Praktická aplikace genetického algoritmu

Petr Dvořáček xdvora0n@stud.fit.vutbr.cz

7. prosince 2014

1 Úvod

Tento dokument popisuje praktickou aplikaci genetických algoritmů, dále jen zkráceně GA. Existuje mnoho problémů, kde můžeme využít GA – nastavování parametrů, návrh HW, návrh obrazových operátorů atd. Cílem tohoto projektu je představení aplikace GA řešící problém obchodního cestujícího, kdy máme n měst a musíme projít všechna města tak, abychom urazili nejmenší vzdálenost. Jedná se o problém spadající do třídy NP úplných problémů.

2 Genetický algoritmus

Genetické algoritmy vychází z principů Darwinovy teorie evoluce a neodarwinismu. Podle Darwina je hlavní hnací silou evoluce tzv. přirozený výběr, takže slabší jedinci umírají a ti silnější přežívají a rozmnožují se.

Genetické algoritmy pracují analogicky. Nejdříve se vytvoří počáteční populace P_0 a to buď náhodně, nebo pomocí nějaké heuristiky. Při heuristice se nejčastěji používájí hotová řešení. V každé generaci i se vyberou dva rodiče $R \subseteq P_i, |R| = 2$. Výběr probíhá nejčastěji podle selektivních operátorů turnaj či ruleta, vizte níže. Z těchto dvou rodičů vznikne potomek (nebo více potomků) a to aplikací genetických operátorů křížení a mutace. Takovýmto způsobem se vytvoří množina potomků S. Další novou generaci přeživších P_{i+1} vybereme z množiny $S \cup P_i$. Výběr může probíhat deterministicky, vybere se $|P_{i+1}|$ nejlepších jedinců z množiny $S \cup P_i$. Nebo také nedeterministicky a to podle turnaje, nebo rulety.

Genetické operátory křížení a mutace slouží k tvorbě nového jedince – potomka. Jejich aplikace může probíhat třeba takto: nejdříve se oba rodiče zduplikují a tyto dvě kopie budou značit dva nové potomky. Křížením se rozumí, že si potomci vymění část své genetické informace – chromozomu. Křížení může probíhat jednobodově, či vícebodově. Podrobně je operace křížení znázorněna v tabulce 1. Mimojiné existují i jiné druhy křižení ku příkladu uniformní křižení. Odlišností u tohoto operátoru je to, že se kříží neurečný počet genů mezi dvěma jedinci.

Rodič A	0	1	2	3	10	9	2	4
Rodič B	8	7	6	2	11	7	9	1

→

Potomek A	8	7	2	3	10	9	2	1
Potomek B	0	1	6	2	11	7	9	4

Tabulka 1: Znázornění dvoubodového křížení

Aby byla zachována genetická pestrost jedinců, použije se na potomky operátor mutace. Mutace mají velký vliv na výsledného jedince. Nejsou-li povoleny vůbec, pak algoritmus nemusí konvergovat k vhodnému řešení. Tento fakt zároveň platí pro křížení. Aplikace operátoru mutace je znázorněna v tabulce 2.

Před mutací	8	7	2	3	10	9	2	1
Po mutaci	8	42	2	3	10	9	2	1

Tabulka 2: Znázornění mutace

Každý jedinec po svém vytvoření je ohodnocen fitness funkcí, která udává jeho kvalitu. Fitness funkce se implementují různými způsoby, záleží totiž na řešeném problému. Máme-li ohodnoceny všechny jedince z $S \cup P_i$, potom musíme vybrat novou množinu $P_{i+1} \subseteq S \cup P_i$. K výběru nám poslouží operátory pro selekci jedinců – turnaj, či ruleta. Tyto operátory můžou být rovněž použity při výběru rodičů R.

Při turnaji se nejdříve vyberou dva náhodní jedinci, ale může být jich více. V turnaji vyhraje ten jedinec, který má *lepší* fitness hodnotu. Otázkou ale zůstává, zda-li je potřeba fitness hodnotu v průběhu evoluce minimalizovat, či maximalizovat. Tento problém spadá do implementace fitness funkce a na řešeném problému. Nechť tedy minimalizujeme fitness funkci, potom se do výsledné množiny vybere jedinec s nižší fitness hodnotou. V případě maximalizace by se postupovalo analogicky.

Selekce pomocí rulety probíhá tak, že se nejdříve každému jedinci vypočte tzv. podíl na ruletě, který odpovídá jeho fitness. Podíl může být určen podle vzorce:

$$p_i = \frac{f_i}{\sum_{i=0}^{N} f_i}$$

, kde f_i je fitness hodnota jedince, $\sum_{i=0}^{N} f_i$ je pak suma fitness hodnot všech jedinců. Podíly nám vytvoří množinu intervalů $I = \{I_0, I_1 ... I_N\}$, kde $I_0 = \langle 0, p_0)$, $I_1 = \langle p_0, p_0 + p_1) \ldots I_N = \langle \sum_{i=0}^{N-1} p_i, 1)$. Výpočet můžeme zobecnit na $I_K = \langle \sum_{i=0}^{K-1} p_i, \sum_{i=0}^{K} p_i)$ pro $K \in \{0, 1 ... N\}$. Tyto intervaly se nepřekrývají a jejich sjednocení nám dá množinu realných čísel v intervalu $\langle 0, 1)$. Tedy platí $I_0 \cup I_1 \cup ... \cup I_N = \langle 0, 1)$ a zároveň $\forall i, j \in \{0, 1, ... N\} : I_i \cap I_j = \emptyset \wedge i \neq j$. Získání jedince z rulety probíhá následovně. Mějme náhodnou hodnotu $r \in \langle 0, 1)$, potom pro výsledného jedinece s indexem n platí: $r \in I_n$.

2.1 Problém ochodního cestujcího

Problém obchodního cestujcího se někdy zkráceně uvádí jako TSP, z anglického Travelling Salesman problem. Obchodní cestující má problém. Potřebuje nalézt nejkratší cestu takovou, že projde všechna města, která jsou od sebe různě vzdálená. Problém může být převeden na hledání nejmenší kružnice v úplném grafu, kde jednotlivé uzly představují města, a kde vzdálenost mezi dvěma městy je znázorněna jako ohodnocená hrana v daném grafu. Pokud graf není úplný (tj. chybí někde hrana), můžeme jej převést pomocí Dijkstrova algoritmu a tím získáme úplný graf. Tato úloha spadá do NP těžkých problémů. Důkaz je k dispozici v literatuře [1].

Chromozóm pro TSP je pak definován jako permutace měst – uzlů grafu. To znamená, že žádný gen se nevyskytuje v chromozomu dvakrát. Znázorněno je to v tabulce 3. Ze zakódovaní vyplývá, že pro n měst existuje $n!$ řešení. Rovněž je zřejmé, že platí rovnice $distance(1, 2, 3, .., n) = distance(n, .., 3, 2, 1)$, kde jednotlivá čísla $1..n$ odpovídají městům. Jinými slovy řečeno, když projdeme všechna města dopředným průchodem, urazíme stejnou vzdálenost, jako kdybychom je procházeli pozpátku. Dále můžeme začít průchod z libovolného města a těch je právě n, tzv. rotace permutace. To znamená, že musíme projít nejméně $(n-1)!/2$ možných kombinací měst.

Díky zakódovaní chromozomu do permutace musí být upraveny genetické operátory křížení a mutace, abychom vždy získali validní řešení. Křížení jedinců lze provést několika způsoby. Jedním z nich je PMX, které je dostupné na přednáškách [2], který můžeme implementovat následovně. Dojde

Brno	Frýdek	Praha	Místek	Plzeň	Ostrava
0	3	5	2	1	4

Tabulka 3: Znázornění chromozomu permutace. První řádek je chromozom reprezentován řetězci. Druhý řádek odpovídá jejich celočíselným hodnotám. V programu se setkáte s druhým zakódováním.

Rodič A	0	4	5	2	1	3
Rodič B	1	3	5	2	0	4

Potomek A	0	4	5	2	0	4
Potomek B	1	3	5	2	1	3

Křížené geny $K_A = \{0, 4\}$, $K_B = \{1, 3\}$

Nekřížené geny $P_A = \{0, 4, 2, 5\}$, $P_B = \{0, 1, 2, 5\}$

$Zmenit_A = \{0, 4\}$, $Zmenit_B = \{1, 3\}$

Potomek A	0	4	5	2	0	4
Potomek B	1	3	5	2	1	3

$Zmenit_A = \{4\}$, $Zmenit_B = \{3\}$

Potomek A	1	4	5	2	0	4
Potomek B	0	3	5	2	1	3

$Zmenit_A = \emptyset$, $Zmenit_B = \emptyset$

Potomek A	1	3	5	2	0	4
Potomek B	0	4	5	2	1	3

Tabulka 4: Krokové znázornění křížení chromozomu, který je definován jako permutace.

ke klasickému křížení s tím rozdílem, že se uloží úseky genů, které byly kříženy, a které naopak nebyly. Vytvoří se tedy pomocné množiny K_A, K_B, které značí křížené geny a P_A, P_B, které značí nekřížené geny. Z těchto množin se vytvoří množiny $Zmenit_A = P_A \cup K_A$ a $Zmenit_B = P_B \cup K_B$. Pomocí prvků z množiny $Zmenit$ se naleznou duplikáty, které se nacházejí v **nekříženém** úseku genu. Tyto geny se mezi potomky prohodí a odeberou se z mnozin $Zmenit$. Iterativně procházíme chromozóm a zaměňujeme duplikáty do té doby než $Zmenit = \emptyset$. Tato situace je znázorněna v tabulce 4. Mutace se provede záměnou dvou měst, tak jak je to znázorněno v tabulce 5.

Je vhodné si všechny vzdálenosti mezi jednotlivými městy předpočítat a uložit do vzdálenostní matice. Fitness hodnota kandidátního řešení je pak rovna vzdálenosti průchodu mezi městy včetně průchodu mezi posledním a prvním genem chromozomu. Je zřejmé, že nejlepší kandidátní řešení z množiny populace je to nejkratší. Takže fitness hodnotu zmenšujeme.

3 Implementace aplikace

Pro implementaci programu byl zvolen programovací jazyk Java jednak kvůli snadné implementaci GUI, ale hlavně jednak kvůli její portability na různé platformy. Program se skládá z pěti souborů – modulů. Soubory `SfcGA.java` a `SfcGA.form` slouží hlavně pro GUI a obsahují hlavní třídu výsledné aplikace. Tento soubor komunikuje s třídou vytvořenou souborem `TSP.java`, kde je implementován problém obchodního cestujícího podle popisu z předchozí kapitoly. Tato třída využívá třídy `SelectionType` k libovolnému typu selekce a `Chromosome`, které se nachází ve stejnomenných souborech. Přeložení programu bylo provedeno v programovacím prostředí Netbeans.

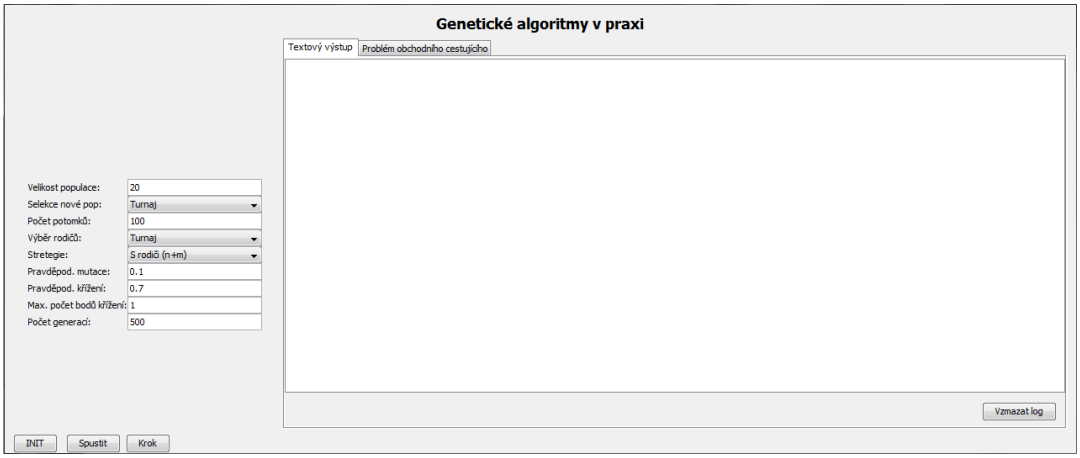
0	3	5	2	1	4
0	1	5	2	3	4

Tabulka 5: Mutace chromozomu, který je definován jako permutace.

4 Použití aplikace

K spuštění aplikace je nutne míti nainstalovánu Javu. K spuštění programu Vám poslouží příkaz `java -jar proj.jar` na libovolném operačním systému podporující javu.

Po spuštění programu se Vám objeví hlavní okno aplikace. Na obrázku 1 vidíte, že v levé části okna se nachází nastavení, které je již přednastaveno. V pravé části je pak textový výstup.



Obrázek 1: Hlavní okno aplikace

4.1 Použití programu

V levé části uživatelského rozhraní leží nastavení GA. Můžete nastavit velikost populace (přirozené číslo), výběr nové populace (turnaj, ruleta, náhodný, či výběr podle nejlepších fitness hodnot), má-li nová populace vzniknout vybraním pouze potomků nebo má-li zahrnout i rodiče, počet potomků (přirozené číslo), výběr rodičů (turnaj, ruleta nebo náhodný), pravděpodobnost mutace a křížení (čísla v intervalu $\langle 0, 1 \rangle$), počet bodů křížení a počet generací (přirozená čísla).

4.2 Spuštění algoritmu

Před spuštěním běhu je nutno definovat uzly grafu. Klepnutím na záložku *Problém obchodního cestujícího*, která se nachází na horní liště, se dostanete k návrhu uzlů grafu. Uzel přidáte klepnutím na libovolné místo v zeleném respektive bílém poli. Může se stát, že při přepínání záložek uzly grafu zmizí, proto tyto případy je zde záchrané tlačítko s vlídným názvem *překreslit*.

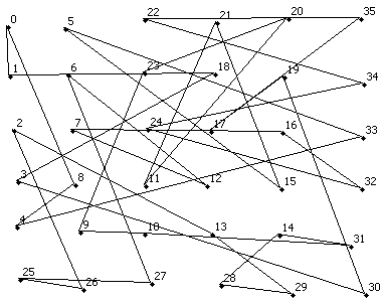
Máme-li definované uzly, potom inicializujeme algoritmus pomocí tlačítka *init* a můžeme jej spustit tlačítkem *spustit*. Rovněž můžeme algoritmus spouštět krokově. Tlačítko *init* se potom využívá k inicializaci dalších běhů GA s různými parametry. Chcete-li inicializovat nový běh s jinými uzly, je nutno klepnout na tlačítko *reset*. Čímž se vymažou všechny uzly grafu.

4.3 Příklad algoritmu

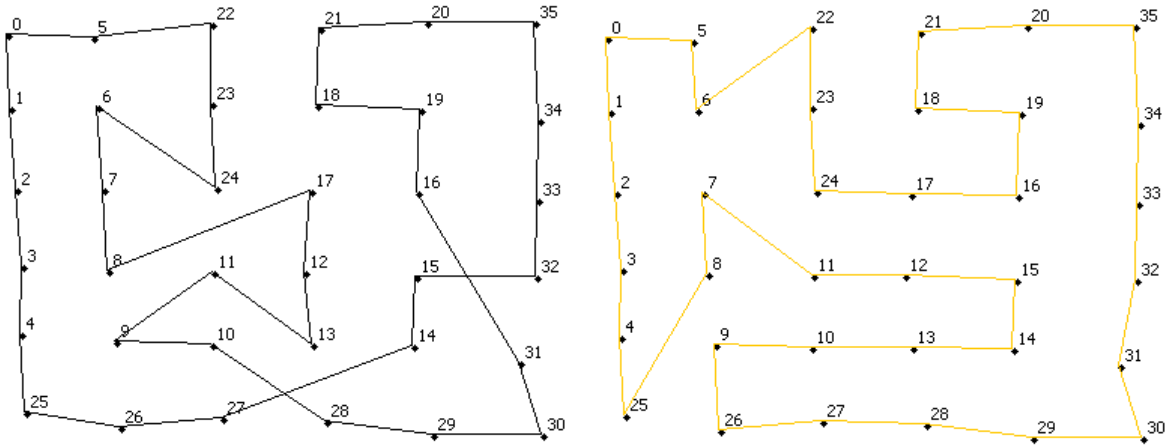
V této sekci naleznete příklad běhu algoritmu. Byly použity přednastavené hodnoty, tak jak jsou uvedeny v programu. Pro příklad byl využito rozmístění uzlů do matice, kde lze optimální řešení snadno nalézt. Toto rozmístění uzlů můžete spatřit na obrázku 2, kde je mimojiné zachycena inicializace cesty. Zároveň můžete vyčíst, že existuje celkem 36! řešení a tudíž je nutno projít minimálně

$35! * 0.5$ řešení (cca. $5.16 * 10^{39}$). Chtěl bych podotknout, že je nutno spustit běh GA několikrát, aby se nalezlo dostatečně kvalitní řešení.

Bylo provedeno 50 nezávislých běhů pro 500 generací o 80ti potomcích. Takže bylo ohodnoceno téměř 2000000 řešení, kde nejlepší z nich dosahovalo fitness hodnoty 2259. Viz obrázek 3. Je zřejmé, že toto řešení, není optimální, avšak kvůli komplexnosti problému je vhodnější použití GA než konvenčních praktik, protože $2 * 10^6 < 35! * 0.5$.



Obrázek 2: Inicializace TSP



Obrázek 3: Vlevo kandidátní řešení po prvním běhu algoritmu s fitness hodnotou 2784, vpravo nejlepší nalezené řešení s fitness hodnotou 2259.

5 Závěr

Bylo představeno použití GA pro řešení problému obchodního cestujícho. Řešení TSP dává uspokojivé výsledky z pohledu na komplexnost problému, avšak tyto výsledky nejsou optimalní. V budoucí práci se můžeme zaměřit na vyzkoušení různých druhů křížení pro TSP a provést porovnání s jinými metodami výpočtu, než jsou genetické algoritmy.

Reference

[1] *Češka V.* Skripta k předmětu Teoretická informatika

[2] *Zbořil F. st.:* přednášky k předmětu Soft Computing