.1}$$

kde $A_1 \in R^{m_1 \times n}$

2. Pomocí soustavy lineárních nerovnic

$$A_2 \cdot \vec{x} \leq \vec{b_2} \tag{4.2}$$

kde $A_2 \in R^{m_2 \times n}$

3. Pomocí horních a dolních mezí pro jedontlivé proměnné - $\vec{lb}, \vec{ub} \in R^n$, takový že platí

$$\vec{lb} \leq \vec{x} \leq \vec{ub} \tag{4.3}$$

Výše zmíněné matice definují omezení souhrně pro všechny identifikátory vstupující do omezení [3]. Do omezení vstupují ovšem jen identifikátory, jenž jsou jedincem definovány. Ze stromového charakteru Optimization struktury vyplývá, že často se u konkrétního jedince vybírá pouze nějaká podmnožina komponent z většího počtu, a tudíž některé identifikátory nejsou vůbec definovány. Potom také příslušné sloupce omezení nejsou definovány. Trochu zjednodušeně se tedy dá říci, že diskrétní hodnoty jedince definují výběr sloupců v rovnicích 4.1 a 4.2 [4], které pak omezují spojitou část. Prohledávaný prostor úlohy je tedy poměrně heterogenní, protože je rozdělen do mnoha mnohostěnů (polytopů), jejichž dimenze $n^{'}$ závisí na konkrétním jedinci a platí $n^{'} \leq n$ a jenž je omezen odpovídajícími soustavami lineárního omezení. Diskrétní část jedince určuje výběr jednoho mnohostěnu, zatímco spojitá část definuje bod uvnitř tohoto mnohostěnu. Je nutné dodat, že né všichni jedinci, kteří mohou být vygenerováni z Optimization struktury, reprezentují neprázdný mnohostěn. Z tohoto důvodu je nezbytné nalézt ty kombinace diskrétních hodnot, které definují přípustný (neprázdný) mnohostěn. Pouze s těmito hodnotami se bude v následujícím algoritmu pracovat. Vzhledem k možnému rozsahu úlohy, kde počet všech diskrétních kombinací může vzrůst až na $\approx 10^9$, není jejich nalezení v reálném čase triviální problém. Více o nalezení v následující kapitole 4.4.3

4.4.3 Předzpracování, předpočítání tříd ekvivalence a jejich pravděpodobností

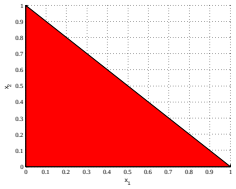
Definice1: Rozšířená soustava omezení je taková soustava $A_r \cdot \vec{x} \leq b_r$, kde A_r vznikne z A_2 doplněním o matici A_1 převedenou na soustavu nerovnic [5].

Definice2: Definujme ekvivalenci nad tělesem rozšířených matic omezení $\equiv$, kde $A_r^{'} \equiv A_r^{''}$, právě tehdy když mají stejné rozšířené matice omezení až na permutace sloupců a řádků.

[3] Jsou to identifikátory, které mají QuantityNumber > 0. Tato hodnota poté udává index sloupce, v maticích A_1, A_2, lb, ub, ke kterému daný identifikátor náleží.

[4] rovnice 4.3 se zřejmě dá snadno převézt na rovnici 4.2.

[5] Každá rovnice typu $A_{1i} \cdot \vec{x} = b_{1i}$ se převede na dvě nerovnice $A_{1i} \cdot \vec{x} \leq b_{1i}$ a $-A_{1i} \cdot \vec{x} \leq -b_{1i}$.



Obrázek 4.3: Ukázka dvoudimenzionálního mnohostěnu.

Poznámky:

- Nutno podotknout, že mnohostěny reprezentované ekvivalentními maticemi v žádném případě nemusí být vždy stejné, protože to mohou být mnohostěny v jiných souřadnicích. Pouze o nich můžeme tvrdit, že mají stejný 'tvar', tedy i stejný prostor řešení, a tudíž stačí na přípustnost, či nepřípustnost všech takových mnohostěnů otestovat pouze jednoho zástupce.

- Třídou v následujícím textu bude vždy myšlena třída ekvivalence ve smyslu předchozích definic. V programu je tato třída reprezentovaná strukturou class, která se v programu vytváří funkcí `create_class()`.

Předzpracování struktury je další krok inicializační fáze. Předzpracování probíhá ve funkci `preprocess_opt_struct()`, jejímž jediným parametrem je Optimization struktura a výstupem je `info` struktura. V této funkci se rekurzivně (prohledáváním do hloubky) projde celý strom a provedou se dva základní úkony. Za prvé to je získání redundantních údajů pro následnou efektivnější práci se strukturou. Jedná se zejména o vektorizaci masek jednotlivých vlastností a také předpočítání indexů jednotlivých identifikátorů [6]. Druhým úkolem této funkce je nalezení všech tříd ekvivalence, které reprezentují neprázdný mnohostěn a jejich pravděpodobnostního rozdělení. Nejtriviálnějším nápadem, jak takové třídy nalézt je vygenerovat všechny kombinace jedinců, k nim najít rozšířené matice omezení a vzájemně porovnat. Tento postup je velice neefektivní a na struktury čítající více než 40 identifikátorů v praxi neaplikovatelný. Proto bylo nutné zvolit jiný způsob, způsob založený na principu dynamického programování tj. z dílčích řešení a jejich agregací získat řešení celkové.

Cílem tedy není generovat všechny možné jedince, ale hledané třídy budovat postupně pří průchodu stromovou strukturou. Budeme si proto u každého identifikátoru (listu ve stromě) uchovávat informaci o tom, kolik tříd z něj a z jeho

[6] Identifikátory se v Optimization struktuře na sebe odkazují svými jmény (znakovými řetězci), což je nevýhodné pro následnou práci, protože by se při každém dereferencování odkazu muselo projít celé pole s identifikátory

podstromu vzniká. V programu se struktura, která tyto informace nese nazývá `class` (viz programátorské komentáře k funkci `create_class()`). Z toho tudíž vyplývá, že námi hledané přípustné třídy ekvivalence jsou všechny třídy kořene stromu. Ty vzniknou postupným agregováním podtříd při rekurzivním prohledávání stromu. Pro správné předávání informací o třídách z nižší úrovně stromu do vyšší je zapotřebí si uvědomit, které identifikátory zvyšují počet tříd, a které ho zachovávají. K tomuto účelu vyčleníme několik typů identifikátorů.

1. Spojitý identifikátor - ve stromové struktůře je to vždy list a jeho příspěvek do celkového počtu tříd je pouze ten, že se rozšíří matice omezení všech tříd o příslušný sloupec tohoto identifikátoru. To znamená, že všechny třídy, kteří budou zahrnovat tento identifikátor budou mít v matici omezení jeho příslušný sloupec z matice omezení. Připomeňme, že spojitý identifikátor má vždy odpovídající sloupec v maticích omezení.

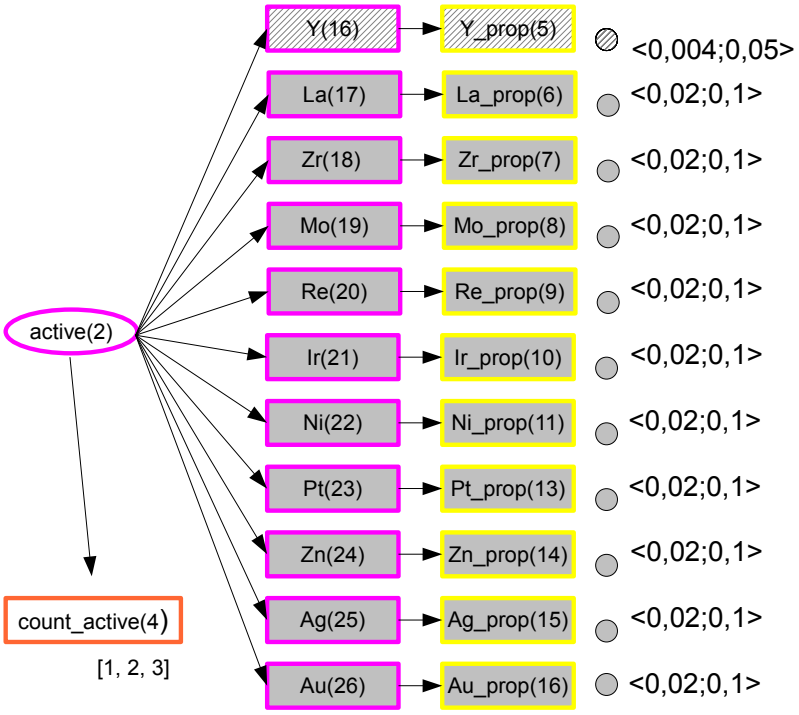
2. Identifikátor typu počet - typicky tento identifikátor určuje počet vybíraných podkomponent daného složeného (`ComposedOf`) identifikátoru a často je sám o sobě zahrnut v matici omezení. Jeho příspěvek do počtu tříd je zvyšující, protože každá volba určitého konkrétního počtu k (typicky počtu podkomponent v ComposedOf identifikátoru) mění matici omezení všech k-prvkových podmnožin podkomponent.

3. Diskrétní identifikátor v omezení - jedná se o (`OneOf`) identifikátor, jenž má v poli `Choices` numerické hodhoty. Jeho přínos do celkového počtu tříd je zvyšující a platí, že každá hodnota identifikátoru vytváří jednu třídu.

4. Diskrétní identifikátor bez omezení - jedná se o (`IsPrimitive` a `OneOf`) identifikátory. Přestože tyto identifikátory způsobují zvýšení počtu diskrétních kombinací, jejich počty tříd se jejich vlivem nezvyšují.

5. Diskrétní složené identifikátory - jedná se o (`ComposedOf`) identifikátory. V tomto případě je počet tříd, které generuje tento identifikátor dán jako kartézský součin tříd jednotlivých podkomponent. V případě, že se vybírají pouze k-prvkové podmnožiny, pak třídy každé takové podmnožiny jsou dány, jako kartézský součin tříd jejich podkomponent a celková množina tříd tohoto identifikátoru je potom sjednocení všech takových kartézských součinů. Jak bude uvedeno dále, právě na tomto místě dochází k největší 'úspoře' a k největšímu rozdílu mezi počtem tříd a počtem diskrétních kombinací. Praxe totiž ukazuje, že podkomponenty složených identifikátorů mají často ekvivalentní [7] matice omezení. Je tedy zřejmé, že stejné k-prvkové kombinace ekvivaletních podkomponent ve vybíraných k-prvkových podmnožinách bude generovat ekvivalentní matice omezení, a tedy všechny takové kombinace budou patřit do stejné třídy ekvivalence. Pro lepší ilustraci této problematiky si můžeme všechny podkomponenty s ekvivalentní maticí obarvit barvami $b = (b_1, \ldots, b_t)$, kde t je počet neekvivalentních podkomponent. Potom do jedné třídy ekvivalence spadnou všechny stejně barevné a stejně početné kombinace barev 4.4.

[7] Ve smyslu **Definice2**.



CID	Barva podtříd	N	D	Prob
1	◍	1	1	1/231
2	●	10	1	10/231
3	◍ ●	10	1	10/231
4	● ●	45	1	45/231
5	◍ ● ●	45	1	45/231
6	● ● ●	120	1	120/231

Obrázek 4.4: Ukázka dělení tříd u podkomponent složené komponenty

Z předchozího dělení vyplývá, že k redukci počtu tříd vůči diskrétním kombinacím dochází ve dvou případech 4) a 5). Ve výsledku může být na celé `Optimization` struktuře rozdíl mezi počtem diskrétních kombinací a počtem tříd ekvivalence několik řádů. V krajních případech ovšem může nastat situace, kdy bude `Optimization` struktura natolik nesourodá, že počet tříd bude asymptoticky roven počtu kombinací, jinými slovy každá třída bude obsahovat jeden prvek (jednu diskrétní kombinaci). Jak jsem ale otestoval na několika reálných `Optimization` strukturách, tak tomuto nedochází, ale naopak je redukce počtu tříd markantní.

4.4.4 Struktura class

Struktura `class` má dva druhy položek. Prvním jsou tzv. *globální položky*, jenž jsou propagovány při průchodu stromu od listů až ke kořeni. Druhým typem jsou položky *lokální*, které mají význam pouze pro jeden indentifikátor (nejčastěji složený - ComposedOf, nebo diskrétní výběr - `OneOf`). V následujícím seznamu je výčet položek struktury `class`:

cid - id třídy číslované od 1 do počtu tříd n_{cl}.. Toto číslo má klíčový význam při generování náhodné populace, protože určuje jakési úplné uspořádání na třídách. Toto uspořádání je definované průchodem algoritmu `preprocess_opt_struct()` stromem.

prob - pravděpodobnost třídy normovaná na interval $\langle 0, 1\rangle$, která je závislá jak na počtu diskrétních kombinací ve třídě, tak na pravděpodobnostním rozdělení vybraných diskrétních hodnot (4.4.5).

idxs - lokální informace. Položka obsahuje matici kombinací indexů podkomponent patřících do této třídy.

quant_idxs - indexy sloupců v matici omezení všech identifikátorů, které daná třída využívá.

n - počet všech diskrétních kombinací, které jsou ve třídě obsaženy.

d - pravděpodobnostní distribuce třídy. Agreguje v sobě informaci o pravděpodobnostních distribucích vybraných diskrétních hodnot.

eq_group - lokální informace, která se objevuje pouze u složených (`ComposedOf`) identifikátorů. Barevná kombinace ekvivalentních podkomponent

type - typy třídy:

- *cont* - tímto příznakem je označena třída pouze u spojitých identifikátorů (FromInterval) a tyto třídy slouží pouze k propagaci údajů o matici omezení.
- *count* - tímto příznakem je označena třída pouze u identifikátorů počtu (Count) a tyto třídy slouží pouze k propagaci údajů o matici omezení.
- *disc* - standardní diskrétní třída všech ostatních identifikátorů nespadajících do žádné z předchozích možností.

Aineq - vybrané sloupce levé strana soustavy nerovností respektující výběr konkrétních diskrétních hodnot.

Aeq - vybrané sloupce levé strana soustavy nerovností respektující výběr konkrétních diskrétních hodnot.

Bineq - pravá strana soustavy nerovností upravená po dosazení konkrétních diskrétních hodnot.

Beq - pravá strana soustavy nerovností upravená po dosazení konkrétních diskrétních hodnot.

4.4.5 Výpočet pravděpodobností, cid a matic omezení u tříd

Značení: Označme nějaký složený [8] K, který má podkomponenty $K_1, \ldots, K_r$ viz obrázek 4.4. Dále jsou definovány počty vybíraných podkomponent z identifikátoru `Count` $k_1, \ldots, k_s$, kde $\max_j(k_j) \leq r$. Každá podkomponenta K_i reprezentuje množinu tříd $C_1 = (C_{i1}, \ldots, C_{it_i})$, kde t_i je počet tříd podkomponenty K_i. Dále označíme $Ccid_j$, Cn_j, Cd_j, Cp_j a Cq_j hodnoty položek `cid`, `n`, `d`, `prob` a `quant_idxs` třídy C_j.

Pro výpočet pravděpodobností (`class.prob`), počtu kombinací (`class.n`), distribucí (`class.d`) a omezení (`class.quant_idxs`) je nutné identifikátory opět brát rozdělené dle předhozí kapitoly 4.4.3.

1. Spojitý identifikátor (`FromInterval`) - vznikne jedna třída typu *cont* s těmito vlastnostmi: $Ccid = 1, Cn = 1, Cd = [], Cp = [], Cq = QuantityNumber$,

2. Identifikátor typu počet (`Count`) - vznikne s-prvkové pole tříd typu count s těmito vlastnostmi: $Ccid_{i1} = i$, $Cn_{i1} = 1$, $Cd_{i1} = Distribution_i$, $Cp_{i1} = \frac{Distribution_i}{\sum_{j=1}^{s} Distribution_j}$, $Cq_{i1} = QuantityNumber$ a také proběhne dosazení příslušné diskrétní hodnoty do matic omezení, čímž se upraví levá strana rovnic. $i \in \{1, \ldots, s\}$. Tato třída je specifická tím, že pouze propaguje informaci o omezení a distribuci hodnot identifikátoru typu `Count`. Není tedy součástí kartézského součinu tříd jednotlivých podkomponent, nýbrž pouze propaguje informaci o omezeních a pravděpodobnostních rozděleních všem třídám, které vznikají karetézským součinem tříd podkomponent. Pokud se jako hodnota Count zvolí například hodnota k_v, pak všechny podkomponenty k_v-prvkové kartézské součiny získají onu informaci o omezeních a pravděpodobnostech.

3. Diskrétní identifikátor (OneOf) s omezením - vznikne s-prvkové pole tříd s těmito vlastnostmi: $Ccid_{i1} = i$, $Cn_{i1} = 1$, $Cd_{i1} = Distribution_i$, $Cp_{i1} = \frac{Distribution_i}{\sum_{j=1}^{s} Distribution_j}$, $Cq_{i1} = QuantityNumber$ a také proběhne dosazení příslušné diskrétní hodnoty do matic omezení, čímž se upraví levá strana rovnic. $i \in \{1, \ldots, s\}$.

[8] identifikátor Složený identifikátor je takový, který má nastaven položku ComposedOf = 1.

4. Diskrétní identifikátor (OneOf) bez omezení - vznikne jedna třída s těmito vlastnostmi: $Ccid = 1$, $Cn = s$, $Cd = 1$, $Cp = Cn \cdot Cd$ a $Cq = []$, kde s je počet možných hodnot identifikátoru.

5. Diskrétní složené identifikátory - jsou to identifikátory, které způsobují větvení stromu na podkomponenty. Z tohoto důvodu zde nedochází k vytvoření nových tříd, nýbrž pouze k rozšíření počtu kartézským součinem tříd příslušných k-tic podkomponent. Výsledkem kartézského součinu k-tice podkomponent $(K_{r_1}, K_{r_2}, \ldots, K_{r_k})$ s již nalezenými třídami $(C_{r_1 t_{r_1}}, C_{r_2 t_{r_2}}, \ldots, C_{r_k t_{r_k}})$, je množina tříd velikosti $\prod_{j=1}^{k} r_j$. Vlastnosti jedné z této množiny tříd vzniklé výběrem vždy první třídy z každé podkomponenty [9] jsou následující: $Ccid = \sum_{i=1}^{k}(Ccid_{r_i} \prod_{j=i+1}^{k} t_{r_j}$, $Cn = \prod_{j=1}^{k} Cn_{r_j 1}$, $Cd = \prod_{j=1}^{k} Cd_{r_j 1}$, $Cp = \prod_{j=1}^{k} Cn_{r_j 1} \cdot Cn_{r_j 1}$, $Cq = (Cq_{r_j 1})_{j=1}^{k}$. Mezi třídy vzniklé kartézským součinem je nutné také propagovat informaci od tříd identifikátoru Count, existuje-li nějaký. Výsledné třídy všech k-tic se na závěr sjednotí do jedné výsledné množiny tříd a přepočítá se `cid` položka tak, aby to byla posloupnost začínající od 1. Pro budoucí inicializaci diskrétní části populace je nezbytné zachovat pevné pořadí sjednocování tříd proto, aby existovala bijekce mezi `cid` identifikátorem a jednoznačným průchodem optimalizačního stromu. Na základě konkrétní hodnoty `cid` je potom možné se v každém uzlu stromu při jeho prohledávání jednoznačně nalézt odpovídající třídu, tedy konkrétní diskrétní kombinaci hodnot. Pokud daná třída obsahuje více, než jednu kombinaci hodnot, potom se jedna z nich vybere náhodně buď s rovnoměrným rozdělením, nebo s explicitní distribucí specifikovanou pro komponentu.

4.4.6 Hledání neprázdných, přípustných tříd ekvivalence

V předzpracování celé `Optimization` struktury proběhla jak inicializace nějakých datových struktur, tak i nalezení všech tříd diskrétních jedinců, majících ekvivalentní matice omezení. V následujícím kroku fáze inicializace proběhne redukce tříd ekvivalence pouze na ty, které reprezentují neprázdný mnohostěn. Za neprázdný mnohostěn budeme považovat jen ten, jehož velikost je větší, než minimální zaokrouhlovací přesnost jednotlivých identifikátorů (položka `Precision`). Velikostí mnohostěnu je v tomto případě myšlen průměr *Chebyshevovy koule* tj. maximální vepsané n-dimenzionální koule obsažené v mnohostěnu. K tomu, abychom byli schopni ověřit prázdnost, či neprázdnost mnohostěnů, budeme muset řešit příslušné lineární soustavy rovnic a nerovnic . Hledání těchto tříd je v našem programu implementováno ve funkci `find_feasible_classes()`.

Předpokládejme, že matice soustavy rovnic resp. nerovnic pro jednu konkrétní třídu jsou $A' = \{A'_{i_1}, \ldots, A'_{i_k}\}, \vec{b'}$ resp. $A'' = \{A''_{i_1}, \ldots, A''_{i_k}\}, \vec{b''}$, kde $i_1, \ldots, i_k$ jsou vybrané sloupce omezení definovaných identifikátorů a $\vec{b'}$ resp. $\vec{b''}$ je pravá strana soustav po dosazení konkrétních diskrétních hodnot. Na tomto místě stojí za povšimnutí, že hodnoty diskrétních identifikátorů jsou již dosazeny do soustavy a tudíž jejich příspěvky se promítnou do pravé strany $\vec{b''}$, $\vec{b''}$ a příslušné sloupce budou eliminovány. Proto vstupem ověřování prázdnosti, či neprázdnosti

[9] Bez újmy na obecnosti – pro zjednodušení formalismu.

mnohostěnů jsou pouze sloupce odpovídající spojitým identifikátorům. Než přistoupíme k samotnému postupu řešení soustav, uvědomme si, že řešením soustavy nerovnic může být:

1. Obecný podprostor omezený nadrovinami
2. Konvexní mnohostěn
3. Prázdná množina

Řešení soustavy rovnic je podprostor dimenze $n - hodnost(A'|\vec{b'})$, kde n je počet sloupců matice A'. Proto intuitivně mají-li obě soustavy platit současně, je jasné, že řešením (pokud nějaké existuje) bude mnohostěn zredukovaný o hodnost matice $A'|\vec{b'}$. Připomeňme ještě, že mnohostěny v katalytické optimalizaci jsou vždy ve všech dimenzích omezené, protože u spojitých identifikátorů jsou vždy definovány horní a dolní mez. Postup pro ověření neprázdnosti mnohostěnů je následující.

Nejprve jsou soustavy otestovány, jestli příslušné dosazené diskrétní hodnoty splňují soustavu nerovnic. Nepřípustnost odhalují řádky typu $\{0, 0, \ldots, 0\} \leq z$, kde $z < 0$. V dalším kroku se určí báze soustavy rovnic. Získáme tak sloupce *bázové – lineárně nezávislé* a sloupce *lineárně závislé*. Postupným vyjádeřním nezávislých proměnných pomocí závislých a jejich následnou substitucí do matice nerovnic dosáhneme toho, že se zredukuje dimenze prostoru řešení soustavy nerovnic o $hodnost(A'|\vec{b'})$. V posledním kroku se zredukovaná soustava nerovností otestuje na neprázdnost. Tímto testem se myslí i porovnání průměru již zmíněné vepsané Čebyševovy koule mnohostěnu s minimální přesností jednotlivých identifikátorů. Pokud je průměr příliš malý, mnohostěn je považován za prázdný.

4.4.7 Inicializace náhodné populace

Posledním krokem inicializace je samotné generování náhodné a přípustné populace `rand_pop()`. K nalezení náhodné populace budeme vycházet z již získaného seznamu neprázdných tříd označme ho FCL [10]. Nyní přijde ke slovu identifikátor třídy položka `cid`. Toto políčko určuje vzájemně jednoznačnou korespondenci mezi třídou ekvivalence a jednoznačným průchodem stromu. Mohli bychom tedy říci, že třídy jsou uspřádány úlným upořádáním definovaným jednoznačným průchodem stromu do hloubky. Díky tomu je možné z konkrétního čísla `cid` nalézt pro každý uzel ve stromu Optimization rozhodnutí, kterými větvemi se vydat dál do nižších vrstev stromu směrem k listům. Tímto postupem jsme schopni zreplikovat komponenty zahrnuté do vybrané třídy. U některých diskrétních identifikátorů ovšem v předzpracování `preprocess_opt_struct()` nedošlo k inicializaci hodnoty. Jedná se na příklad o identifikátory typu OneOf bez omezení (kategorie 4) v kapitole 4.4.3). Tyto identifikátory nenavýšily počet kombinací v jedné třídě, protože žádným způsobem neovlivňují matice omezení, a tedy ani ekvivalenci mezi diskrétními jedinci. V tomto kroku generování kompletní populace je nutné náhodně doinicializovat i tyto diskrétní hodnoty. Pro nalezení náhodné populace slouží tento algoritmus 7.

[10] FCL - Feasible CLasses

Algorithm 7 Nalezení náhodné populace

Require: N - velikost nové populace, FCL - seznam neprázdných tříd, $ProbFCL$- pravděpodobnosti *Optimization* - optimalizační struktura

{Náhodná inicializace populace funkce.}

for $i = 1$ to N **do**

$FCL_{rnd} \leftarrow RandDist(FCL, ProbFCL)$ {Náhodně vybere jednu třídu}

$Pop_i \leftarrow FindRandInd(Optimization, FCL_{rnd})$ {Nalezene se nového jedince ve třídě FCL_{rnd}}

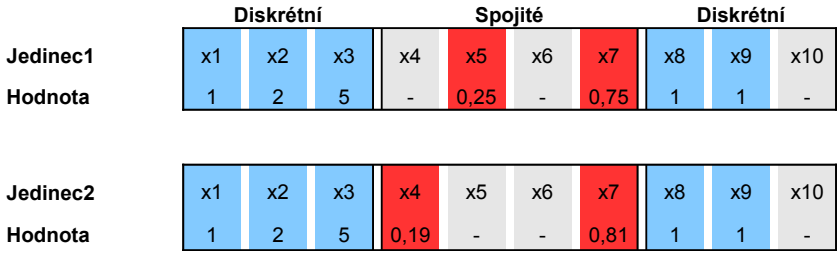
end for

return Pop

Nejprve se funkcí `RandDist()` algoritmu náhodně vybere jedna třída FCL_{rnd} reprezentující neprázdný mnohostěn. Tato třída může reprezentovat více diskrétních jedinců. Ve funkci `FindRandInd()` se potom pomocí `cid` a průchodem `Optimization` struktury nalezne kompletní diskrétní jedinec. Na jeho základě se potě náhodně inicializují spojité hodnoty tak, aby splňovaly lineární omezení příslušné třídy, do které jedinec spadá. Náhodná inicializace spojitých přípustných hodnot probíhá ve dvou krocích. První je nalezení náhodného bodu uvnitř zredukovaného mnohostěnu tj. inicializují se ty proměnné, které odpovídají lineárně závislým sloupcům při redukci omezení viz kapitola 4.4.6. Ve druhém kroku se z vazeb definovaných v soustavě rovností dopočítají zbývající lineárně nezávislé souřadnice.

4.4.8 Evoluční cyklus

Druhou částí navrženého algoritmu je *evoluční krok*. Jeho základem je tzv. *evoluční cyklus* zajišťuje přechod z jedné generace do druhé, dokud není splněno koncové kritérium. V našem případě to je buď dosažení maximálního počtu generací, nebo ohodnocení fitness funkce. Obojí je možné nadefinovat ve struktuře `options` v položkách `MaxGenerations` a `MaxFunEval`. tělem evolučního cyklu je periodické opakování diskrétního `EA_disc_generation()` a spojitého kroku optimalizace `EA_cont_generation()`. Zapojením obou těchto kroků se naplní podmínka zadání, kterou je smíšená optimalizace. V diskrétním kroku je v našem případě řešen modifikovaným diskrétním algoritmem 3.1. Následuje spojitý krok optimalizace, jenž je poněkud složitější, než krok diskrétní.



Obrázek 4.5: Ukázka dvou jedinců, kteří nepatří do stejné subpopulace, protože mají definované jiné spojité souřadnice, a tudíž náleží do jiného mnohostěnu.

Uvědomme si, že pro účely spojitého kroku není možné aplikovat jednotlivé spojité algoritmy přímo na celou populaci vzniklou v předešlém kroku. V populaci jsou různí diskrétní jedinci (mnohostěny), kteří mají často odličné dimenze, a tudíž spoití jedinci (body uvnitř mnohostěnů) jsou vzájemně nekompatibilní viz obrázek 4.5 [11]. Vstupem spojitého algoritmu jsou proto tzv. *subpopulace* jedinců patřících do stejného mnohostěnu. Jedinci patřící do jedné subpopulace, jsou tedy body uvnitř jednoho mnohostěnu. Jinými slovy v jedné subpopulaci jsou jedinci mající stejnou diskrétní část. Před samotným spojitým krokem je tedy populace rozdělena na *subpopulace*, na něž se následně aplikuje spojitý krok. Stejně jako krok diskrétní i krok spojitý musí garantovat zachování základních podmínek naší smíšené optimalizace.

[11] Nekompatibilitou se zde myslí odlišnost dimenzí, nebo odlišnost souřadnic v případě shody dimenzí.

5. Výsledky testů

Tato předposlední kapitola se bude věnovat samotnému testování a srovnávání jednotlivých algoritmů. V první části bude testována efektivita zpracování `Optimization` struktury. Pozornost bude zejména zaměřena na efektivitu výpočtu tříd ekvivalentních matic omezení. Druhá část této kapitoly se zaměří na samotné experimenty v porovnání evolučních algoritmů na úloze *katalytické optimalizace*.

5.1 Efektivita předzpracování úloh

Jako dílčí, nicméně ne zanedbatelný výsledek této práce, považuji přístup k nalezení neprázdných mnohostěnů, který je klíčem k řešení celé úlohy. Jeho neefektivní varianta by vedla k abnormální paměťové a časové složitosti v případě rozsáhlejších optimalizačních struktur. Neefektivní varianta této podúlohy by spočívala v generování všech existujících diskrétních jedinců. Ke všem takovým jedincům by se následně musely vytvořit matice lineárních omezení, které by potom byly mezi sebou porovnány na shodu prvků. Lépe řečeno matice diskrétních jedinců by byly porovnány s existujícími třídami. Pokud bychom pro aktuálního diskrétního jedince našli třídu, se kterou je ekvivalentní, potom by se do ní přidal. Pokud bychom takovou třídu nenašli, pak by se z tohoto jedince stal reprezentant nové diskrétní třídy. Tady je nutné podotknout, že přidání jedince do některé z existujících tříd platí pouze v případě, že jeho matice omezení reprezentuje neprázdný mnohostěn. Tímto způsobem získáme pole přípustných tříd ekvivalence FCL viz kapitola 4.4.6.

Máme tady dva přístupy pro nalezení přípustných tříd ekvivalence FCL:

1. V předzpracování stromové struktury
2. Generováním všech diskrétních hodnot

Prvním faktorem, který je možné měřit a který je u obou přístupů stejný je úspora testů na neprázdnost mnohostěnu [1] seskupením diskrétních jedinců do tříd ekvivalence. Výsledky pro některé konkrétní optimalizační struktury (`HCN_catalyst`, `CO2Giant_catalyst`, `EthyleneBenzen_catalyst`, `Cats3part`) jsou v tabulce 5.1.

[1] řešení soustavy lineárních rovnic, nerovnic

Optimalizační struktura	# d. jedinců	# tříd	# neprázdných tříd	# d. j. / # tříd
HCN_catalyst	2931	3	2	1732,5
CO2Giant_catalyst	4536	72	48	94,5
EthyleneBenzen_catalyst	6138000	756	118	52017,0
Cats3part	874848240	720	78	11216003,1

Tabulka 5.1: Tabulka porvnávající úsporu testů na neprázdnost mnohostěnu

5.2 Testování evolučních alogritmů

Pro testování navržených Evolučních algoritmů byly použity následující dva scénáře. Pro adekvátnost výsledků byly v jednom scénáři nastaveny stejné vstupní evoluční parametry (ve struktuře `options`) pro všechny porovnávané evoluční algoritmy. Součástí nastavení je kromě výběru evolučních algoritmů a jejich parametrů (definovaných v položkách `ContParams`, `DiscParams` struktury `options`) také volba velikosti populace (`PopulationSize`) a limity nastavení koncového kritéria (`MaxGenerations`, `MaxFunEvals`).

5.2.1 1.testovací scénář

První testovací scénář proběhl na všech třech spojitých algoritmech – Gentetický alg., Diferenciální evoluce a Evoluční strategie. Jako diskrétní algoritmus byl použit Genetický algoritmus. Každý algoritmus byl spuštěn desetkrát a jako výsledek byl brán průměr těchto deseti běhů. Nastavení vstupních kontrolních parametrů bylo následující (`options`):

Globální kontrolní parametry:

PopulationSize = 50

MaxGenerations = 100

MaxFunEvals = 200000

Kontrolní parametry pro *Genetický algoritmus v diskrétním prostoru*:

EliteCount = 2 - počet nejlepších jedinců postupujících do další generace

CrossoverFraction = 0.5 - poměr mezi křízenými a mutovanými jedinci.

Kontrolní parametry pro *Genetický algoritmus ve spojitém prostoru*:

EliteFraction = 0.1 - poměr nejlepších jedinců, kteří postoupí do další generace vůči velikosti subpopulace.

CrossoverFraction = 0.6 - poměr mezi křízenými a mutovanými jedinci.

SelectionFcn = ruletová selekce

CrossoverFcn = aritmetické křížení (funkce v Matlabu `crossoveraritmetic()`)

MutationFcn = mutace přípustných jedinců (funkce v Matlabu `mutationadaptfeasible()`)

Kontrolní parametry pro *Diferenciální evoluci*:

Strategy = 1 - strategie RAND.

CR = 0.8 - pravděpodobnost křížení.

F = 0.5 - diferenciální konstanta.

MinPopulationSize = 3 - minimální velkost vstupní subpopulace.

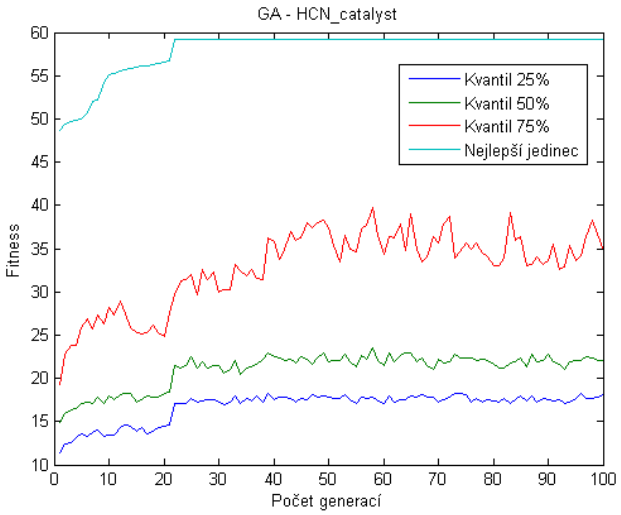
Kontrolní parametry pro algoritmus *Evoluční strategie s adaptací kovarianční matice a dynamickým vytvářením nik*:

Q = 5 - odhadovaný počet nik

Rho = 0.05 - minimální vzdálenost mezi vrcholy

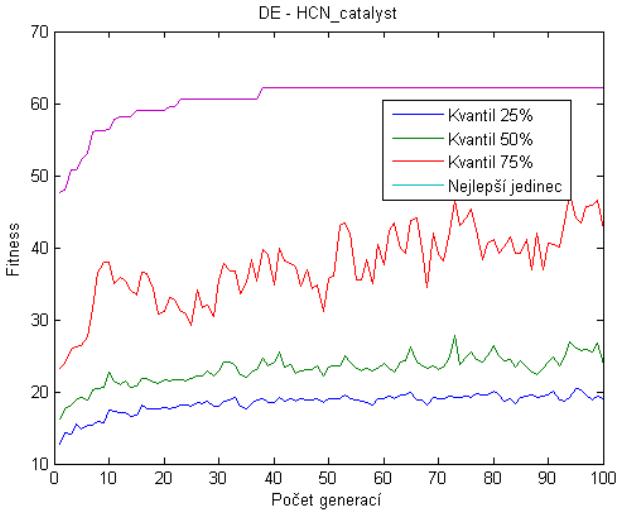
Lambda = 1 - perioda reinicializace nevrcholových jedinců

Výsledky testů jsou zobrazeny na obrázcích 5.1, 5.2, 5.3, 5.4

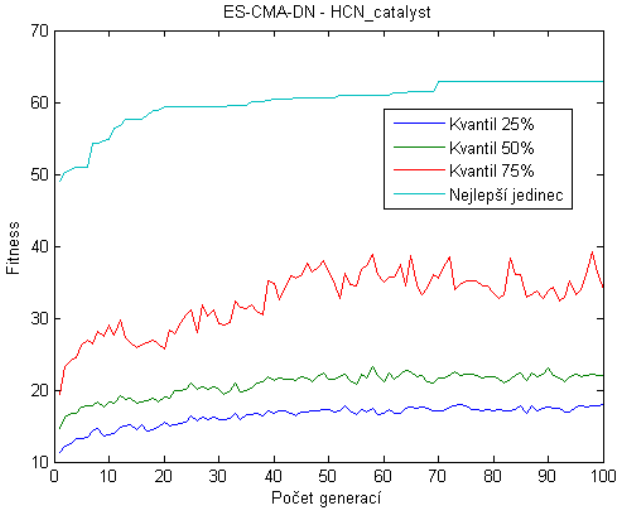


Obrázek 5.1: Kvantilové rozpětí 10-ti běhů algoritmu GA a vývoj nejlepšího jedince

Z jednotlivých grafů vyplývá, že při této konfiguraci kontrolních parametrů jednotlivých algoritmů se jako nejlepší jeví algoritmus ES-CMA-DN, který nalezl nejlepší maximum (62.8615). Při studiu průběhu jednotlivých algoritmů stojí za povšimnutí vývoj kvantilového rozpětí. Z grafů 5.1, 5.2 a 5.3 vyplývá, že kvantilové rozpětí mezi 25%. kvantilem a 50%.kvantilem zůstává přibližně konstantní, zatímco rozdíl mezi 50%. kvantilem a 75%.kvantilem se v průběhu algoritmů zvětšuje. Tento fakt je zapříčiněn poměrně vysokým podílem mutovaných jedinců (0.5) v diskrétním kroku Genetického algoritmu. Takto vysoká míra náhodně generovaných jedinců snižuje pravděpodobnost uvíznutí algoritmů v lokálním minimu. Předchozí zkušební testy totiž ukázaly, že právě volba tohoto parametru výrazně ovlivnila konvergenci algoritmu.



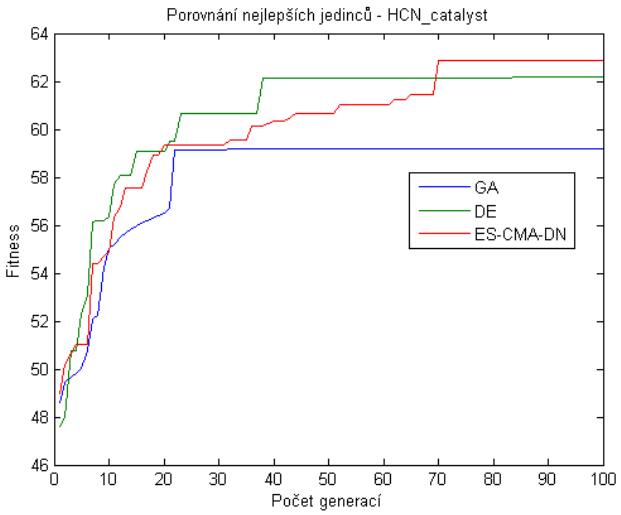
Obrázek 5.2: Kvantilové rozpětí 10-ti běhů algoritmu DE a vývoj nejlepšího jedince



Obrázek 5.3: Kvantilové rozpětí 10-ti běhů algoritmu ES-CMA-DN a vývoj nejlepšího jedince

Algoritmus	Generace nalezení nejlepšího jedince	Nejlepší jedinec
GA	21	59.1643
DE	84	62.1936
ES-CMA-DN	70	62.8615

Tabulka 5.2: Tabulka porovnávající nalezené nejlepší řešení jednotlivých algoritmů



Obrázek 5.4: Srovnání vývvoje nejlepších jedinců při testování jednotlivých algoritmů

5.2.2 2.testovací scénář

Druhý testovací scénář proběhl opět na všech třech spojitých algoritmech – GA, DE a ES-CMA-DN. Jako diskrétní algoritmus byl použit Genetický algoritmus. I v tomto případě byl každý algoritmus spuštěn desetkrát. Optimalizační strukturou byla nyní struktura `CO2Giant`. Nastavení vstupních kontrolních parametrů bylo následující (`options`):

Globální kontrolní parametry:

PopulationSize = 50

MaxGenerations = 100

MaxFunEvals = 200000

Kontrolní parametry pro *Genetický algoritmus v diskrétním prostoru*:

EliteCount = 4

CrossoverFraction = 0.6 - poměr mezi křížených a mutovaných jedinci.

Kontrolní parametry pro *Genetický algoritmus ve spojitém prostoru*:

EliteFraction = 0.05

CrossoverFraction = 0.7

SelectionFcn = ruletová selekce

CrossoverFcn = aritmetické křížení (funkce v Matlabu `crossoveraritmetic()`)

MutationFcn = mutace přípustných jedinců (funkce v Matlabu `mutationadaptfeasible()`)

Kontrolní parametry pro *Diferenciální evoluci*:

Strategy = 1 - strategie RAND

CR = 0.7 - pravděpodobnost křížení

F = 0.05 - diferenciální konstanta

MinPopulationSize = 3 - minimální velkost vstupní subpopulace.

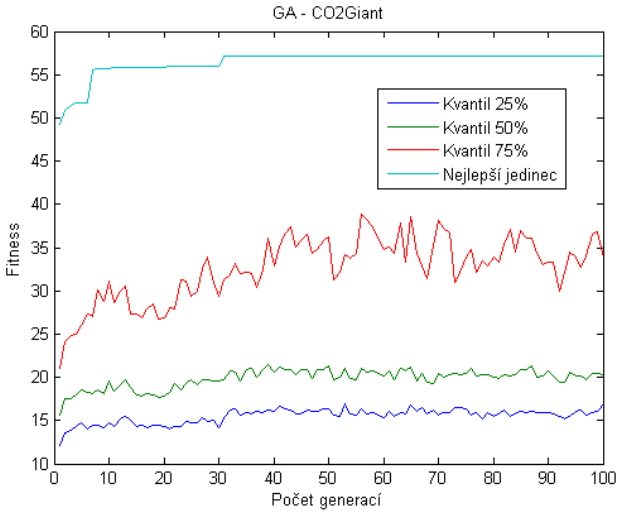
Kontrolní parametry pro algoritmus *Evoluční strategie s adaptací kovarianční matice a dynamickým vytvářením nik*:

Q = 8 - odhadovaný počet nik

Rho = 0.08 - minimální vzdálenost mezi vrcholy

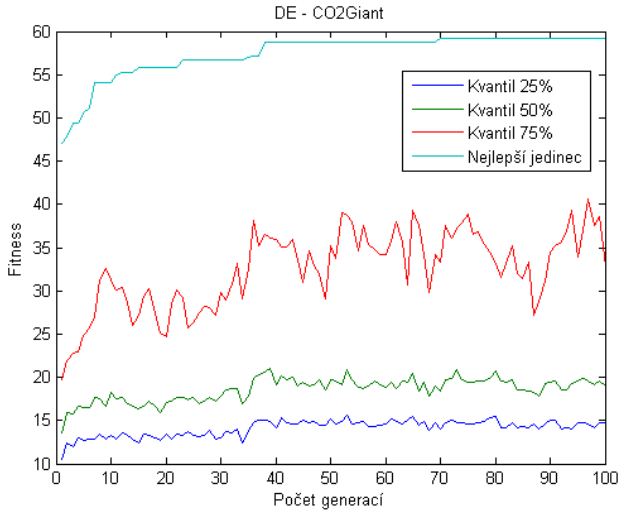
Lambda = 1 - perioda reinicializace nevrcholových jedinců

Výsledky testů jsou zobrazeny na obrázcích 5.5, 5.6, 5.7, 5.8

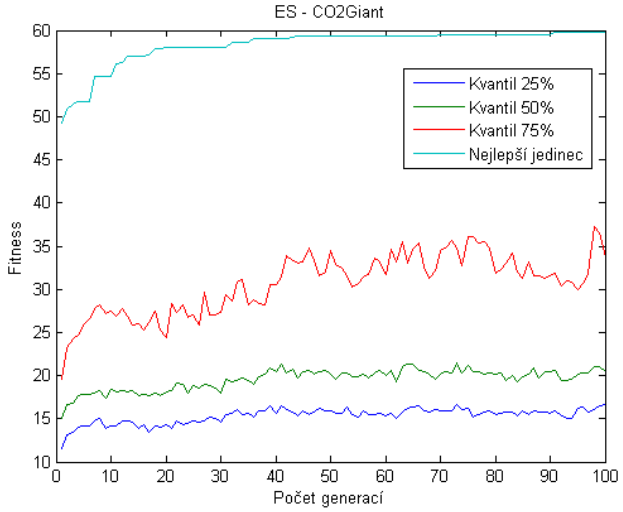


Obrázek 5.5: Kvantilové rozpětí 10-ti běhů algoritmu GA a vývoj nejlepšího jedince

Z jednotlivých grafů vyplývá, že i při této konfiguraci kontrolních parametrů jednotlivých algoritmů pří testování optimalizační struktury `CO2Giant` dosahuje nejlepšího výsledku algoritmus ES-CMA-DN, který nalezl nejlepší maximum fitness funkce s hodnotou 59.8141. Tabulka 5.2.2 ukazuje hlavní ukazatele výsledku jednotlivých algoritmů.



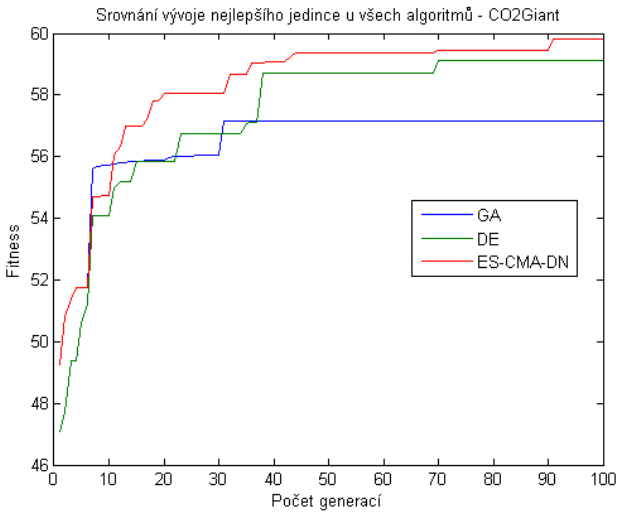
Obrázek 5.6: Kvantilové rozpětí 10-ti běhů algoritmu DE a vývoj nejlepšího jedince



Obrázek 5.7: Kvantilové rozpětí 10-ti běhů algoritmu ES-CMA-DN a vývoj nejlepšího jedince

Algoritmus	Generace nalezení nejlepšího jedince	Nejlepší jedinec
GA	73	57.1643
DE	70	59.1304
ES-CMA-DN	92	59.8141

Tabulka 5.3: Tabulka porovnávající nalezené nejlepší řešení jednotlivých algoritmů



Obrázek 5.8: Srovnání vývvoje nejlepších jedinců při testování jednotlivých algoritmů

6. Závěr

Jak je uvedeno v kapitole 1.1, cílem této práce je studie a porovnání základních typů evolučních algoritmů. Algoritmy měly být aplikovatelné na úlohu smíšené optimalizace s lineárním omezením a dále pak testovány jak na jedné reálné, tak i na testovací optimalizační úloze. Podívejme se tedy jak byly tyto cíle naplněny a jaké jsou tedy výsledky práce.

V této práci se podařilo navrhnout a implementovat jednotný algoritmus 6 pro řešení smíšené optimalizace s lineárním omezením. Do tohoto algoritmu je možné jednotlivé evoluční algoritmy přidávat jako moduly a různě navzájem kombinovat. Fakt, že jsou tyto moduly zahrnuty v jednotném rámci umožňuje následné relevantní porovnání jednoho vůči ostatním. Pro účely této práce jsou implementovány tyto 4 moduly (algoritmy): Genetický algoritmus pro spojitý i diskrétní prostor, Diferenciální evoluce pro spojitý prostor, Evoluční strategie s adaptací kovarianční matice a dynamickým vytvářením nik. Srovnáním těchto algoritmů na reálné katalystické úloze vyšel jako nejefektivnější algoritmus ES-CMA-DN, který v obou testovaných případech dosahoval nejlepších výsledků. Navzdory těmto výsledkům je možné, že při jiném nastavení kontrolních parametrů a jiné volbě evolučních operátorů by by výsledky dopadly odlišně.

Nevýhodou navrženého algoritmu je příliš velká provázanost právě s testovanou katalytickou optimalizační úlohou, které výrazně zhoršuje aplikovatelnost algoritmu na jiné testovací funkce smíšené optimalizace. Právě kvůli tomuto nedostatku nebyl daný algoritmus na těchto funkcích testován. Pro to, aby se tak mohlo učinit, by bylo nutné pro příslušnou testovací funkce vytvořit jakousi fiktivní `Optimization` strukturu, která by reflektovala vstupní formát funkce. Kromě toho by byly nutné další úpravy hlavního algoritmu.

Vzhledem k nedostatku času v závěru psaní této práce a vzhledem k velké náročnosti při zpracování samotné katalytické optimalizační úlohy, nebyly v této práci implementovány moduly z kategorie algoritmů Estimation of Distribution Algorithms (EDA) jak pro spojitou, tak i diskrétní část.

Přes některé nedostatky, považují výsledky této práce za uspokojivé. Navržené algoritmy umožňují další vývoj a rozšiřování o nové moduly evolučních algoritmů jak pro spojitý, tak i diskrétní prohledávací prostor.

Seznam použité literatury

[1] Bajer, Lukáš: *Urychlení evolučních algoritmů pomocí směsí rozdělení pravděpodobnosti*. Diplomová práce. MFF UK Praha, 2009.

[2] Volpe, Anthony F., Holeňa, Martin: *High-throughput screening in chemical catalysis, kapitola 6.*. Villey-VCH 2004, ISBN: 3-527-30814-8

[3] Guerrero José Luis, García Jesús, Martí Luis, Molina José Manuel, Berlanga Antonio: *A Stopping Criterion Based on Kalman Estimation Techniques with Several Progress Indicators*.

[4] Fogel, L. J., Owens, A. J. a Walsh, M. J.: *Artificial intelligence through simulated evolution*. John Wiley and Sons, 1996.

[5] Mathew, Tom V.: *Genetic Algorithm*. Indian Institute of Technology Bombay, Mumbai-400076.

[6] Price, Kenneth V. , Storn, Rainer: *Differential Evolution: A Practical Approach to Global Optimization*. Springer, 2005, ISBN 9783540209508

[7] Beyer, Hans G., Schwefel, Hans P.: *Evolution strategies: A comprehensive introduction*. Dortmund, Germany, 2002

[8] Hansen, Nikolaus, Ostermeier, Andreas: *Adapting Arbitrary Normal Mutation Distirbutions in Evolution Strategies: The Covariance Matrix Adaptation*. Berlin, Germany

[9] Shir, Ofer M., Back, Thomas : *Dynamic Niching in Evolution Strategies with Covariance Matrix Adaptation*. Dortmund, Germany

[10] *http://en.wikipedia.org/wiki/Intersection_ of_ a_ polyhedron_ with_ a_ line*. Wikipedie [EN]

[11] *http://control.ee.ethz.ch/ mpt/*. Multi-Parametric Toolbox (MPT)

[12] *http://www.mathworks.com/help/toolbox/gads/f6010dfi3.html*. MathWorks, Inc., on-line dokumentace

[13] Bartz-Beielstein, Thomas : *Experimental Research in Evolutionary Computation*. ISBN 978-3-540-32026-5, Springer 2006

Seznam použitých zkratek

EA - Evoluční algoritmy

GA - Genetický algoritmus

DE - Defierenciální evoluce

ES - Evolučníá strategie

ES-CMA - Evoluční strategie s adaptací kovarianční matice (Evolution Strategy with Covariance Matrix Adaptation)

ES-DN - Evoluční strategie s dynamickým shlukováním (Evolution Strategy with Dynamic Niching)

ES-CMA-DN - Evoluční strategie s adaptací kovarianční matice a dynamickým shlukováním (Evolution Strategy with Covariance Matrix Adaptation and Dynamic Niching)

RBF sítě - neuronové sítě, které používají RBF funkce jako aktivaci (Radial basis function networks)

Přílohy

Součástí diplomové práce je přiložený CD-ROM, který obsahuje

- zdrojové soubory implementace algoritmů
- testovací data a příklady demonstrující použití algoritmů
- tuto práci ve formátu PDF