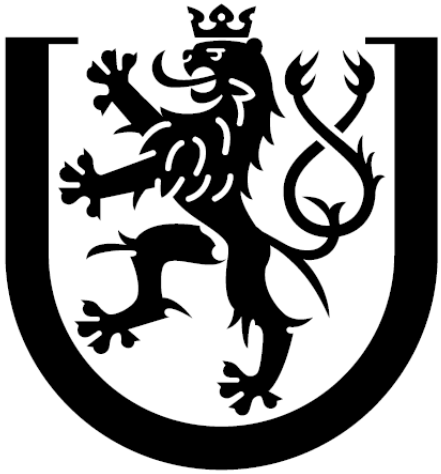

Fakulta aplikovaných věd
Západočeská univerzita v Plzni

Semestrální práce KIV/PC
Optimalizace funkce geneticým algoritmem

Ľubomír Bartoš (A16B0003P)
bartosl@students.zcu.cz

6. 1. 2019

Contents

1 Zadání

Naprogramujte v ANSI C přenositelnou konzolovou aplikaci, která bude hledat extrém funkce pomocí genetického algoritmu. Genetický algoritmus je technika, která napodobuje procesy známé z biologie a hledá tak optimální řešení zadaného problému. Podle zadaného problému se nadefinuje tzv. fitness funkce, jejíž hodnota vyjadřuje životaschopnost potomka a vhodnost řešení problému.

2 Základní princip genetického algoritmu:

1. Náhodně vytvoříme několik různých řešení zadané úlohy a jedincù.
2. Provedeme křížení jedincù.
3. Náhodně zmutujeme malý počet jedinců.
4. Ověříme, jak dobře jeden každý jedinec řeší náš problém tak, že pro každého získáme hodnotu fitness funkce.
5. Vytvoříme novou generaci.
6. Opakujeme body 2 až 5.

3 Úvod

Semestrální práce je napsána v jazyce c, obsahuje implementaci genetického algoritmu, který dokáže ladit vstupní hodnoty programu s optimalizovanou funkcí a hledat tak extrém možného fitness ohodnocení. Program s optimalizovanou funkcí, který hodnotil tvořené vstupní hodnoty, byl při vývoji jako black box, k dispozici byl jen vstup/výstup programu, proto nevím zda nejlepší výsledky, které jsem pomocí svého algoritmu dosáhl, jsou opravdu globálním extrémem. Mohu zatím jen z testování a pozorování říct, že algoritmus během několika (10-20 generací) generací spěje k lepšímu fitness, objeví se jedinec s určitou vysokou fitness a ostatní generace už ho pravděpodobně nepřekonají.

Má implementace se v několika ohledech odchyluje od zadání. Na funkčnost a správnost algoritmu by to nemělo mít vliv. Mutaci populace jsem v procesu posunul před tvoření potomků. V pokročilejších generacích by zmutovaní jedinci skoro nikdy nepřežili část umírání, protože by měli oproti zbytku malý fitness. Křížení binární části genu jsem implementoval tak, že se každý bit bere náhodně od matky nebo od otce, místo rozdělení a slepení genů na vhodném místě. Výběr jedinců k páření probíhá až po umírání slabých jedinců, navíc ze zbylých naprůměrných jedinců se bere jeden naprůměrný rodič (z přeživších), pro mírné zvýšení kvality populace, a druhý náhodně.

4 Princip tohoto genetického algoritmu:

Viz[1].

1. Náhodně vytvoříme několik různých jedinců, obsahujících geny s řešením zadané úlohy.
2. Otestujeme geny jedinců a přiřadíme jim fitness (ohodnocení).
3. Jedince s podprůměrným fitness odstraníme.
4. Náhodně zmutujeme zadané procento populace (pokaždé u jedince jen jeden typ genu).
5. Doplníme generaci o potomky existujících jedinců (křížením vybraných rodičů).
6. Opakujeme body 2 až 5.

4.1 Konfigurace programu

Program je konfigurovatelný pomocí tzv. metadata souboru, kde se definuje název a parametry (proměnné a konstanty) programu s optimalizovanou funkcí. Parametry načítá program s optimalizovanou funkcí z tohoto souboru, nepředávají se v příkazovém řádku. Počet jedinců v generaci nebyl blíže specifikován, vybral jsem tedy konstantní limit populace 100, definovaný v souboru structures.h. Při tvoření počáteční populace a při plození potomků se vždy tvoří jedinci, dokud populace nedosáhne definovaného limitu.

4.2 Formát souboru s metadaty

- Na první řádce se nachází vždy spustitelný soubor s laděnou funkcí, který nám dává fitness pro jedince.
- Následují data o konstantách a proměnných.
  - Proměnná je definovaná na dvou řádcích. První řádek obsahuje informace o intervalu proměnné a o jejím typu. Na následujícím řádku se nachází název proměnné a nějaká konkrétní hodnota. Pouze tento typ řádku se v průběhu programu mění. Program momentálně umí pracovat s celočíselnou a reálnou proměnnou.
  - Příklad reálné proměnné:
    - #_(10,18);R
    - a = 15.42069
  - Příklad celočíselné proměnné:
    - #_(0,500);Z
    - c = 420

  - Konstanty mají formát jako proměnné, bez prvního řádku:
    * b = 2

- Prázdné řádky jsou ignorovány.

5 Přeložení, spuštění a úklid

Algoritmus funguje na Linuxu (testováno na Linux Mint 19 Tara) i na Windows, musí se ale v metadata souboru změnit název spustitelného souboru. Pro Linux použijte *#_func*, pro Windows *#_func_win.exe*. Program je přeložitelný překladačem gcc. Program vypisuje informace o průběhu do konzole a do souborů ge.txt a val.txt.

- Instrukce pro překlad
  - *make*
- Instrukce pro spuštění
  - bez parametru procenta mutace
    * *./run <metadata soubor><počet generací>*
  - s parametrem procenta mutace
    * *./run <metadata soubor><počet generací>-m <procento populace ke zmutování>*
  - příklad
    * *./run func01_meta.txt 50 -m 20*
- Instrukce pro úklid (neuklízí soubory val.txt a gen.txt)
  - *make clean*

Po spuštění a dokončení běhu programu dostaneme dva soubory:

- **gen.txt** obsahuje zápis genu a fitness nejlepšího jedince z každé proběhlé generace
- do **val.txt** se zapisuje gen a fitness jedince při každém získávání fitness

6 Struktury a konstanty:

Pro pojmenovávání struktur a funkcí jsem se snažil hledat slova podobná strukturám a procesům přírody. Strukt environment představuje "prostředí" , neboli konfiguraci pro vývoj jedinců. Obsahuje také názvy souborů a programů, které potřebujeme pro spuštění laděné funkce. Strukt creature představuje jedince, který nese gen s proměnnými vstupními hodnotami laděné funkce.

Pro uložení populace jsem zvolil obousměrný seznam. Nový jedinci se přilepí vždy na konec seznamu. Pro uložení tzv. genu jedince (vstupní hodnoty do laděné funkce) jsem zvolil pole unionů(int, float). Podle uložené konfigurace pak rozlišujeme, zda je daný gen celoříselný nebo reálný.

6.1 Konstanty ve structures.h

- VARIABLE_TYPE_INTEGER 'Z'
  - typ proměnné z metadata souboru, představuje celé číslo
- VARIABLE_TYPE_REAL 'R'
  - typ proměnné z metadata souboru, představuje reálné číslo
- POPULATION_LIMIT 100
  - limit populace
- DEFAULT_MUTATION_RATE 5
  - pokud není procento mutace populace zadáno z příkazové řádky, je nastaveno na 5%

6.2 Proměnné ve structures.h

- environment
  - executable
    - soubor ke spuštění laděné funkce
  - meta_data_file
    - soubor s metadaty
  - variable_names
    - pole jedno-znakových názvů proměnných z metadata souboru
  - count_of_parameters
    - počet parametrů
  - parameters
    - pole jednoznakových typů proměnných z metadata souboru
  - intervals
    - pole intervalů k proměnným z metadata souboru

- struct creature - obousměrný spojový seznam
  - char name
    - číselný název složený z indexů jedincovo rodičů
  - union gene *gene - pole unionů:
    - binary - celočíselná proměnná
    - real - reálná proměnná

  - first
    * 1 pokud jedinec je první v seznamu populace
    * 0 pokud ne
  - last
    * 1 pokud jedinec je poslední v seznamu populace
    * 0 pokud ne
  - is_alpha
    * 1 pokud má jedinec nejlepší fitness
    * 0 pokud ne
  - fitness
    * ohodnocení jedince
  - next
    * ukazatel následujícího jedince v seznamu
  - previous
    * ukazatel předchozího jedince v seznamu

- union gene
  - int binary
    * celočíselná proměnná
  - float real
    * reálná proměnná

6.3 Zdrojový soubor nature.c

V tomto souboru souboru jsou definované zejména funkce simulující přírodní procesy přírody (vytvoření jedinců, množení, umírání, mutace, logika evoluce).

6.4 Zdrojový soubor config.c

V tomto zdrojovém souboru jsou definovány operace hledání nad seznamem populace, práce se soubory a laděnou funkcí, načtení konfigurace z metadata souboru.

7 Testování jedinců

Testování jedinců probíhá tak, že se v souboru s metadaty přepíšou proměnné podle genu jedince a spustí se program s optimalizovanou funkcí, který nám vrátí fitness pro jedince. Jedincovi uložíme fitness a zapíšeme záznam do val.txt.

8 Umírání jedinců

Jednou za generaci, těsně před pářením, umřou jedinci s podprůměrným fitness.

9 Mutace

Podle zadaného procenta mutace zjistíme počet jedinců pro zmutování. Poté vybereme náhodně daný počet jedinců a vytvoříme jim nový náhodný gen buď pro reálné nebo binární části. Viz[3].

10 Páření jedinců

Páření jedinců probíhá po období umírání. Hledání páru probíhá tak, že náhodný, nadprůměrný, jedinec dostane k páření náhodného jedince a křížením genů vytvoří potomka, který se přilepí na konec seznamu populace. Toto probíhá dokud není naplněn limit populace.

11 Křížení genů

- křížení celočíselného genu
  - každý bit bereme náhodně od matky/otce, viz[2]
- křížení reálného genu
  - uděláme průměr genů rodičů

12 Diagramy

Níže se nachází diagramy procesů, které mi přišly pro tento algoritmus důležité nebo zajímavé.

13 Závěr

Aplikace splňuje zadání, dokáže vylepšovat vstupní hodnoty podle fitness ohodnocení. Aplikace by mohla být rozšířena o další typy páření/přírodního výberu/křížení/mutace. Například zajímavé by bylo porovnat různé typy výběru párů (každý s každým, dominantní jedinec s každým) nebo výběru slabých jedinců k odstranění (momentálně zemřou všichni podprůměrní před pářením, v reálu by někteří podprůměrní chvíli dál šířili geny, ale ne tak dlouho jako naprůměrní jedinci). Valgrind vypisuje 0 chyb a žádný možný únik paměti.

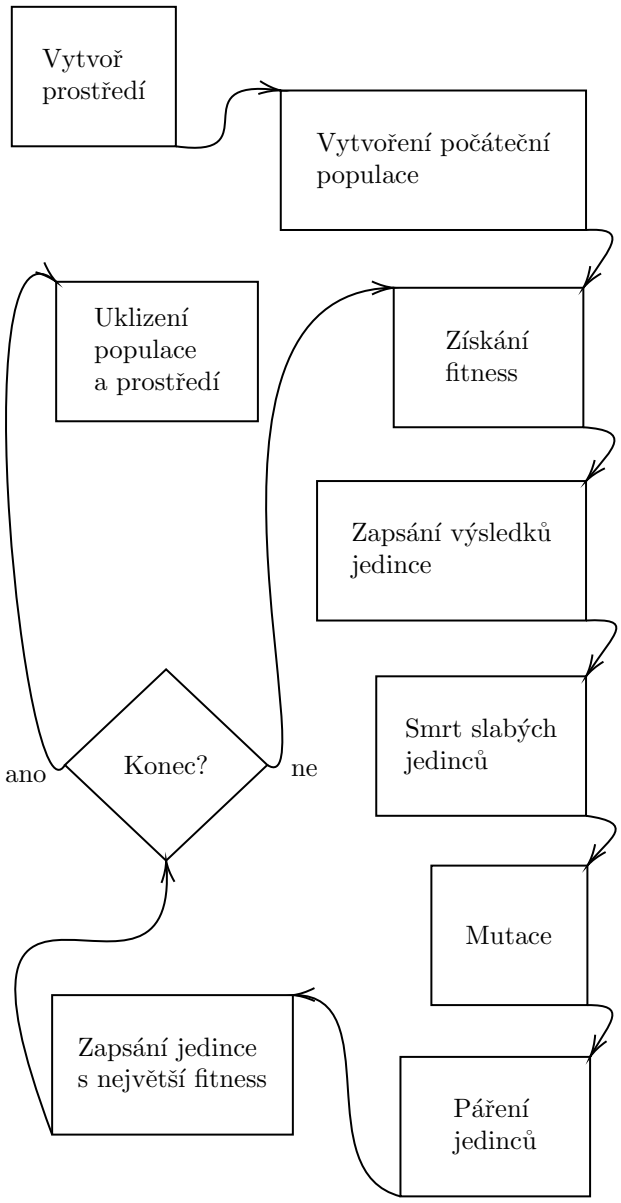

Figure 1: Hlavní procesy a tok algoritmu

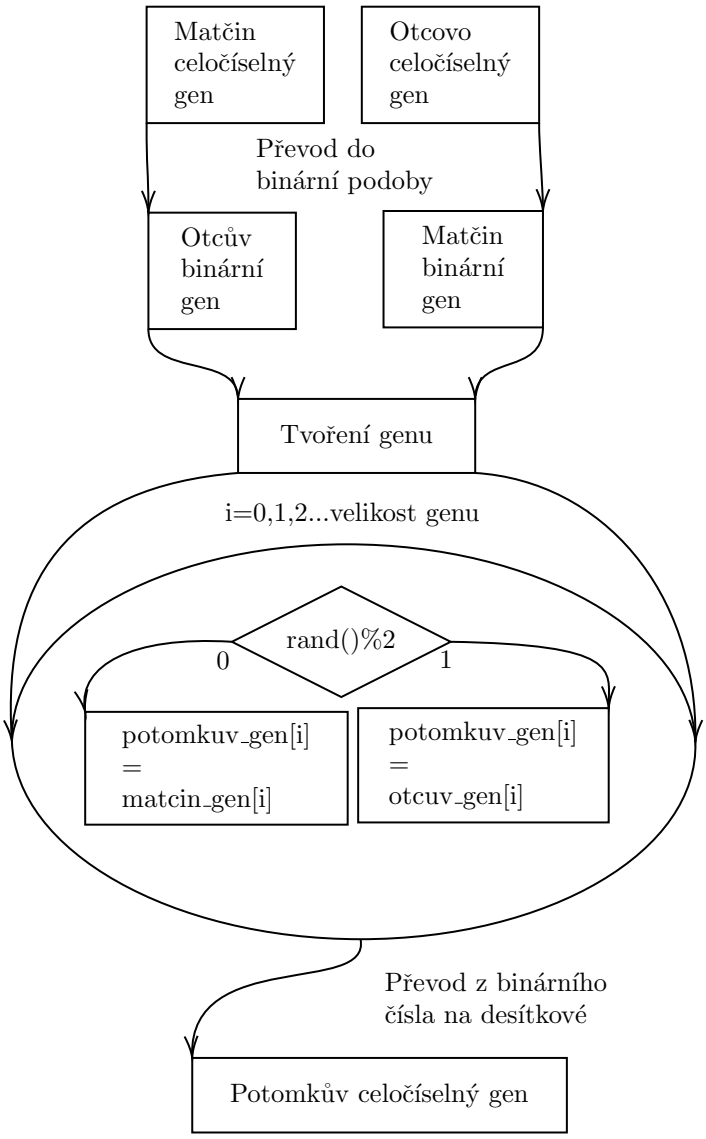

Figure 2: Křížení celočíselného genu

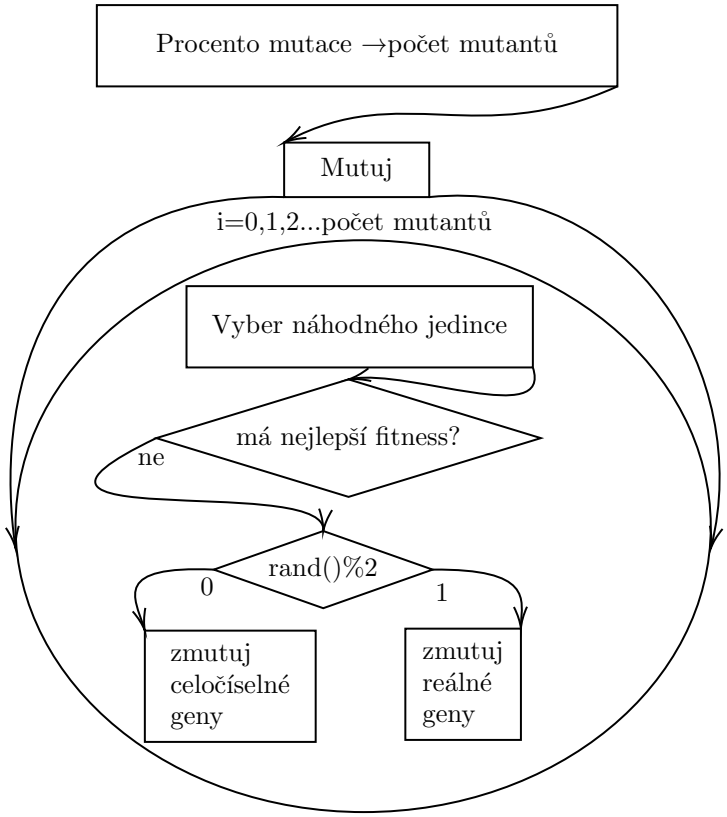

Figure 3: Mutace