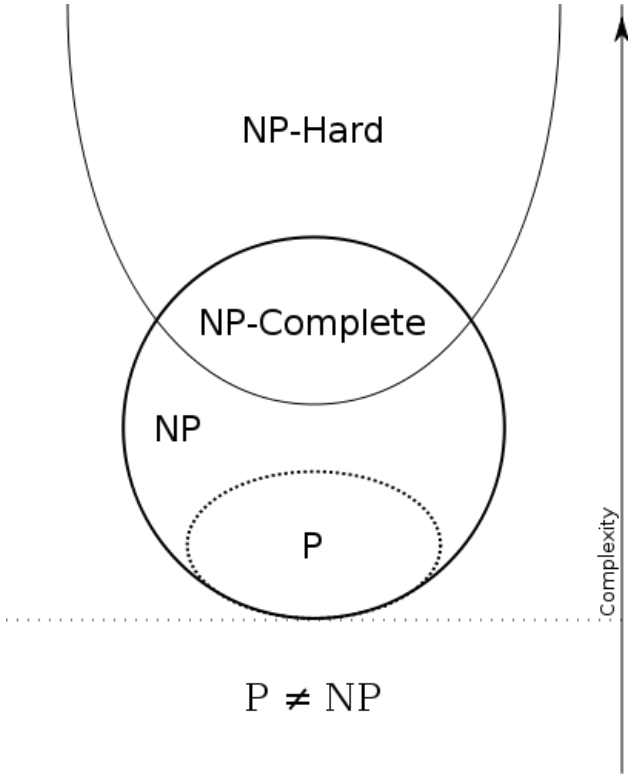
08.11.2018r.

Zadanie domowe nr 1
Algorytm genetyczny

Dawid Ćwik
Informatyka II
gr. 1
238137

1. Opis problemu

Sprawozdanie dotyczy analizy problemu spełnialności formuł w koniunkcyjnej postaci normalnej. Badany przeze mnie przykład dotyczy 3 iterałów. Polega on na znalezieniu wektora, który spełnia koniunkcje klauzul zawierających alternatywy zmiennych.



Przypadek 3-SAT, w przeciwieństwie do 1-SAT oraz 2-SAT, nie ma rozwiazania w P. jest on NP-zupełny, najtrudniejszy do rozwiązania z klasy NP. Różnica pomiędzy problemami P i NP polega na tym, że w przypadku P **znalezienie** rozwiązania ma mieć złożoność wielomianową, podczas gdy dla NP **sprawdzenie** podanego z zewnątrz rozwiązania ma mieć taką złożoność.

2. Wykorzystane technologie i narzędzia

Przy pracy nad problemem korzystałem z języka **R** oraz dedykowanego programu **Rstudio**. Funkcja generująca algorytm genetyczny, przedstawiona na labolatoriach, znajduje się w bibliotece ***genalg – rbga.bin*** *(R Based Genetic Algorithm (binary chromosome))*.

3. Kod programu

Główną funkcją odpowiadająca za ostateczny wynik algorytmu genetycznego jest metoda fitness. Zawiera ona w sobie informacje, czy dany chromosom wygenerowany przez algorytm, spełnia nasze wymagania.

```
function(chromosome) {
    counter = 0
    result = FALSE
    for(i in 1:nrow(values)){
        row <- values[i,]
        row_1 <- abs(row[1])
        row_2 <- abs(row[2])
        row_3 <- abs(row[3])

        x1 = chromosome[row_1]
        if(row[1] < 0){ x1 = !x1 }
        x2 = chromosome[row_2]
        if(row[2] < 0){ x2 = !x2 }
        x3 = chromosome[row_3]
        if(row[3] < 0){ x3 = !x3 }

        result = x1 | x2 | x3
        if(!result){
            counter = counter + 1
        }
    }
    return(counter)
}
```

W tym przypadku prezentowana jest jedna z metod użytych w moim projekcie. Realizowana jest za pomoca komendy biblioteki *genalg*

```
GAmodel <- rbga.bin(size = "iloscZmiennyc", popSize = "rozmiar populacji",
iters = "iloscIteracji", mutationChance="szansaMutacji", elitism=T,
evalFunc=fitnessFunc)
```

Efektywność zależna jest od poprawnego ustawienia paramaterów, co opisane jest w dalszej części dokumentu.

Wszystkie komendy użyte przy projekcie można znaleźć w pliku *komendy.r* lub *HISTORIA.Rhistory.*

4. Chromosomy oraz funkcje fitness

Chromosomy tworzone przez algorytm są to ciagi bitów o długości odpowiadającej ilości zmiennych w problemie. Jeżeli spełniają one klazule, uznawane są za dobre.
W przypadku funkcji fitness istnieja rózne sposoby na jej wybranie. Zaprezentowana przeze mnie wyżej, polega na przyznawaniu ‘kary’ za każdą niespełnioną klazule. Program sprawdza caly rząd i sprawdza czy dana klauzula zwróci false. Jezeli tak, chromosom otrzymuje punkt „karny”.
Druga funkcja fitness którą stworzyłem jest:

```
function(chromosome) {
    counter = 0
    result = FALSE
    for(i in 1:nrow(values)){
        row <- values[i,]
        row_1 <- abs(row[1])
        row_2 <- abs(row[2])
        row_3 <- abs(row[3])

        x1 = chromosome[row_1]
        if(row[1] < 0){ x1 = !x1 }
        if(x1 == 1) {
            counter = counter +1
        }
        else {
            x2 = chromosome[row_2]
            if(row[2] < 0){ x2 = !x2 }
            if(x2 == 1) {
                counter = counter +1
            }
            else {
                x3 = chromosome[row_3]
                if(row[3] < 0){ x3 = !x3 }
                if(x3 == 1) {
                    counter = counter +1
                }
            }
        }
    }
    return(-counter)
}
```

Polega ona na sprawdzaniu elemetnów w klauzuli, jeżeli program natrafi na chociaż jeden element prawdziwy od razu przyznaje punk nagrody za poprawną klauzulę stworzoną przez zmienne w chromosomie. Dzieje się tak, ponieważ alternatywa elementów jest prawdziwa w momencie kiedy przynajmniej jeden z nich jest prawdziwy.

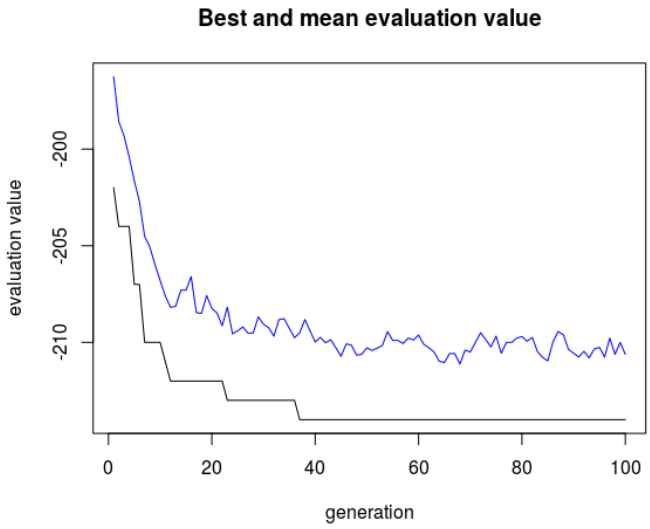
Rodzaj metody	Ilość zmiennych	Czas [s]
Nagradzanie	50	4,1
Karanie		4,5
Nagradzanie	100	8,1
Karanie		8,2

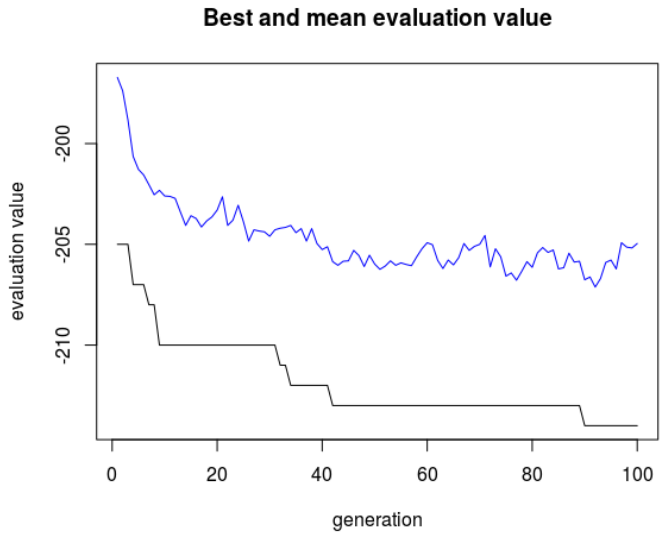
W tabeli przedstawiono porównanie pracy opisanych wyżej metod fitness. Badanie pokazuje, że nagradzanie bywa minimalnie szybsze. Spowodowane jest to prawdopodobnie zmniejszeniem ilosci analizowania zmiennych w wierszu. Kosztowne jednak może być używanie takiej ilości funkcji warunkowej *If Else*. Prawdopodobnie w przypadku kiedy prawdziwe były by zawsze ostatenie argumenty w klauzuli, sposob nagradzający mógłby być gorszy.
Na podstawie tej obserwacji, w moich dalszych badaniach opierać się będę na metodzie fitness nagradzającej za poprawną klazulę.

5. Manipulacja parametrami algorytmu bazując na jednej instancji problemu

Problem użyty do badań zawiera **100** zmiennych oraz **403** klauzul.
Po obserwacjach zachowań algorytmu, wpływ na jego końcowy efekt mają takie zmienne jak:

- **szansa mutacji** - możliwe jest utworzenie algorytmu calkowicie pozbawionego mutacji, jednak zawierać on musi bardzo dużą ilość populacji początkowej. Z drugiej strony, zbyt duża mutacja, nawet rzędu czasami 10%, sprawia że chromosomy mutują się zbyt często i osiągnięcie dobrego rezultatu w stosunkowo małej ilości iteracji staje się niemożliwe. Ustawienie mutacji 100%, generuje całkowicie przypadkowe chromosomy w okolicach tego samego wyniku.





Wykres po lewej mutacja 10%
wykres po prawej mutacja 20%

- **wielkość populacji początkowej** – zwiększanie znacząco wydłuża czas trwania algrotymu. Jest to spowodowane ilością chromosomów wymagających mutacji. Mimo to, dzięki większej ilości populacji początkowej, wynik bliżej satysfakcjonującego ukazuje się po mniejszej ilości iteracji. Nie jest to jednak tak dobry wynik w stosunku do czasu poświeconego na jego obliczenie.

- **elityzm** – nie wpływa na szybkość obliczeń w momencie kiedy określoną mamy ilość iteracji. Może wpływać na moment osiągnięcia spełnionego wyniku, co w przypadku 3SAT jest ciężkie w niedługim czasie. Z moich obserwacji i własnych preferencji elityzm jest używany w dalszych badaniach ze względu na zachowywanie najlepszych rozwiązań. Po przeanalizowaniu argumentów, nie zaobserwowałem znaczących przeciwskazań do używania elityzmu.

- **ilość iteracji** – im więcej dopuszczalnych iteracji tym dokładniejszy będzie nasz wynik. Również znacząco wpływa na czas wykonywania metody jednak końcowy efekt jest bardziej satysfakcjonujący niż w przypadku zwiększania tylko populacji początkowej.

Podsumowując, najlepszym pod wzgledem stosunku czasu do otrzymanego rezultatu jest zachowanie średniej ilosci początkowej populacji i zwiekszenie iteracji. Mutacja powinna być na niskim poziomie, około 1-4% dzieki temu wyniki nie będą się drastycznie zmieniać względem siebie, a teoretycznie korzystne wyniki nie będą się tak często modyfikować.
Przy moich danych, zachowanym elityzmie, mutacji na poziomie 1%, populacji 50 i 1000 iteracji, czas potrzebny do osiągnięcia **400** poprawnych klauzul wyniósł **85** sekund.

6. Manipulacja długością klauzul oraz liczbą zmiennych

Na wykresie przedstawiono czas w którym wykonywany został algorytm w zależnosci od ilości zmiennych. Jak widać, czas działania algorytmu wzrasta liniowo.
Głównym czynnikiem wpływającym na czas jest ilość początkowej populacji. Ze względu na zwiększającą się liczbę zmiennych, przyszła mutacja wymaga większej ilości działań. Ilość iteracji nie ulegała zmianie, a ostateczny wynik stawał się coraz mniej satysfakcjonujący.