# Sprawozdanie 2 (algorytmy genetyczne/ewolucyjne)

**Autorzy:** Dominika Skrzyńska, Daniel Witkowski  
**Grupa:** ININ5\_PR3.1  
**Data:** 15.12.2019  
**Przedmiot:** Sztuczna inteligencja

|  |  |
| --- | --- |
| **Cel zadania** | Rozwiązanie problemu plecakowego przy pomocy algorytmu genetycznego |
| **Źródło danych** | Zbiór danych to 28 jednostek bojowych, z których każda posiada atrybuty cząstkowe: atak, obrona, przyrost oraz koszt jednostkowy. Dane wprowadzono ręcznie. |
| **Struktura chromosomu** | Chromosom jest ciągiem binarnym o długości 28 bitów. Każdy bit odpowiada  włączeniu jednostki do armii (wartość 1) lub pozostawieniu (wartość 0) |
| **Które chromosomy są dobre?** | Chromosom jest tym lepszy, im większa jest wartość bojowa jednostek, na które wskazują jego bity (pod warunkiem, że wejdą w skład armii). |
| **Działanie funkcji Fitness** | Funkcja przyjmuje jako parametr chromosom chr. i oblicza łączny koszt wskazywanych przez niego jednostek. Jeżeli koszt zostanie przekroczony zwraca wartość 0 (najgorsza zmożliwych ocen). Jeżeli koszt nie zostanie przekroczony funkcja zwraca wartość zabranych jednostek pomnożoną przez -1. |
| **Wnioski i podsumowanie** | Warunki zadania:   * Łączny koszt armii poniżej 300. * Uzyskanie maksymalnej wartości bojowej.   Parametry algorytmu zostały dobrane pod kątem optymalizacji czasowej:   * liczebność populacji (popSize) = 50 * prawdopodobieństwo krzyżowania (pcrossover) = 0.25 * prawdopodobieństwo permutacji (pmutation) = 0.1 * maksymalna ilość iteracji (maxiter) = 82   Najlepsze uzyskane rozwiązanie dla danych ustawionych losowo:  [1] "Rozwiązanie: "  jednostka atak obrona przyrost koszt\_jedn wartosc\_bojowa koszt\_sumaryczny  5 Mnich 61 41 3 33 306 99  15 Szkielet 8 9 7 6 119 42  22 Goblin 11 15 12 5 312 60  26 Rok 59 50 2 20 218 40  28 Behemot 92 22 3 16 342 48  [1] "Koszt armii = 289"  [1] "Wartość bojowa jednostek = 1297"  Czas wykonania algorytmu: 0,9649873 s.[[1]](#footnote-2)  Wykres dla zastosowanego agorytmu wygląda następująco:    Zastosowanie sortowania rosnącego dla kosztów daje wynik zupełnie inny, przy zachowaniu podobnego czasu wykonania skryptu:  [1] "Rozwiązanie: "  jednostka atak obrona przyrost koszt\_jedn wartosc\_bojowa koszt\_sumaryczny  1 Pikinier 10 11 9 10 189 19  2 Łucznik 25 2 8 45 216 10  3 Gryf 28 31 6 30 354 2  4 Zbrojny 35 14 5 21 245 21  5 Mnich 61 41 3 33 306 17  6 Kawalerzysta 82 72 2 82 308 14  7 Anioł 102 62 1 91 164 16  9 Harpia 14 4 9 27 162 12  10 Złe oko 21 17 12 33 456 9  11 Meduza 30 28 5 14 290 23  12 Minotaur 54 52 4 65 424 3  13 Mantikora 70 60 2 80 260 18  14 Czerwony smok 172 12 3 101 552 6  16 Ożywieniec 19 17 11 25 396 24  17 Zjawa 23 21 5 64 220 8  18 Wampir 38 30 4 44 272 4  19 Lisz 49 45 13 78 1222 5  20 Czarny Rycerz 66 46 1 81 112 7  21 Kościany smok 82 35 3 120 351 1  22 Goblin 11 15 12 5 312 20  23 Wilczy jeździec 24 22 8 25 368 27  24 Ork 28 8 3 37 108 11  25 Ogr 55 47 6 44 612 22  [1] "Koszt armii = 299"  [1] "Wartość bojowa jednostek = 7899"  Time difference of 1.073792 secs    Zastosowanie sortowania malejącego kosztu daje wynik minimalnie gorszy:  [1] "Koszt armii = 299"  [1] "Wartość bojowa jednostek = 7677"  Czas wykonania jest podobny.  Pdsumowując, zastosowanie mniejszego prawdopodobieństwa mutacji oraz mniejszej populacji w algorymie wymaga mniejszej ilości iteracji do uzyskania maksymalnego wyniku, algorytm jest dokładniejszy. Dla danych losowych po osiągnięciu maksymalnego wyniku ilość iteracji przestaje mieć znaczenie.  Porównanie wyniku dla danych ustawionych losowo i danych posortowanych pozwala zauważyć, że ten pierwszy jest daleki od właściwego maksimum. |
| **Bibliografia** | 1. Grzegorz Dudek, *Wykład na temat ewolucyjnych metod optymalizacji,* <https://www.youtube.com/watch?v=5emurnF-5fc> 2. [https://www.rdocumentation.org](https://www.rdocumentation.org/) 3. <https://www.rdocumentation.org/packages/GA/versions/3.2> |

## Kod źródłowy rozwiązania wraz z dodatkowymi (szczegółowymi) komentarzami:

|  |
| --- |
| Zapis momentu rozpoczęcia wykonania skryptu (timestamp)  **start\_time <- Sys.time()**  Treść zadania:  **#Gracz musi zebrac jednostki do bitwy.**  **#Każda z jednostek ma swoją wartość bojową, na którą sklądają się atak i obrona.**  **#Jednostki każdego typu są dostępne w określonej ilości (przyrost).**  **#Gracz ma do wykorzystania 300 złotych monet.**  **#Określ optymalny skład armii z zachowaniem limitu kosztu.**  Następuje instalacja i wczytanie paczki "GA", pozwalającej na korzystanie z funkcji algorytmu genetycznego:  **install.packages("GA")**  **library(GA)**  Przy pomocy ***data.frame*** definiujemy zbiór danych w postaci macierzy 29 x 5, z czego pierwszy rząd zawiera etykiety kolumn, pozostałe 28 rzędów zawiera dane jednostek bojowych (atak, obrona, przyrost, czyli ilość dostępnych jednostek danego typu, oraz koszt jednostkowy):  **armia = data.frame(**  **jednostka = c("Pikinier", "Łucznik", "Gryf", "Zbrojny", "Mnich", "Kawalerzysta", "Anioł", "Troglodyta", "Harpia", "Złe oko", "Meduza", "Minotaur", "Mantikora", "Czerwony smok", "Szkielet", "Ożywieniec", "Zjawa", "Wampir", "Lisz", "Czarny Rycerz", "Kościany smok","Goblin", "Wilczy jeździec", "Ork", "Ogr", "Rok", "Cyklop", "Behemot"),**  **atak = c(10, 25, 28, 35, 61, 82, 102, 4, 14, 21, 30, 54, 70, 172, 8, 19, 23, 38, 49, 66, 82, 11, 24, 28, 55, 59, 93, 92),**  **obrona = c(11, 2, 31, 14, 41, 72, 62, 14, 4, 17, 28, 52, 60, 12, 9, 17, 21, 30, 45, 46, 35, 15, 22, 8, 47, 50, 33, 22),**  **przyrost = c(9, 8, 6, 5, 3, 2, 1, 10, 9, 12, 5, 4, 2, 3, 7, 11, 5, 4, 13, 1, 3, 12, 8, 3, 6, 2, 1, 3),**  **koszt\_jedn = c(10, 45, 30, 21, 33, 82, 91, 11, 27, 33, 14, 65, 80, 101, 6, 25, 64, 44, 78, 81, 120, 5, 25, 37, 44, 20, 78, 16)**  **)**  Definiujemy maksymalny koszt armii:  **armiaKosztMax = 300**  Następnie definiujemy dodatkowe kolumny w macierzy przechowujące wyliczenie łącznej wartości bojowej oddziau jednostek (***wartosc\_bojowa***)oraz jego sumaryczny koszt (***koszt\_sumaryczny*** *=* przyrost \* koszt\_jedn):  **armia$wartosc\_bojowa <- rowSums(armia[,2:3], na.rm=FALSE) \* armia[,4]**  **armia$koszt\_sumaryczny <- armia[,4]\*armia[,5]**  Opcjonalnie można zastosować sortowanie danych w porządku rosnącym:  #armia$wartosc\_bojowa <- order(rowSums(armia[,2:3], na.rm=FALSE) \* armia[,4])  #armia$koszt\_sumaryczny <- order(armia[,4]\*armia[,5])  Lub sortowanie danych w porządku malejącym:  #armia$wartosc\_bojowa <- order(-rowSums(armia[,2:3], na.rm=FALSE) \* armia[,4])  #armia$koszt\_sumaryczny <- order(-armia[,4]\*armia[,5])  Definiujemy funkcję przystosowania ***fitnessFunc***, która uwzględnia sumaryczną wartość i koszt jednorodnego oddziału jednostek. Jeżeli łączny koszt wybranego zestawu jednostek (***calkowitykosztChr***)jest większy niż koszt maksymalny funkcja zwraca ujemną całkowitą wartość bojową zestawu (***calkowitaWartoscChr***), w odwrotnym przypadku zwraca wartość bojową.  **fitnessFunc = function(chr) {**  **calkowitaWartoscChr = chr %\*% armia$wartosc\_bojowa**  **calkowitykosztChr = chr %\*% armia$koszt\_sumaryczny**  **print(calkowitaWartoscChr)**  **print(calkowitykosztChr)**  **if (calkowitykosztChr > armiaKosztMax) return(-calkowitaWartoscChr)**  **else return(calkowitaWartoscChr)**  **}**  Następuje uruchamienie algorytmu genetycznego dla zadanych parametrów, gdzie ***nBits*** to ilośc pól chromosomu, ***popSize*** – wielkość populacji, ***pcrossover*** - prawdopodobieństwo krzyżowania,  ***pmutation*** - prawdopodobieństwo permutacji, ***maxiter*** - maksymalna ilość iteracji:  **wyniki=ga( type="binary",**  **nBits=28,**  **fitness=fitnessFunc,**  **popSize=50,**  **pcrossover=0.25,**  **pmutation=0.1,**  **elitism=5,**  **maxiter=82,**  **seed=10 )**  Skrypt wypisuje podsumawanie działania algorytmu genetycznego oraz rysuje wykres:    **summary(wyniki)**  **plot(wyniki)**  Następnie następuje dekodowanie (prezentacja) pojedynczego rozwiązania, ze wskazaniem wybranych oddziałów jednostek, tam gdzie pole chromosomu jest równe 1, oraz ich łącznego kosztu i łącznej wartości bojowej.  **decode=function(chr){**  **print("Rozwiązanie: ")**  **print( armia[chr == 1, ] )**  **print( paste("Koszt armii =",chr %\*% armia$koszt\_sumaryczny) )**  **print( paste("Wartość bojowa jednostek =",chr %\*% armia$wartosc\_bojowa) )**  **}**  **decode(wyniki@solution[1,])**  Następuje zapis czasu zakończenia działania skryptu i wypisanie różnicy między momentem końca i startu, co pozwala na określenie łącznego czasu działania skryptu:  **end\_time <- Sys.time()**  **print(end\_time - start\_time)** |

1. Algorytm testowany w systemie Linux Ubuntu 18.04 [↑](#footnote-ref-2)