

Sieć kanałów – Algorytmy genetyczne

Dominik Czarnota

Luty 2014

1. Opis programu

Program został napisany w języku C++ przy użyciu najnowszego standardu oraz skompilowany kompilatorem g++ (GCC) w wersji 4.8.1, z flagami:

-Wall -Wextra -Wpedantic -std=c++11 -O3, na systemie Windows 7.

Program składa się z czterech głównych modułów:

- main.cpp* – zawiera najważniejsze ustawienia (liczebność populacji, liczba generacji, prawdopodobieństwo mutacji i krzyżowania) oraz uruchamiania algorytmu
- Individual.h*, *Individual.cpp* – zawiera klasę osobnika
- Population.h*, *Population.cpp* – zawiera klasę populacji oraz metody operujące na niej
- Objective.h*, *Objective.cpp* – zawiera strukturę miasta oraz celu (który służy do obliczania funkcji dostosowania)

Dodatkowo do testowania wyników programu, wykorzystałem proste skrypty napisane w pythonie 3, znajdujące się w folderze *skrypty*:

- generateMap.py* – generuje plik mapy *param.ini*
- pyplot.py* – generuje podgląd mapy wraz z rurociągami dla zadanego *param.ini* oraz *czarnota.txt*. Wymaga biblioteki *matplotlib*.
- OdpalaczProgramow* – skrypt służący do włączania programów znajdujących się w *Execs/* z różnymi parametrami w celu porównania różnych metod. Skrypt zapisuje wyniki do folderu *TestCases*, które następnie możemy zobaczyć na wykresie uruchamiając skrypt *WykresProgramow.py* (generuje wykres zależności długości rurociągów od $1 - \text{liczba_nawodnionych_miast} / \text{liczba_miast}$). Skrypt wymaga bibliotek *shutil* oraz *subprocess*.

2. Zastosowane kodowanie

Osobnik kodowany jest jako tablica 10 dodatnich liczb całkowitych:

```
typedef unsigned int SIZE_T;  
const int MAX_PIPES_COUNT = 10;  
SIZE_T _genotype[MAX_PIPES_COUNT];
```

Każda liczba różna od zera oznacza jeden rurowciąg. Zero interpretujemy jako brak rurowciągu.

Dany rurowciąg kodowany jest na poszczególnych bitach liczby:

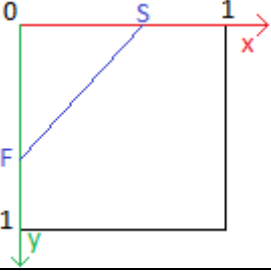
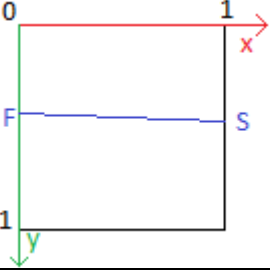
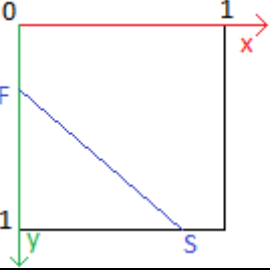
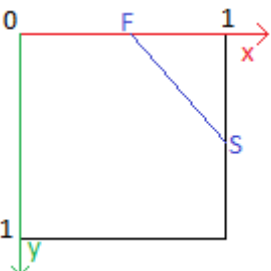
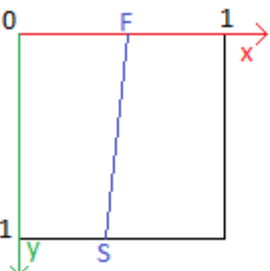
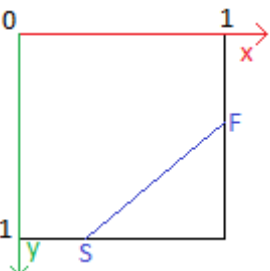
<i>firstVal</i> (16 bitów)	<i>secondVal</i> (16 bitów)
----------------------------	-----------------------------

Liczby *firstVal* oraz *secondVal* zapisane są w kodowaniu binarnym, oznaczają odpowiednie współrzędne końców rurowciągu. Dodatkowo, cała liczba koduje to, z której ściany na którą prowadzimy rurowciąg, co realizuje kod:

```
SIZE_T pointsDirection = ind._genotype[gene] % 6;
switch (pointsDirection) { ... }
```

W zależności od *pointsDirection* dostajemy takie przykładowe rurowciągi:

(na rysunkach: F – *firstVal*, S – *secondVal*)

<p><i>pointsDirection</i> == 0</p> <p><i>x1</i> = 0 <i>y1</i> = <i>firstVal</i> <i>x2</i> = <i>secondVal</i> <i>y2</i> = 0</p> 	<p><i>pointsDirection</i> == 1</p> <p><i>x1</i> = 0 <i>y1</i> = <i>firstVal</i> <i>x2</i> = 1 <i>y2</i> = <i>secondVal</i></p> 	<p><i>pointsDirection</i> == 2</p> <p><i>x1</i> = 0 <i>y1</i> = <i>firstVal</i> <i>x2</i> = <i>secondVal</i> <i>y2</i> = 1</p> 
<p><i>pointsDirection</i> == 3</p> <p><i>x1</i> = <i>firstVal</i> <i>y1</i> = 0 <i>x2</i> = 1 <i>y2</i> = <i>secondVal</i></p> 	<p><i>pointsDirection</i> == 4</p> <p><i>x1</i> = <i>firstVal</i> <i>y1</i> = 0 <i>x2</i> = <i>secondVal</i> <i>y2</i> = 1</p> 	<p><i>pointsDirection</i> == 5</p> <p><i>x1</i> = 1 <i>y1</i> = <i>firstVal</i> <i>x2</i> = <i>secondVal</i> <i>y2</i> = 1</p> 

3. Operatory genetyczne

a) Mutacja

Usuwa lub dodaje losowy rurowciąg osobnika.

Testowanie różnych wersji pokazało, iż zmienianie losowych bitów jest dużo mniej efektywne niż zastosowana metoda, ze względu na zastosowane kodowanie rurowciągu – zmiana jakiegokolwiek bitu może spowodować zmianę ścian, na i z których jest prowadzony rurowciąg oraz zmianę punktu położenia na którejs

z nich, co zazwyczaj okazywało się zbyt dużą ingerencją w osobnika. Taka sytuacja wymagała bardzo niskiego prawdopodobieństwa mutacji, rzędu 0.001 oraz dodatkowego operatora genetycznego, który mógłby usunąć/stworzyć nowy rurociąg.

b) Krzyżowanie

Wykorzystane zostało krzyżowanie jednopunktowe – pierwszych 5 liczb z tablicy `_genotype[]` dwóch osobników jest zamienianych ze sobą.

Testowano również krzyżowanie, które zamienia losowe rurociągi, ale przyniosło ono gorsze rezultaty oraz z uwagi na „branch prediction fail” powodowało wykonanie mniejszej ilości generacji programu.

4. Selekcja

Zastosowana metoda selekcji ma dwie wersje:

a) Gdy populacja jest „słaba” (liczba generacji < 3000)

Do nowej populacji brane jest 10% nowo stworzonych osobników

b) Gdy populacja jest „dobra” (liczba generacji ≥ 3000)

Do nowej populacji brane jest 10% najlepszych osobników

Procent osobników wybieranych wyżej wymienioną metodą regulowana jest wartością zmiennej `ELITIST_PERCENTAGE` w pliku `Population.h`.

Reszta populacji (90%) wybierana jest metodą selekcji rankingowej.

Testowana została również metoda selekcji ruletkowej, ale nawet z różnymi parametrami, przyniosła dużo gorsze wyniki.

5. Funkcja dostosowania

Obliczanie funkcji dostosowania realizowane jest w klasie *Objective*, w metodzie *CountFitness (Individual& ind)*, według następującego wzoru:

$fitness = A + 0.8 * B$, gdzie:

$A = (max_długość_rurociągów - długość_rurociągów) / max_długość_rurociągów$

$B = liczba_nawodnionych_miast / liczba_wszystkich_miast$

Zastosowane zostało również negowanie liczby nawodnionych miast (waga 0.8), ze względu na to, iż bardzo łatwo o mapę, w której jedno miasto spowoduje duży wzrost długości rurociągów.