Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Použití evolučních optimalizačních technik k řešení NP úplných problémů

Ondřej Franěk

Bakalářská práce

2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Ondřej Franěk**
Osobní číslo:	**I09103**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Použití evolučních optimalizačních technik k řešení NP úplných problémů**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Anotace:
Cíl práce: Cílem je nalézt a vhodně modifikovat různé evoluční algoritmy tak, aby pokud možno efektivně řešily ukázkové NP úplné problémy (dva loupežníci, problém batohu, obchodní cestující, ...).
Teoretická část: Je třeba popsat NP úplnost po teoretické stránce, popsat evoluční algoritmy, které budou použity a dále popsat historii a zadání jednotlivých NP úplných problémů.
Implementační část: Student v JAVĚ (případně v jiném jazyce, podle domluvy) naprogramuje funkce pro použití jednotlivých evolučních algoritmů. Tyto funkce pak aplikuje na ukázkové NP úplné problémy a porovná jejich účinnost.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

***ZELINKA, I. Umělá inteligence v problémech globální optimalizace. Praha : BEN - technická literatura, 2002. 193 s. ISBN 80-7300-069-5**
***HYNEK, J. Genetické algoritmy a genetické programování. Praha: GRADA, 2008. 200 s. ISBN 978-80-247-2695-3**

Vedoucí bakalářské práce: **Ing. Petr Doležel, Ph.D.**
Katedra řízení procesů

Datum zadání bakalářské práce: **21. prosince 2012**
Termín odevzdání bakalářské práce: **10. května 2013**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 29. března 2013

Prohlášení

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 15. 8. 2013 Ondřej Franěk

Poděkování

Rád bych poděkoval Ing. Petru Doleželovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce.

Anotace

Bakalářská práce je zaměřená na používání evolučních optimalizačních technik k řešení NP úplných problémů. V teoretické části jsou vysvětleny a popsány: NP úplnost, vybrané NPC problémy a evoluční algoritmy.

V praktické části jsou implementovány a optimalizovány jednotlivé metody evolučních technik. Tyto metody jsou dále aplikovány na ukázkových NP problémech a závěrem jsou porovnány jejich účinnosti.

Klíčová slova

Genetické algoritmy, evoluční strategie, diferenciální evoluce, horolezecký algoritmus, NP úplnost, problém batoh, hledání globálního minima

Title

Evolutionary Algorithms for Solving of NP-complete Problems

Annotation

This Bachelor work is focused on using evolution of optimization techniques for solution NP-complete problems. In the theoretical part are explained and described: NP completeness, selected NPC problems and evolution algorithms.

In the practice part are implemented and optimized the individual methods of evolutionary techniques. These methods are applied to the demonstration NP issues and conclusions are compared their effectiveness.

Keywords

Genetic algorithm, Evaluation strategy, Differential evolution, Hill climbing, NP-complete, Problem bagging, search global minima

Obsah

Seznam zkratek

ES Evoluční strategie

HC Horolezecký algoritmus

GA Genetický algoritmus

DE Diferenciální evoluce

Seznam obrázků a tabulek

Úvod

Tématem mé bakalářské práce je použití evolučních optimalizačních technik k řešení NP úplných problémů. Pro NP úplné problémy nebyl zatím vymyšlen žádný algoritmus, který by problém s přesností a v čase vyřešil. Proto se využívají evoluční optimalizační techniky, které v reálném čase mohou nalézt řešení blížícímu se k optimálnímu. Tyto techniky se inspiruji ve fyzice a v biologii. Většina algoritmů je inspirována Darwinovou teorií evoluce.

Vlastní práce je zaměřena na vývoj aplikace, která bude implementovat jednotlivé evoluční optimalizační techniky pro efektivní řešení NP úplných problémů.

Bakalářská práce je rozdělena na teoretickou část, kde je popsán úvod do problematiky NP úplných problémů a také zde budou popsány jednotlivé algoritmy evolučních optimalizačních technik a na část technickou, kterou představuje grafická aplikace, kde je možné vidět průběhy algoritmů a výsledné hodnoty po provedení.

1 Cíl bakalářské práce

Cílem bakalářské práce je nalézt a vhodně modifikovat různé evoluční algoritmy tak, aby pokud možno efektivně řešily ukázkové NP úplné problémy (dva loupežníci, problém batohu, hledání globálního minima, obchodní cestující, …).

Teoretická část:

- Vysvětlit třídu složitosti NP-úplnost
- Popsat jednotlivé evoluční algoritmy

Praktická část:

- Implementace jednotlivých evolučních algoritmů v jazyce JAVA
- Aplikace algoritmů na vybrané problémy
- Porovnání účinnosti jednotlivých algoritmů

2 Třídy složitosti

2.1 Třída P

Třída složitosti P obsahuje všechny možné úlohy, u kterých jde nalézt řešení pomocí deterministického Turingova stroje v polynomiálním čase. Jinak řečeno, problém $L \in P \Leftrightarrow \exists$ polynom f a $\exists$ algoritmus A takový, že $\forall x : L(x) = A(x)$ a $A(x)$ doběhne v čase $O(f(x))$. Třída P obsahuje velké množství přirozených problémů. Například hledání největšího společného dělitele a nalezení maximálního párování grafů.

2.2 Třída NP

Třída složitosti NP obsahuje úlohy, u kterých využíváme nedeterministický Turingův stroj, který umožňuje v každém kroku rozvětvit výpočet na n větví, ve kterých pak hledáme optimální řešení daného problému. Jsou to ty problémy, u kterých můžeme ověřit jejich řešení v polynomiálním čase, ale nevíme, zda lze také v polynomiálním čase vyhledat řešení.

Tato třída složitosti NP obsahuje například problém faktorizace přirozených čísel, faktorizace celých čísel, izomorfismus grafů, problém obchodního cestujícího a další.

2.3 Třída NPC (NP-úplnost)

Třída složitosti NPC spadá pod třídu složitosti NP. Každý problém z NP může být polynomiálně převeden na problém NPC. Jedná se o skupinu nejtěžších problémů z třídy NP, pro které se ještě nepodařilo najít polynomiální algoritmus, ale ví se, že vyřešením jednoho problému by se daly vyřešit i ostatní problémy. Při vyřešení tohoto problému by se třída NP zařadila do třídy P (NP = P).[1]

2.4 Vztah mezi P a NP

Tento vztah je prozatím neobjasněn a spadá mezi problémy tisíciletí. Problémy tisíciletí je označeno 7 matematických problémů, které v roce 2000 vyhlásil Clayův matematický institut jako nejdůležitější otevřené problémy soudobné matematiky. Na vyřešení každého z nich institut vypsal odměnu jednoho milionu dolarů. Do dnešního dne byl vyřešen pouze jediný ze 7 problémů.[2]

3 NPC problémy

NPC problémy můžeme rozdělit do několika kategorií. Já jsem problémy rozdělil do 3 kategorii logické, grafové, číselné.

Logické

- SAT
- 3-SAT
- 3,3-SAT
- SAT pro obecné formule
- Obvodový SAT

Grafové

- Nezávislá množina
- Klika
- 3D
- Barvení grafu
- Hamiltonovská kružnice
- Globální optimalizace

Číselné

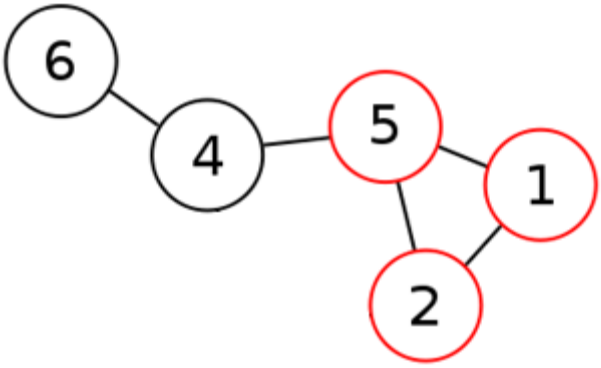
- Batoh
- Problém obchodního cestujícího
- Batoh – optimalizace
- Loupežníci
- Celočíselné lineární programování

3.1 Problém SAT

Rozhodovací problém splnitelnosti booleovských formulí se ptá, zdali existuje ohodnocení, ve kterém je daná formule zapsaná v konjunktivním normálním tvaru, pravdivá.

3.2 Problém klika

Klika je podgraf grafu, který je úplným grafem. To znamená, že každý vrchol podgrafu je spojen minimálně jednou hranou se všemi ostatními vrcholy podgrafu. Klikovost grafu je celé číslo a udává největší počet vrcholů v daném grafu. Klikovost grafu se značí ω(g).



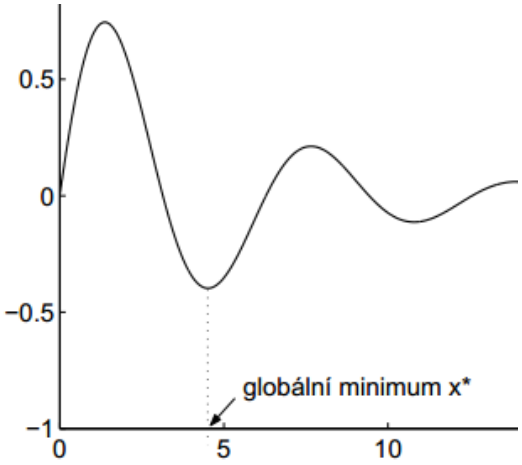
Obr. 1 – Ukázka kliky v grafu

3.3 Problém barvení grafů

Tento problém je jednou z části teorie grafů, která se zabývá výpočtem minimálního počtu potřebných barev k obarvení grafu. Přiřazovat barvy můžeme k různým objektům grafu, například jednotlivým vrcholům, hranám, stěnám atd. Přiřazená barva vrcholu nesmí sousedit se stejnou barvou jiného vrcholu, které jsou mezi sebou propojené hranou.

3.4 Problém globální optimalizace

Tento problém řeší nalezení souřadnic v definičním oboru funkce, kde hodnota funkce dosahuje extrémní hodnoty. Hledá se nejmenší nebo největší hodnota funkce. Problém globálního minima funkce s jedním argumentem můžeme vidět na obrázku (Obr. 2).



Obr. 2 – Globální minimum

Na obrázku (Obr. 2) můžeme vidět, že v definičním oboru funkce D[0,15], funkce obsahuje několik minim, ale jen jedno globální. V globálním minimu x^* je nejmenší funkční hodnota grafu.

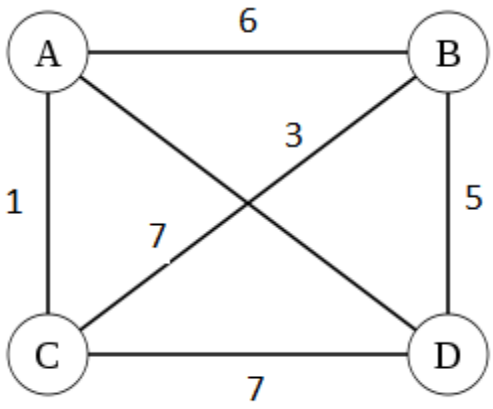
U funkcí, u kterých lze provést první a druhou derivaci, můžeme globální extrémy nalézt podle jednoduchého výpočtu. Nalézt takto jednoduše globální extrémy nelze u všech funkcí, například funkce má více než jeden parametr nebo při vstupu počítáme s vektorem. Navíc není každá funkce diferencovatelná, ale přesto potřebujeme její globální extrémy nalézt anebo alespoň se přiblížit k optimálnímu řešení.

3.5 Problém obchodního cestujícího

V úloze je za cílem nalézt v zadaném ohodnoceném úplném grafu takovou cestu, která prochází předem určenými vrcholy a zároveň je nejméně ohodnocena. Jinými slovy se jedná o nalezení nejkratší cesty, aby vedly přes předem určené body, mezi kterými známe všechny vzdálenosti, které je možné vidět na obrázku (Obr. 3).

3.5.1 Formulace pomocí lineárního programování

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} y_{ij} \tag{1}$$



Obr. 3 – Ohodnocený graf měst

3.6 Problém batoh

V úloze je cílem vybrat z množiny předmětů předměty o specifické ceně a hmotnosti, vložit do batohu, který má předem určenou nosnost. Batoh musí obsahovat množinu předmětů o maximální ceně. Jednotlivé předměty nelze dělit, buď jsou do batohu vloženy celé, nebo zde nemohou být.

3.6.1 Složitost

Složitost můžeme zapsat vzorcem $O(n*2^n)$. Kde n je počet předmětů v množině, ze které vybíráme.

3.6.2 Formulace pomocí lineárního programování

$$max \sum_{i \in I} x_i * c_i \tag{2}$$

Za podmínky:

$$\sum_{i \in I} x_i * w_i < W \tag{3}$$

4 Evoluční optimalizační techniky

Evoluční algoritmy jsou na principu evoluce, kterou před 150lety popsal Charles Darwin ve své knize „O původu druhů“. Darwinova evoluční teorie spočívá v jednoduché myšlence, že život je v čase proměnlivý a mladší formy života následují za staršími formami. [3]

Tato myšlenka už existovala před Darwinem, například už Darwinův dědeček Erasma Darwin se tímto problémem zajímal, ale Charles Darwin a jeho následníci jí rozvinuli do podoby umožňující vysvětlit různorodost forem života, přizpůsobování živých organismů na proměnlivé prostředí. Koncepce evoluce spočívá v kombinaci následujících šesti principů:

1. Živé organizmy se s postupem času vyvíjí.
2. Evoluční změny mají charakter větvení.
3. Nové druhy vznikly tak, že populace se rozdělila na izolované podpopulace.
4. Evoluční změny jsou postupné.
5. Mechanizmus adaptivní změny je přírodní výběr.
6. Jeden z hlavních mechanizmů vzniku složitých struktur je „bricolage“.

4.1 Optimalizační a heuristické algoritmy

Evoluční algoritmy zařazujeme do skupiny heuristických algoritmů. Heuristické algoritmy dále můžeme rozdělit do dvou skupin na deterministické algoritmy a stochastické algoritmy.

Deterministické algoritmy vždy začínají ze stejných vstupních podmínek, každý krok algoritmu je předem definovaný a proto je algoritmus předvídatelný a řešení při každém spuštění je stejné.

Stochastické algoritmy se liší v tom, že některé kroky se generují náhodně a proto výsledky jednotlivých řešení se mohou lišit. Proto se doporučuje jednotlivá řešení spustit vícekrát a z této množiny výsledků vybrat optimální hodnotu pro vaší optimalizační úlohu.

4.2 Stochastické heuristické metody

Stochastické heuristické metody (stochastic heuristic methods) se také někdy označují jako metaheuristiky, protože obsahují pouze obecný algoritmus a ostatní operace je třeba implementovat v závislosti na zkoumaném problému. Tyto operace jsou například u genetických algoritmů mutace, křížení nebo prohledávání okolí populace. Protože se tyto

metody často inspirují přírodou a myšlenkou Darwinovy evoluce nazýváme je také evolučními algoritmy.

Podle strategie, kterou evoluční algoritmy používají, je můžeme rozdělit do dvou skupin. Skupina založena na bodové strategii, která především využívá prohledávání sousedství, ve kterém hledáme lepší řešení, než je aktuální. Do této skupiny patří algoritmy simulovaného žíhání, horolezecký algoritmus nebo zakázané prohledávání. Druhá skupina je založena na populaci. Kde vytváříme při prvním kroku novou náhodnou populaci a dále postupujeme podle strategie algoritmu, kde se provádí operace selekce, křížení, mutace nebo výběr nejlepšího potomka. Tyto algoritmy jsou například genetický algoritmus, evoluční strategie, diferenciální evoluce a další.

4.3 Algoritmy založené na bodové strategii

4.3.1 Simulované žíhání

Algoritmus simulovaného žíhání má základy ve fyzice, na rozdíl od jiných evolučních optimalizačních technik, které si většinou berou základy z biologie. Ve fyzice žíhání je proces, při kterém je těleso zahřáno na vysokou teplotu a postupným snižováním teploty se odstraňují vnitřní defekty. Při prohledávání stavového prostoru se může snadno stát, že algoritmus uvázne v lokálním minimu. V metodě se tomu snažíme zabránit tím, že zpočátku provádíme velké změny a díky těmto změnám se můžeme jednodušeji dostat z lokálního minima. Velikost změn záleží na aktuální teplotě, čím větší teplota tím větší změny se provádí. Algoritmus žíhání může také přijmout horší řešení než je aktuální. Pravděpodobnost přijetí horšího řešení závisí na aktuální teplotě žíhání.

Algoritmus:

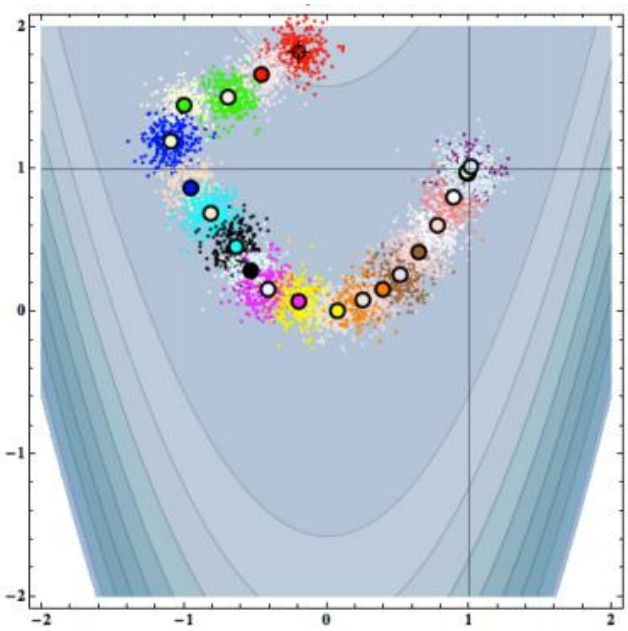
```
aktualniReseni = InicializacePocatecnichResení(x0);
teplota = NastaveniTeploty(1);
while(PocetIteraci){
        smer = vyberNahodnySmer();
        noveReseni = posunOVzdalenostASmer(NahodnaVzdalenost, smer);
        if(aktualniReseni>=noveReseni){
                aktualniReseni = noveReseni;
        }else{
                if(aktualniReseni!=noveReseni){
                        if( pravdepodobnostPrijetiHorsihoReseni() ){
```

```
                                aktualniReseni = noveReseni;
                                teplota = snizeniTeploty();
                            }
                        }
                    }
}
```

4.3.2 Horolezecký algoritmus (HC)

Základní myšlenkou horolezeckého algoritmu je, že náhodně zvolíme první uzel, ze kterého má prohledávání zahájit. Tento uzel nastavíme jako aktuální a spustíme algoritmus. Algoritmus provádíme do doby, dokud nenastane jedna z ukončovacích podmínek. Ukončovacích podmínek můžeme mít více například předem určený počet iterací nebo podmínka, která hlídá zacyklení v uzlu (v okolí aktuálního uzlu nemůže najít lepší řešení).
První krok algoritmu je vygenerování sousedních uzlů aktuálního řešení. Z tohoto seznamu sousedních uzlů vybereme uzel, který je lepší než aktuální uzel. Tímto lepším uzlem nahradíme aktuální uzel a provedeme algoritmus znova. Jednotlivé kroky můžete vidět na obrázku (Obr. 4).

Nevýhodou tohoto algoritmu je možné zacyklení v lokálním extrému. Této nevýhodě můžeme předejít, pokud algoritmus zpustíme více krát z náhodně zvoleného uzlu.



Obr. 4 – Průběh HC algoritmu

Tomuto uzlu určíme okolní sousedy a ze sousedů vybereme nejlepší řešení a zvolíme ho jako aktuální. Tento krok opakujeme, dokud není splněna ukončovací podmínka. Ukončovací podmínka může být předem určený počet iterací a podmínka, která hlídá zacyklení v globálním minimu nebo maximu.

Algoritmus:

```
aktualniReseni = inicializacePocatecnihoReseni();
while (t != pocetIteraci || uvaznutiVExtremu() ) {
        reseniVOkoli = generujReseniVOkoli(aktualniReseni, sigma);
        noveReseni = najdiNejlepsiReseni(reseniVOkoli);
        if (porovnejReseni(aktualniReseni, noveReseni)) {
          aktualniReseni = noveReseni;
        }
        t++;
}
```

4.4 Algoritmy založené na strategii populace

4.4.1 Genetický algoritmus (GA)

Algoritmus bere inspirace z Darwinovy teorie přirozeného výběru, přežívají jen nejlépe přizpůsobení jedinci populace. Míru přizpůsobení značíme jako fitness jedince. Fitness můžeme v biologii chápat jako relativní schopnost přežití jedince a následnou reprodukci genotypu jedince.

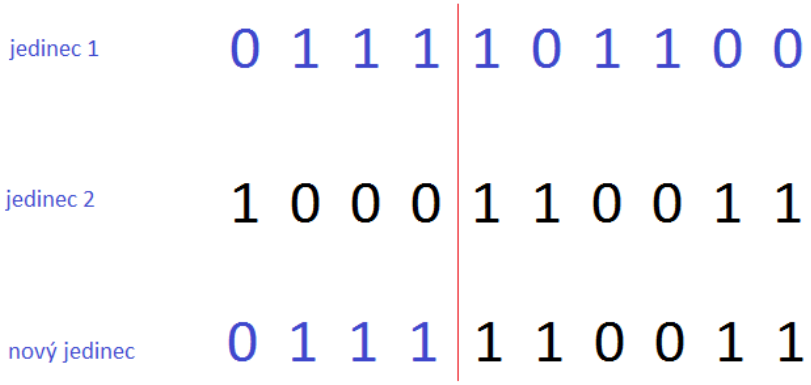
Výhodou genetického algoritmu je hlavně jednoduchost implementace na různé typy úloh. Při implementaci v teorii umělé inteligence je genetický algoritmus proces postupného vylepšení populace jedinců opakovanou aplikací genetických operátorů, které vedou k evoluci takových jedinců, kteří lépe vyhovují než jedinci, ze kterých noví jedinci vznikli. Genetické operátory jsou například křížení, mutace a přirozený výběr.

4.4.1.1 Populace a jedinec

Populace se skládá z více jedinců. Každý jedinec má určeny své geny, které v informační technologii může zapsat jako binární vektor.

4.4.1.2 Křížení

Při použití operátoru křížení vybereme z populace několik jedinců. Tito jedinci si mezi sebou náhodně vytvoří páry a nadále si vymění část svých genů.



Obr. 5 – Ukázka křížení jedinců

4.4.1.3 Mutace

Používání operátoru mutace se řídí podle nastavené pravděpodobnosti. Operátor náhodně vybere náhodný gen v náhodném chromozomu a změní jeho hodnotu.

4.4.1.4 Selekce

Operátor selekce porovnává a vybírá podle hodnoty fitness nejlepší nové jedince a kopíruje je do nové kvalitnější populace.

Algoritmus:

```
t=0;
populace(t) = inicializaceNovePopulace();
ohodnoceniPopulace(populace(t));
while(ukoncovaciPodminka){
        t++;
        populace(t) = selekceJedincu(populace(t-1));
        aplikujKrizeni(populace(t));
        aplikujMutaci(populace(t));
        ohodnotPopulace(populace(t));
}
```

4.4.2 Evoluční strategie (ES)

Evoluční strategie byla navržena v letech 1960 až 1970. Hlavními vývojáři byly Ingo Rechenberg, Hans-Paul Schwefel a jeho spolupracovníci.[8]

Základní myšlenka ES je podobná slepému náhodnému prohledávání, ale rozdíl je v tom, že nový jedinec se generuje jako mutace jedince tak, že jednotlivé složky vektoru x se změní přičtením hodnot normálně rozdělených náhodných veličin podle vztahu:

$$y = x + u, \;\; u \sim N(0, \sigma^2 I) \tag{4}$$

Kde $u = (U_1, U_2, \ldots U_n)$ je náhodný vektor, jehož každý prvek je náhodná veličina $U_i \sim N(0, \sigma^2 I)$, $i = 1,2, \ldots d$ a tyto veličiny jsou navzájem nezávislé.

4.4.2.1 Směrodatná odchylka

Směrodatná odchylka určuje velikost mutace daného jedince. Odchylka nemusí být vždy jen jedna, ale můžeme využívat také i vektor odchylek:

$$\sigma = (\sigma_1, \sigma_2, \ldots \sigma_n) \tag{5}$$

Navíc ne každá iterace evoluční strategie má stejnou směrodatnou odchylku. Rechenberg studoval vliv velikosti mutace při generování nových potomků. Z jeho studie odvodil pravidlo jedné pětiny.

4.4.2.2 Pravidlo jedné pětiny

Podle pravidla jedné pětiny určujeme velikost aktuální směrodatné odchylky za pomocí procentuálního vyjádření úspěšnosti mutace. Pro pravidlo jedné pětiny platí vztah:

$$\sigma_i \begin{cases} c_1 \sigma_{i-1} \; když \; \varphi(n) < \frac{1}{5} \\ c_2 \sigma_{i-1} \; když \; \varphi(n) > \frac{1}{5} \\ \sigma_{i-1} \; když \; \varphi(n) == \frac{1}{5} \end{cases} \tag{6}$$

$c_1 < 1$ a $c_2 > 1$ jsou vstupní parametry, které řídí zvětšovaní či zmenšováni mutace potomků. Konstanta c_1 se obvykle volí hodnota 0.82 a konstantu c_2 můžeme dopočítat ze vzorce: $c_2 = \frac{1}{c_1}$.

Pro rychlejší výpočet byla do ES rozšířena populace větší než 1. Pro orientaci si zavedeme velikost populace rodičů M a velikost populace potomků L. Evoluční strategie volí dvě možná řešení nové populace rodič:

- ES(M + L)
- ES(M, L)

4.4.2.3 Strategie ES(M + L)

V této strategii spojíme populaci rodičů a populaci potomků. Celou novou populaci seřadíme podle hodnoty fitness a vybereme jen nejlépe hodnocené potomky. V této variantě se dodržuje tzv. elitismus – nejlepší jedinci se vždy dostanou do další populace.

4.4.2.4 Strategie ES(M, L)

V této strategii celou populaci rodičů nahradíme nejlepšími jedinci z populace potomků.

Algoritmus:

```
populace(t) = inicializaceNovePopulace();
while (ukoncovaciPodminka) {
        t++;
        uspesnost = uspesnostMutace(populace (t-1));
        if (uspesnost < (1 / 5)) {
                sigma = sigma * konstanta;
         }
        if (uspesnost > (1 / 5)) {
                sigma = sigma / konstanta;
        }
        potomci = mutace(populace (t-1), sigma);
        populace(t)  = vyberNejlepsichChromozom(potomci ));
 }
```

4.4.3 Diferencialní Evoluce (DE)

Algoritmus diferenciální evoluce je postupem k heuristickému hledání globálního minima, který navrhli R. Storn a K. Price kolem roku 1995. Tento algoritmus je jeden z nejnovějších evolučních algoritmů. Diferenciální evoluce patří mezi jeden z nejpopulárnějších algoritmů a je často implementován a rozvíjen a je na něm založeno mnoho výzkumů. Velkou výhodou DE je často mnohem rychleji konverguje k optimálnímu řešení oproti ostatním evolučním algoritmům pro globální optimalizaci.

Při implementaci diferenciální evoluce vytváříme nulovou populaci Q. Z této populace se vždy pro každého jedince vytvoří nový potencionální konkurentní jedinec a zařadí se do aktuální populace T jedinec, který má lepší fitness. Nová populace N vznikne zkřížením staré populace Q a aktuální populace T. Křížení probíhá nahrazení starého jedince novým podle pravděpodobnosti c.

Pro vytváření nových jedinců je hned více možných postupů.

4.4.3.1 Postup RAND

Při použití postupu RAND vyberme ze staré populace 3 jedince a z těchto 3 jedinců vypočítáme jedince podle vztahu:

$$u = r_1 + F(r_2 - r_3) \tag{7}$$

r_1,r_2,r_3 jsou různí potomci náhodně vybraní ze staré populace.

4.4.3.2 Postup BEST

Postup BEST vybere ze staré populace nejlepšího jedince a k němu 4 náhodně zvolené jedince. Nový jedinec se vypočítá podle vztahu:

$$u = x_{best} + F(r_1 + r_2 - r_3 - r_4) \tag{8}$$

x_{best} je nejlepší jedinec ze staré populace, r_1, r_2, r_3, r_4 jsou různí potomci náhodně vybraní ze staré populace.

Algoritmus:

```
populace(t) = inicializaceNovePopulace();
while (ukoncovaciPodminka) {
        potomci = mutaceAKrizeni(populace(t--), mutaceKonstanta, prahKrizeni));
        populace(t) = vyberNejlepsiChromozomy(potomci);
        t++;
}
```

5 Aplikace

Úkolem mé bakalářské práce bylo vybrat si a implementovat ve vhodném programovacím jazyce několik evolučních optimalizačních technik a otestovat na řešení NP úplných problémů. Jednotlivé metody poté porovnat a zjistit jejich účinnost.

5.1 Výběr NP úplných problémů

Při studii NP úplných problémů, jsem si pro svou bakalářskou práci vybral jeden známý a jeden méně známý NP úplný problém. První zmiňovaný problém je hledání globálního minima na testovacích funkcích a druhý je problém batohu.

5.2 Problém hledání globálního minima

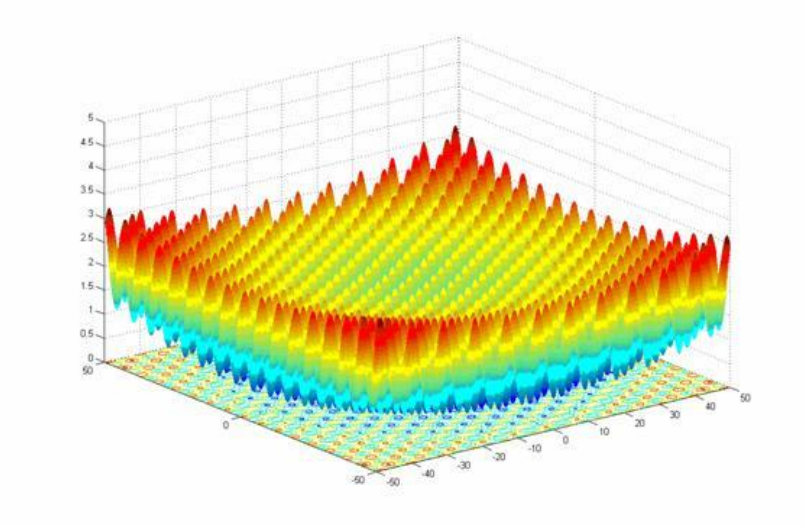
Pro testování evolučních optimalizačních technik jsem při hledání globálního minima použil testovací funkci Griewangk.

5.2.1 Testovací funkce Griewangk

Testovací funkce se zapisuje vzorcem:

$$f_n(x_1, \dots, x_n) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) \tag{9}$$

x ∈ (-600,600)



Obr. 6 – Funkce griewangk

Třída, kterou ve své aplikaci využívám, implementuje rozhraní IFunkce, které obsahuje všechny důležité metody pro implementování jakékoliv testovací funkce. Rozhraní

jsem vytvořil, kvůli abstrakci funkce. Použitím rozhraní v aplikaci lze implementovat i jiné například složitější testovací funkce.

Ukázka rozhraní IFunkce

```
public interface IFunkce {
        public void setX_max(int x_max);
        public void setX_min(int x_min);
        public void setY_max(int y_max);
        public void setY_min(int y_min);
        public double getFitness(double x, double y);
        public double getPomer();
}
```

Ukázka funkce getFitness()

```
public double getFitness(double x, double y) {
        return ((((x * x) + (y * y)) / 4000) - (Math.cos(x) * Math.cos((y / Math.sqrt(2))))) + 1;
}
```

5.3 Problém batohu - optimalizován

V aplikaci jsem chtěl mít i nějaký méně známý NP úplný problém, proto jsem si vybral problém batohu. Problém batohu se zdá jednoduchý. Máme množinu předmětů, kde každý předmět má předem určenou cenu a hmotnost. Dále máme batoh s udanou celkovou nosností. Za úkol máme vybrat předměty, které batoh unese a zároveň cena všech předmětů bude maximální.

V aplikaci jsem si kvůli abstrakci a přehlednosti vytvořil k problému batohu dvě třídy, které obstarávají všechny používané funkce.

Třída Batoh implementuje rozhraní IBatoh a obsahuje následující metody:

- public boolean pridejPredmet(IPredmet p);
- public boolean odeberPredmet(IPredmet p);
- public double ohodnotBatoh();
- public int getPocetPredmetu();
- public List<IPredmet> getBatoh();

- public double getCena();
- public double getHmotnostPredmetu();
- public String getNazev();
- public double getNosnost();
- public boolean isUspesnost();

Třída Předmět implementuje rozhraní IPredmet a obsahuje následující metody:

- public double getHodnoceni();
- public double getCena();
- public double getVaha();

5.4 Implementace evolučních optimalizačních technik

Ve své aplikaci jsem implementoval celkem čtyři evoluční optimalizační techniky, které se mi zdály pro vybrané problémy a porovnání zajímavé. Pro problém hledání globálního minima na testovací funkci jsem vybral následující algoritmy:

- Horolezecký algoritmus
- Evoluční strategie
- Diferenciální evoluce

Pro problém batohu jsem použil následující algoritmy:

- Evoluční strategie
- Genetický algoritmus

5.4.1 Horolezecký algoritmus

V aplikaci jsem horolezecký algoritmus implementoval pro hledání globálního minima na testovací funkci Griewangk. Zjednodušený vývojový diagram základního algoritmu můžete vidět v příloze č.1 – Vývojový diagram - Horolezecký algoritmus.

Pro možné rozšíření i na ostatní problémy jsem si vytvořil interface IHorolezeckyAlgoritmus, který obsahuje všechny potřebné metody:

- public void startHorolezeckehoAlgortimu();
- public Souradnice novePocatecniReseni();
- public List<Souradnice> generujReseniVOkoli(Souradnice p, double sigma, int pocetReseni);
- public Souradnice najdiNejlepsiReseni(List<Souradnice> seznam);
- public boolean porovnejReseni(Souradnice aktualniReseni, Souradnice tmp);

Tento interface implementuje třída HorolezeckyAlgoritmusFunkce, kde jsou naprogramovány všechny potřebné funkce pro hledání globálního minima.

V této třídě je vytvořen konstruktor pro nastavení základních parametrů:

- **Počet iterací** – Jedna z ukončovacích podmínek určuje, kolikrát se algoritmus bude opakovat.
- **Velikost populace** – Počet souřadnic, pro které se algoritmus bude provádět. Můžeme najednou provádět i více výpočtů. Slouží k rychlejšímu nalezení globálního minima.
- **Sigma** – Maximální vzdálenost nového potomka od rodiče.
- **Počet řešení** – Proměnná, které určuje, kolik se vygeneruje nových potomků.
- **Funkce** – Třída, která implementuje interface IFunkce. Slouží pro vložení jakékoliv testovací funkce.

Jednou z nejdůležitějších metod v algoritmu horolezeckého algoritmu je metoda generování nových potomků v okolí. V této metodě generujeme všechny možné potomky v okruhu sigma a z těchto potomků pak vybíráme určitý počet, kteří budou v nové populaci. Pro výpočet vzdálenosti jsem použil matematický vzorec pro výpočet vzdálenosti dvou bodů, pro který platí:

$$A = [a_x, a_y], B = [b_x, b_y] \tag{10}$$

$$v = \sqrt{(a_x - a_y)^2 + (b_x - b_y)^2} \tag{11}$$

Zápis v programovacím jazyce:

```
double vzdalenost = Math.sqrt(((x - aktualniX) * (x - aktualniX)) + ((y - aktualniY) * (y - aktualniY)));
```

5.4.2 Diferenciální evoluce

Algoritmus jsem si vybral z několika důvodů:

- Patří mezi nejnovější (vznikl v roce 1995)
- Jeden z nejvíce používaných a často rozvíjených algoritmů z evolučních optimalizačních technik

Hlavní stavební kámen algoritmu můžete vidět v příloze č.2 – Vývojový diagram - Diferenciální evoluce.

Pro implementaci na problém globálního minima byl tento algoritmus nejjednodušší a nejrychlejší. V aplikaci používám třídu DiferencialniEvoluceFunkce, která implementuje rozhraní IDiferencialniEvoluce se základními metodami:

- public void provedAlgoritmus();
- public List<List<Populace>> getPopulaceRodicu();
- public List<List<Populace>> getPopulacePotomku();
- public List<Populace> vygenerujPopulaciRodicu();
- public List<Populace> mutaceAKrizeni(List<Populace> s);
- public List<Populace> vyberNejlepsiChromozomy(List<Populace> potomci, List<Populace> rodice);

Třída implementující rozhraní IDiferencialniEvoluce obsahuje konstruktor, který nastavuje základní parametry pro spouštění algoritmu. Tyto parametry jsou:

- **velikost populace** – určení počtu jedinců v populaci
- **počet generací** – ukončovací podmínka po x-tém provedení
- **mutace** – konstanta, která se připočítává k novému jedinci při průběhu mutace
- **práh křížení** – pravděpodobnost proběhnutí křížení mezi jedinci

Metoda **mutaceAKrizeni()** zde představuje spojení dvou kroků algoritmu do jedné metody. V této metodě vybereme 3 náhodné jedince z populace rodičů a provedeme vektorový výpočet nového jedince, podle pravidla RAND:

$$u = r_1 + F(r_2 - r_3) \tag{12}$$

Implementace vzniku nové potoka:

```
x = prvekR3.getX() + ((prvekR2.getX() - prvekR1.getX()) * mutaceKonstanta);
y = prvekR3.getY() + ((prvekR2.getY() - prvekR1.getY()) * mutaceKonstanta);
```

Implementace křížení

```
if (Math.random() < prahKrizeni) {
        novyPotomek.setX(x);
} else {
        novyPotomek.setX(aktualniPrvke.getX());
}
 if (Math.random() < prahKrizeni) {
```

```
            novyPotomek.setY(y);
        } else {
            novyPotomek.setY(aktualniPrvke.getY());
        }
```

5.4.3 Evoluční strategie

Algoritmus ES jsem si při výběru zvolil z důvodu implementace na oba řešené problémy. Jak pro problém globálního minima, tak i pro problém batohu. Vývojový diagram můžete vidět v příloze č.3 – Vývojový diagram - Evoluční Strategie.

Pro každý problém jsem vytvořil vlastní třídu (EvolucniStrategieBatoh, EvolucniStrategieFunkce), která implementuje stejné rozhraní IEvolucniStrategie.
Rozhraní obsahuje následující metody:

- public void provedAlgoritmus();
- public Populace vytvorPopulaci();
- public Populace vyberNejlepsichChromozom(Populace staraPopulace, Populace novaPopulace);
- public Populace mutace(Populace p);
- public void ohodnotPopulaci(Populace p);
- public double uspesnostMutace(Populace p);
- public List<Populace> getPopulaceRodicu();
- public List<Populace> getPopulacePotomku();

5.4.3.1 Třída evolucniStrategieFunkce

Konstruktor této třídy obsahuje nastavující parametry pro správný průběh algoritmu. Hlavními parametry jsou:

- **Funkce** – Třída, která implementuje rozhraní IFunkce. Parametry funkce je tu schválně kvůli abstrakci. Možné vložit jakoukoliv testovací funkci.
- **Velikost populace** – Určení počtu jedinců v populaci.
- **Počet iterací** – Ukončovací podmínka po x-tém provedení.
- **Sigma** – Největší vzdálenost nového potomka.
- **Konstanta** – Konstanta, která zvětšuje či zmenšuje velikost sigma podle procentuální úspěšnosti mutace.

Pro algoritmus jsem vybral pravidlo ES(N+L), kde při vytváření nové populace jsem sjednotil populaci rodičů a nových potomků. Tuto novou populaci jsem seřadil podle hodnoty fitness a vybral jen nejlepší jedince a z těchto jedinců sestavil novou populaci.

Ukázka metody výběru nejlepšího jedince:

```
List<Souradnice> tmp = new ArrayList<>();
      for (Souradnice s : staraPopulace) {
         tmp.add(s);
      }
      for (Souradnice s : novaPopulace) {
         tmp.add(s);
      }
      Collections.sort(tmp, new srovnavaniSouradnic());
      List<Souradnice> tmp2 = new ArrayList<>();
      for (int i = 0; i < staraPopulace.size(); i++) {
         tmp2.add(tmp.get(i));
      }
 return tmp2;
```

Nejdůležitější metodou v tomto algoritmu byla metoda mutace. V této metodě probíhá výpočet nových souřadnic s připočítáním odchylky podle normálového rozdělení. Část zdrojového kódu můžete vidět zde:

```
          double normalRozlozeni = Math.abs(r.nextGaussian() * sigma) / 10
          double sx = ((Math.random() * (normalRozlozeni * 2)) - normalRozlozeni);
          newX = s.getX() + sx;
          double ssy = ((Math.sqrt((normalRozlozeni * normalRozlozeni) - (sx * sx))));
          double sy = ((Math.random() * (ssy * 2)) - ssy);
           newY = s.getY() + sy;
```

5.4.3.2 Třída evolucniStrategieBatoh

Konstruktor nastavuje všechny stejné parametry jako v třídě evolucniStrategieFunkce, jen místo parametru funkce zde třídě předávám množinu všech předmětů, ze kterých můžeme vybírat a vkládat předměty do batohu.

Metoda výběru nejlepšího jedince je stejná jako ve třídě evolucniStrategieFunkce. Využívá se také pravidla ES(N + L).

Jedinou metodou, která nemohla zůstat stejná a musela se složitěji upravovat je metoda mutace. V této metodě v parametru dostáváme ArrayList jedinců neboli batohů.

Podle vzorce $n_i = p_i + G(0, \sigma)$ si vypočítáme velikost odchylky mutace a určíme procentuální změnu v batohu. Podle této odchylky určíme počet předmětů, které můžeme vyměnit a nahradit z dané množiny.

Ukázka metody:

```
int celkovyPocetPredmetuVBatohu = s.getPocetPredmetu();
normalRozlozeni = Math.abs(r.nextGaussian() * sigma);
 int pocetProcent = (int) Math.round(normalRozlozeni) + 1;
double pocetPredmetuNaVymenu = (int) Math.round(celkovyPocetPredmetuVBatohu * pocetProcent
/ 100) +1;
IBatoh batohPotomek = new Batoh(s);
int a = 0;
while (a < pocetPredmetuNaVymenu) {
        int nahodnyPredmet = (int) Math.round(Math.random() *
        batohPotomek.getPocetPredmetu());
        IPredmet pr = batohPotomek.getPredmetZBatohu(nahodnyPredmet);
        batohPotomek.odeberPredmet(pr);
        a++;
}
pridejNahodnePredmety(batohPotomek);
```

5.5 Genetický algoritmus

Pro porovnání konvergence dat jsem implementoval jeden z nejstarších algoritmů evoluční optimalizační techniky na problém batohu. Vývojový diagram můžete najít v příloze č.4 - Vývojový diagram - Genetický algoritmus.

Pro moje použití jsem si vytvořil rozhraní IGenetickýAlgoritmus, který obsahuje hlavní metody potřebné pro průběh algoritmu.

- public void provedAlgoritmus();
- public Populace vytvorPopulaci();
- public Populace krizeni(Populace p);

- public Populace mutace(Populace p);
- public Populace selekce(Populace p);
- public List<Populace> getPopulaceRodicu();

Třída GenetickyAlgoritmusBatoh implementuje toto rozhraní a jsou zde navrženy metody přímo pro tento problém.

Při vytvoření instance této třídy nastavujeme nejdůležitější parametry pro chod algoritmu.

- **Množina všech předmětů** – ArrayList obsahuje všechny předměty, které můžeme použít při hledání nejlepší kombinace předmětů.
- **Počet iterací** – Nastavení maximálního počtu opakování algoritmu.
- **Pravděpodobnost mutace** – Konstanta, podle které určujeme, jestli daného jedince budeme mutovat nebo ne.
- **Velikost populace** – Určuje maximální počet jedinců v populaci.

Nejzákladnějšími metodami genetického algoritmu jsou selekce, mutace a křížení.

5.5.1 Metoda selekce

V metodě selekce jsem vybral možnost ruletového výběru. To znamená, že každý jedinec v populaci je ohodnocen hodnotou fitness. Jedinec s nejlepší ohodnocením má větší pravděpodobnost výběru.

Ukázka zdrojového kódu:

```
for (IBatoh s : p) {
        celkem += s.ohodnotBatoh();
    }
    for (int i = 0; i < p.size(); i++) {
        nahodny = Math.random() * celkem;
        aktualni = 0.0;
        for (IBatoh s : p) {
            if (aktualni <= nahodny && (aktualni + s.ohodnotBatoh()) >= nahodny) {
                tmp.add(s);
            }
            aktualni += s.ohodnotBatoh();
        }
    }
    return tmp;
```

5.5.2 Metoda křížení

V této metodě vytváříme nové potomky populace tím, že si dva jedinci prohodí části svých chromozomů. V implementaci pro problém batohu vždy určíme, o jaké hmotnosti předměty vyměníme, a tyto dvě množiny předmětů o stejné hmotnosti prohodíme.

Ukázka zdrojového kódu:

```
double rnd = Math.random() * nosnostBatohu;
 double aktualniVaha = rnd;
 Collections.sort(batoh1, new srovnaniDleVahy());
 Collections.sort(batoh2, new srovnaniDleVahy());
 for (IPredmet pr : batoh1) {
    if (pr.getVaha() < aktualniVaha) {
       potomek1.add(pr);
       aktualniVaha = aktualniVaha - pr.getVaha();
    } else {
       potomek2.add(pr);
    }
 }
 aktualniVaha = rnd;
 for (IPredmet pr : batoh2) {
    if (pr.getVaha() < aktualniVaha) {
       potomek2.add(pr);
       aktualniVaha = aktualniVaha - pr.getVaha();
    } else {
       potomek1.add(pr);
    }
 }
 t0 = b1.setMnozinaPredmetuVBatohu(potomek1);
 t1 = b2.setMnozinaPredmetuVBatohu(potomek2);
```

6 Porovnání jednotlivých optimalizačních technik

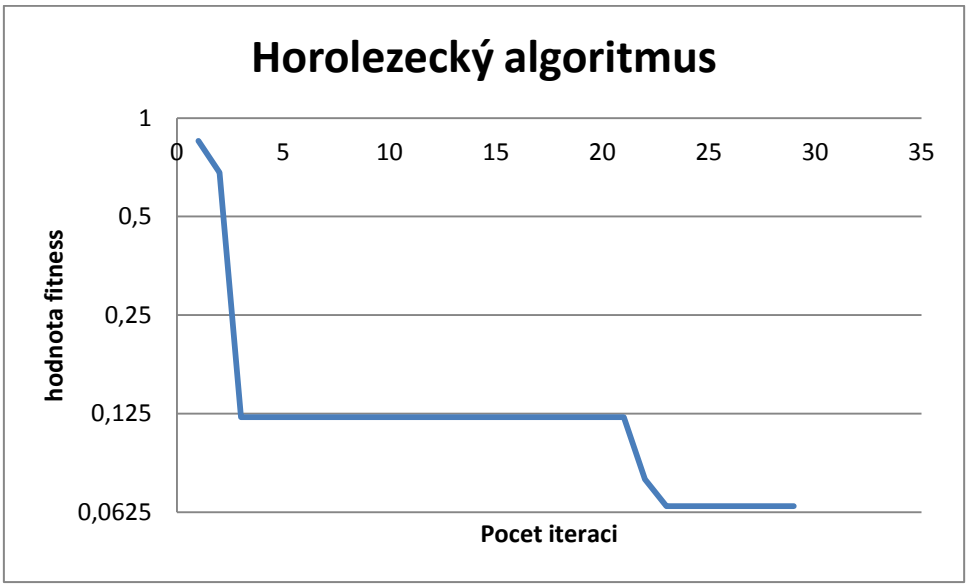
Na testování optimalizačních technik jsem využil export dat, která jsem exportoval do Microsoft Excel 2007. Data můžete nalézt na přiloženém CD.

Všechny testy probíhaly na stejném stroji za stejných podmínek. Testováno na funkci Griewangk v rozsahu [-5,-5] do [5,5].

6.1 Testování horolezeckého algoritmu

Základní nastavení hodnot:

- Velikost populace: 10
- Počet iterací: 100
- Sigma: 2
- Počet nových řešení: 400



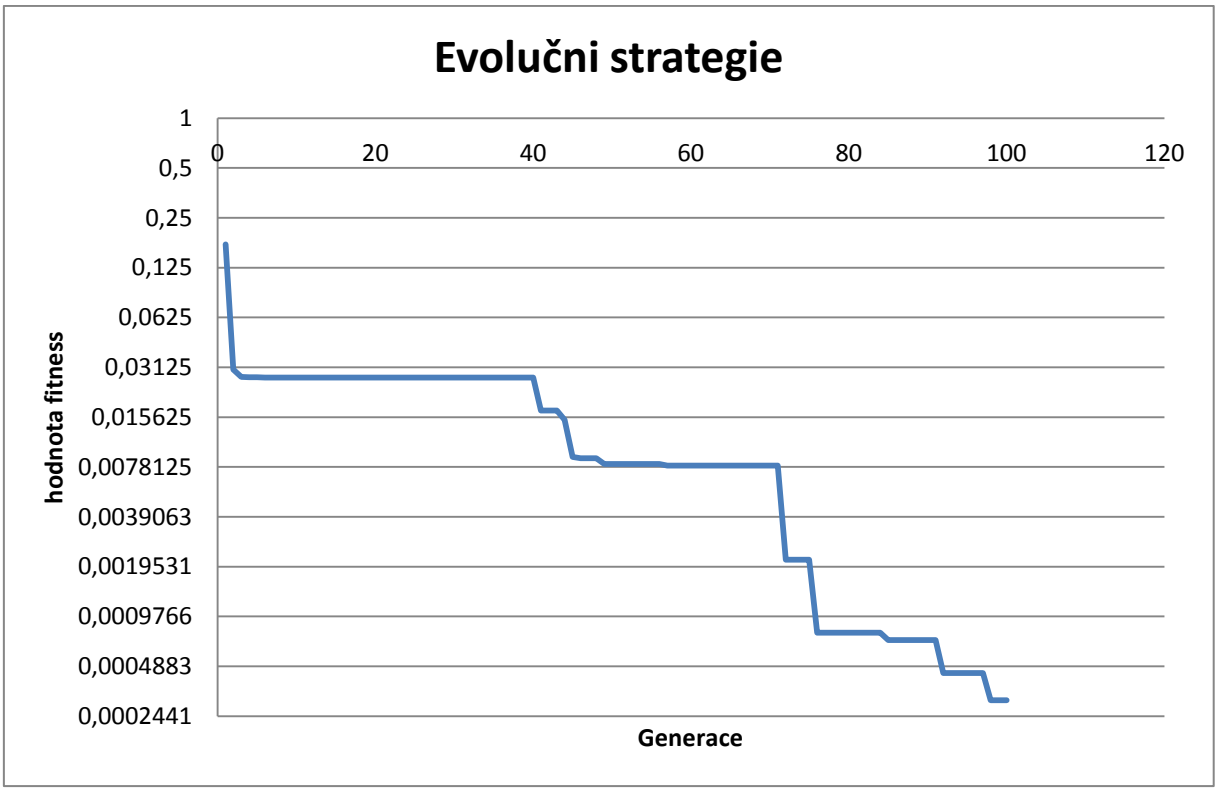
Obr. 7 – Průběh horolezecky algoritmus

Optimální řešení bylo nalezeno v hodnotě 0,0625.

6.2 Testování evoluční strategie

Základní nastavení hodnot

- Velikost populace: 10
- Počet iterací: 100
- Počet potomků: 5
- Konstanta: 0,807
- Sigma: 2



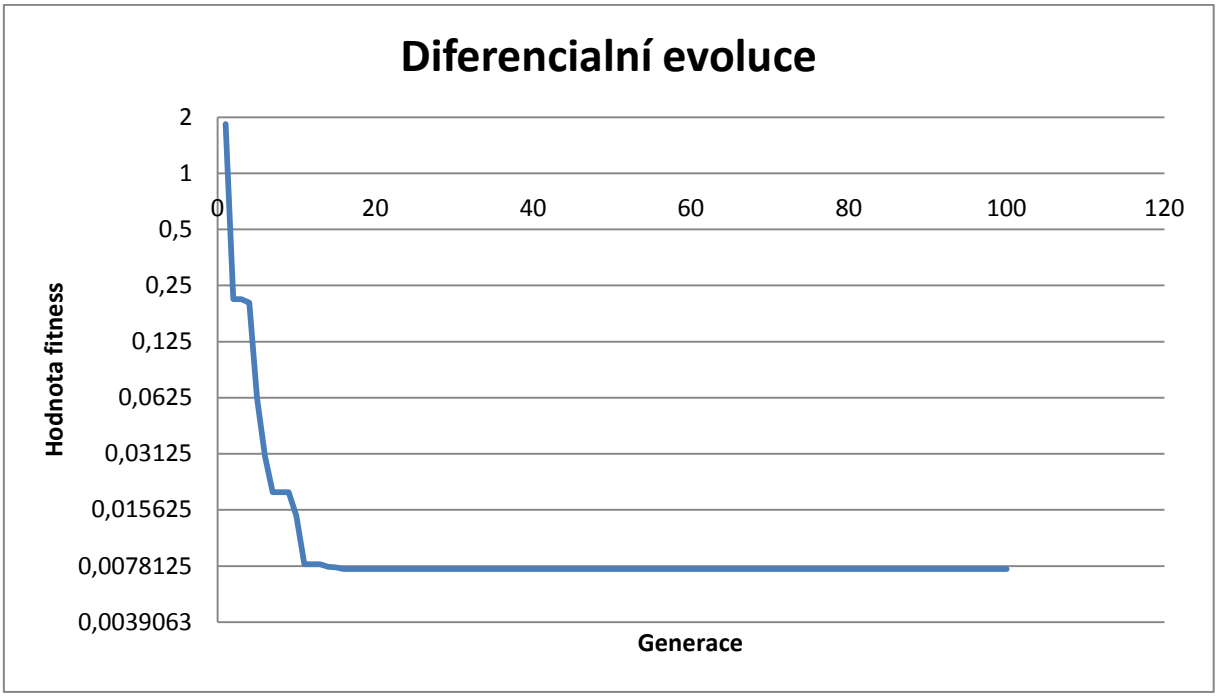
Obr. 8 – Graf průběh algoritmu evoluční strategie

Optimální řešení bylo nalezeno v hodnotě 0,0002441.

6.3 Testování diferenciální evoluce

Základní nastavení hodnot

- Velikost populace: 10
- Počet iterací: 100
- Mutace konstanta: 0,982
- Prah kříženi: 0,5



Obr. 9 – Průběh algoritmu diferenciální evoluce

Optimální řešení bylo nalezeno v hodnotě 0,0078125.

6.4 Porovnání algoritmů

Pro porovnání algoritmu jsem provedl 50 opakování jednotlivých průběhů algoritmů. Data jednotlivých testů můžete vidět v příloze na CD, nebo průměrné hodnoty v tabulce (Tab. 1).

	Diferenciální evoluce	**Evoluční algoritmus**	**Horolezecký algoritmus**
Průměrná fitness	0,029866706	0,019483898	0,079332
Průměrný čas (ms)	2,28	31,64	33,762

Tab. 1 – Průměrné hodnoty

7 Závěr

Cílem mé bakalářské práce bylo nalézt a vhodně modifikovat různé evoluční algoritmy tak, aby pokud možno efektivně řešily ukázkové NP úplné problémy. Vybral jsem si dva NP úplné problémy pro testování evolučních optimalizačních technik – Problém hledání globálního minima na testovacích funkcích a problém batohu.

V teoretické části jsem popsal třídy složitosti, nejpoužívanější NPC problémy a některé z evolučních optimalizačních technik (horolezecký algoritmus, evoluční strategii, genetický algoritmus, diferenciální evoluce a metodu simulovaného žíhání). Poznatky z této části jsem dále uplatnil v aplikačním řešení.

V praktické části jsem pro oba problémy nejdříve implementoval řešení pomocí evolučních optimalizačních technik (horolezecký algoritmus, evoluční strategie a diferenciální evoluce). Dále jsem provedl testování algoritmů na jednom problému - hledání globálního minima.

U problému hledání globálního minima na testovacích funkcích jsem zjistil, že použití diferenciální evoluce byla nejméně náročná na výpočet. Průměrná doba výpočtu byla 2,28 ms a průměrně naměřená fitness hodnota byla 0,02987. Evoluční algoritmus nalezl nejnižší průměrnou fitness hodnotu. Hodnota byla 0,01948 a průměrnou dobou 31,64 ms. Horolezecký algoritmus měl průměrnou fitness hodnotu 0,07933 a průměrnou dobu 33,762 ms. Pokud potřebujeme nalézt řešení v krátkém čase, použil bych algoritmus diferenciální evoluce. Pokud ale potřebujeme nalézt optimální řešení a nezáleží nám na čase, použil bych evoluční strategii.

8 Použitá literatura

[1] GAREY, Michael R, David S JOHNSON. Computers and intractability: a guide to the theory of NP-completeness. New York: W. H. Freeman, 1979, x, 338 s. ISBN 07-167-1045-5.

[2] DEVLIN, Keith. Problémy pro třetí tisíciletí. Praha : Argo a Dokořán, 2005. ISBN 80-7363016-8 (Dokořán), 80-7203-739-0 (Argo).

[3] KVASNIČKA, V; POSPÍCHAL, P. Evolučné algoritmy. První vydání. Bratislava: Slovenská technická univerzita, 2000. ISBN 80-227-1377-5.

[4] ZELINKA, Ivan. Umělá inteligence: v problémech globální optimalizace. 1. vyd. Praha: BEN, 2002, 189 s. ISBN 80-730-0069-5.

[5] HYNEK, Josef. Genetické algoritmy a genetické programování. 1. vyd. Praha: Grada, 2008, 182 s. ISBN 978-80-247-2695-3.

[6] SCHWARZ, Josef. Aplikované evoluční algoritmy: EVO. Brno: Fakulta informačních technologií, 2008, 100 s.

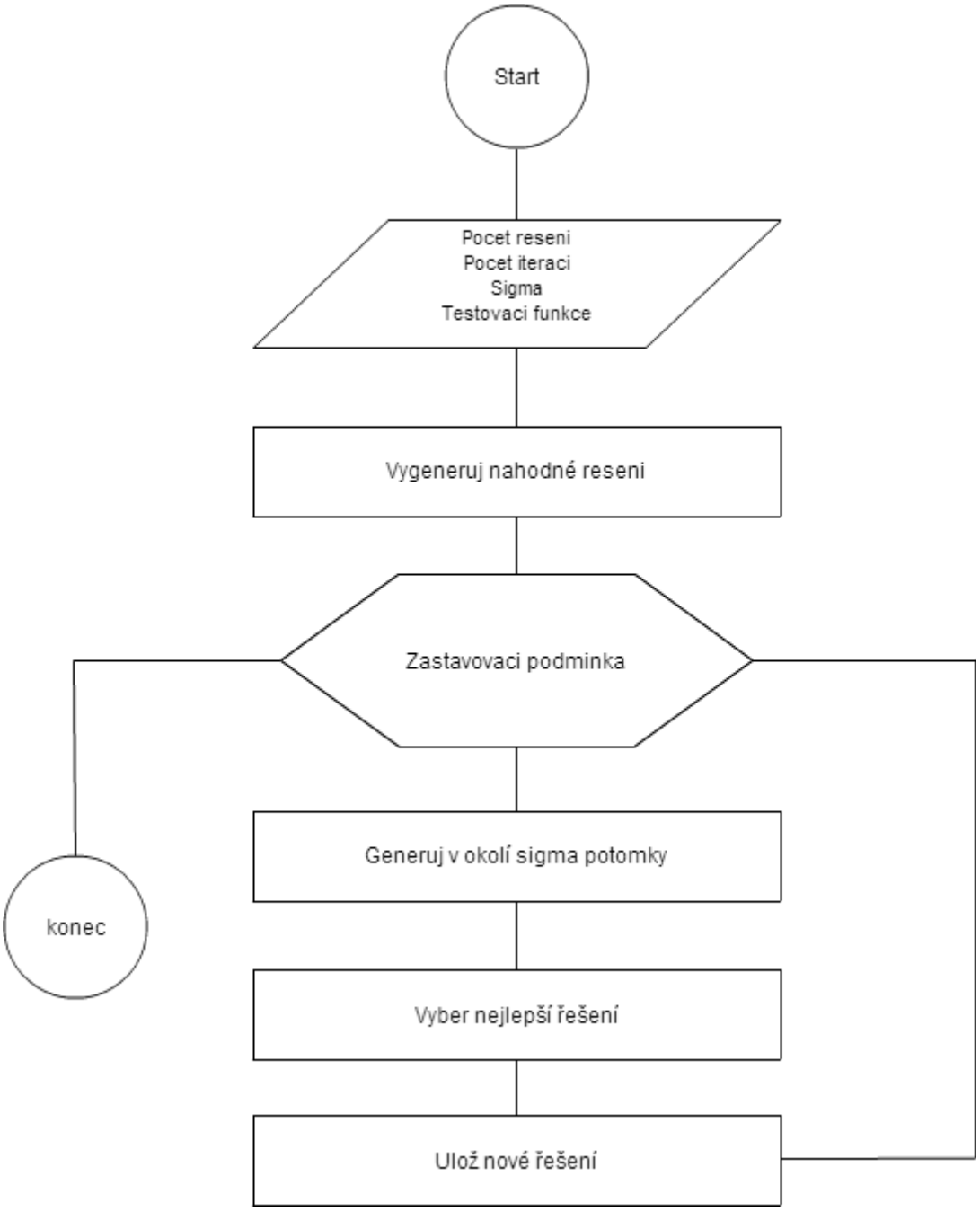
[7] TVRDÍK, J. Evoluční algoritmy, Ostrava 2004, s.35-39

[8] PANUŠ, Jan. Evoluční algoritmy v optimalizačních problémech veřejné správy. Pardubice 2008, Doktorská dizertační práce. Univerzita Pardubice, Fakulta ekonomicko správní.

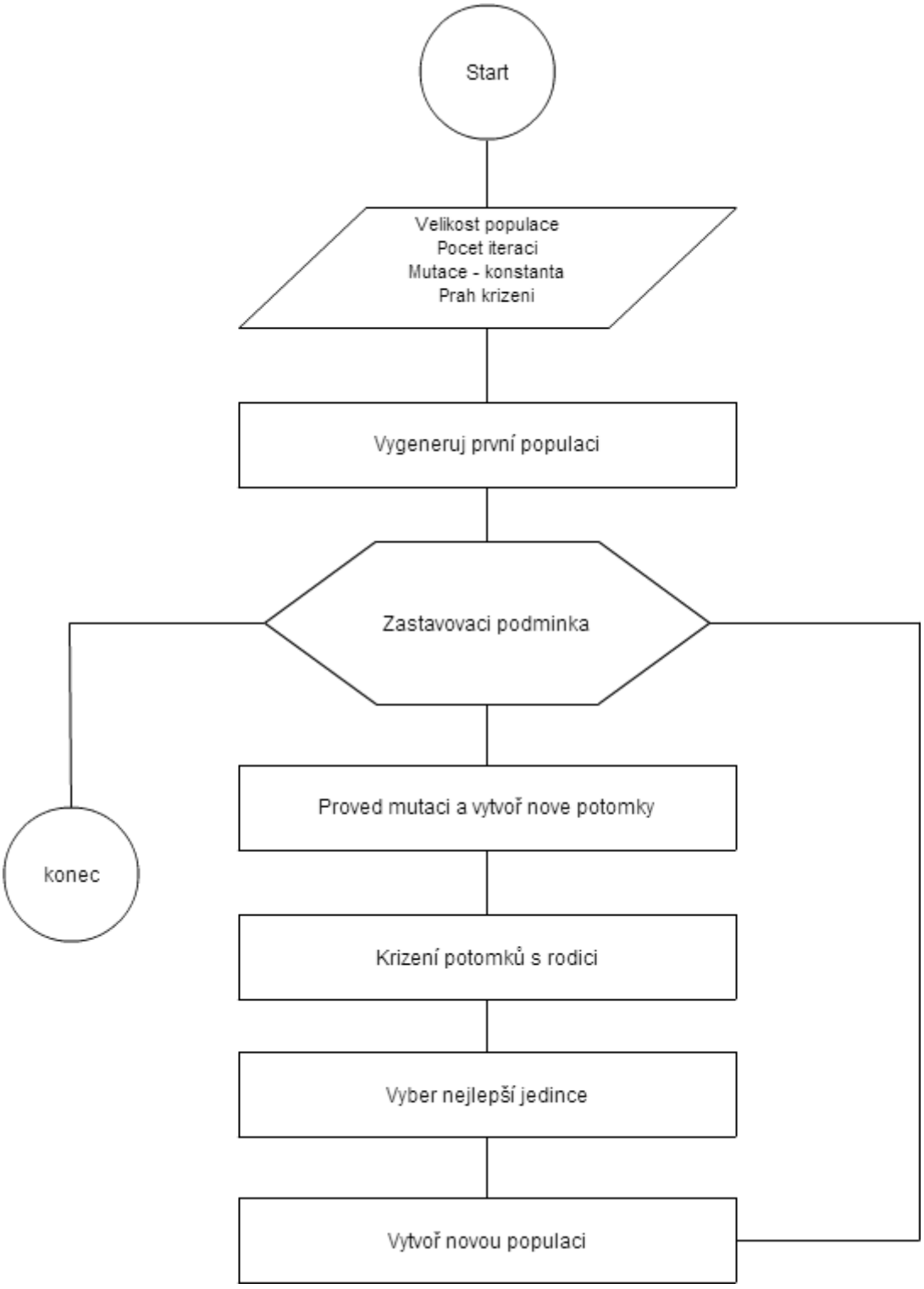
[9] OBITKO, M. Introduction to Genetic Algorithms: Genetic Algorithm. [online]. 1998, [cit. 2010-10-21]. Dostupné z WWW: http://www.obitko.com/tutorials/genetic-algorithms/ga-basic-description.php

9 Přílohy

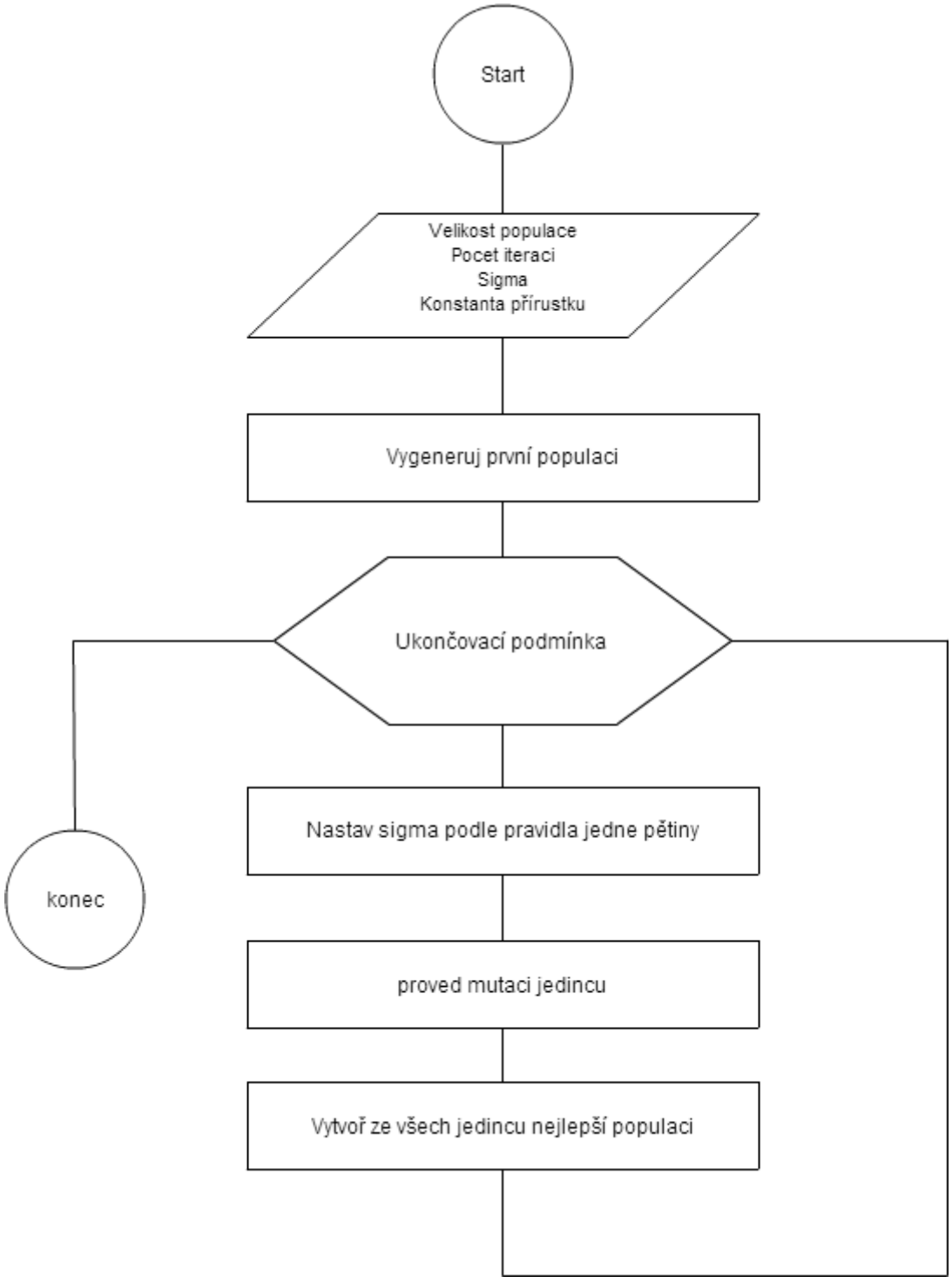
Příloha A – Vývojový diagram horolezecký algoritmus



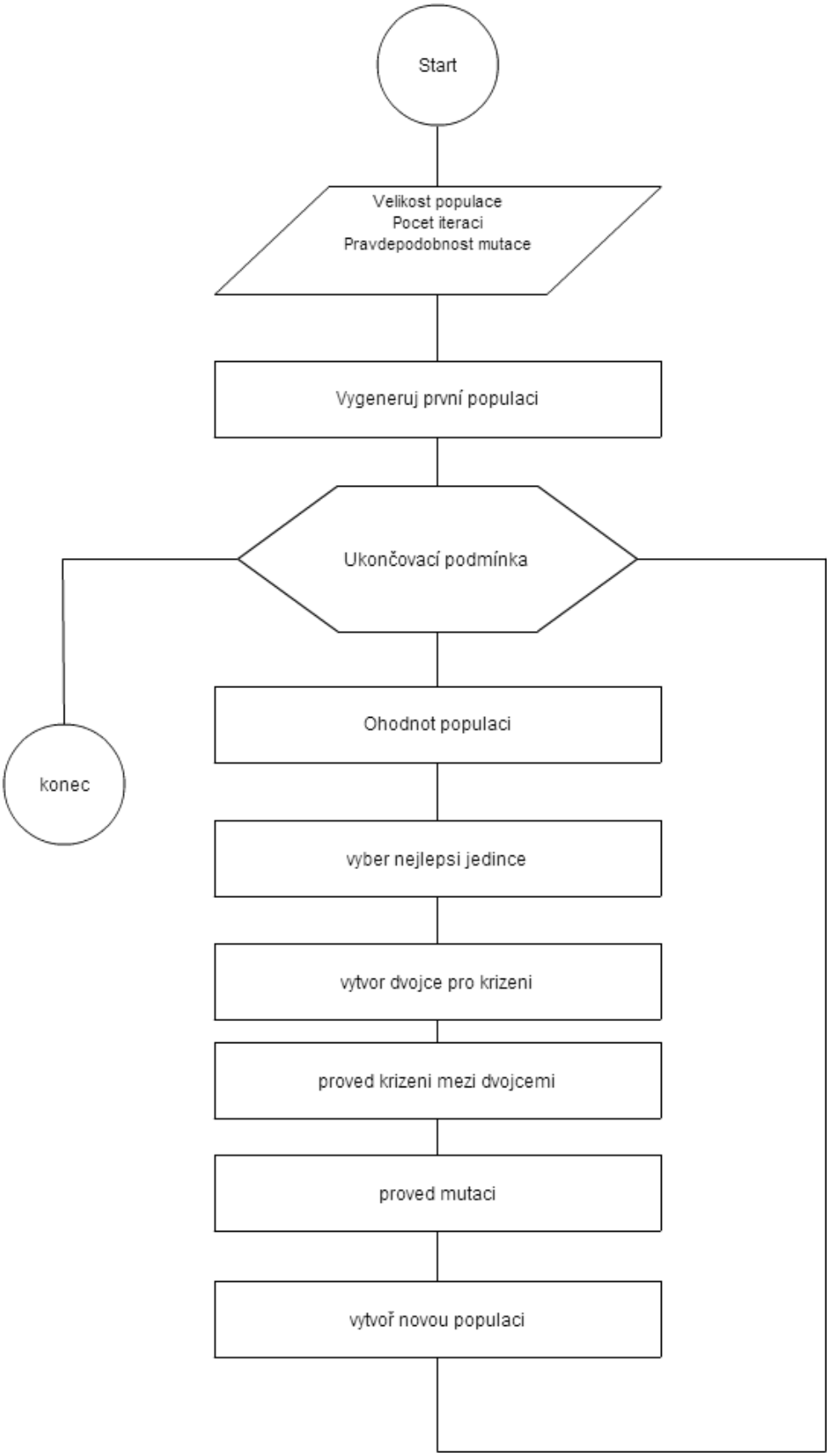
Příloha B – Vývojový diagram - diferenciální evoluce



Příloha C – Vývojový diagram Evoluční Strategie



Příloha D - Vývojový diagram Genetický algoritmus



Příloha E - CD