Podstawy Sztucznej Inteligencji projekt

Problem plecakowy - Do plecaka o pojemności C chcemy zapakować przedmioty o jak największej wartości. Każdy przedmiot definiowany jest przez wartość W i objętość O jaką zajmuje w plecaku. Zastosować algorytm genetyczny i zamodelować pewien sposób reprezentacji genotypu do rozwiązania tego problemu. WE: plik z listą przedmiotów opisanych wartością W i objętością O, pojemność plecaka C. WY: przedmioty mieszczące się w plecaku, które maksymalizują jego wartość.

Dodatkowo założyliśmy, że wartości W, O i C są liczbami całkowitymi dodatnimi oraz, że kolejność przedmiotów na wyjściu nie ma znaczenia

Podział odpowiedzialności

Michał Kucharski	**Stanisław Czobot**
• Szkielet programu • Ocena osobników • Krzyżowanie osobników • Mutowanie osobników • Selekcja metodą n najlepszych • Dokumentacja projektu	• Algorytm brutalny • Obsługa wejść/wyjść • Testy, debugowanie, pomiary • Usprawnienia kodu • Selekcja metodą ruletki • Dokumentacja kodu - JavaDoc

Opis programu

Program został stworzony w języku Java i podzielony na klasy, tak aby był czytelny oraz jak najlepiej spełniał zasady dobrego programowania. Rolę fenotypu pełni lista przedmiotów, natomiast genotypy to tablice wartości logicznych true/false (jeśli wartość na miejscu o indeksie i = true tzn. że i-ty przedmiot z dostarczonych znajduje się w danym genotypie, a jeśli false to nie znajduje się) . Wybraliśmy ten sposób kodowania ze względu na prostotę i intuicyjność jego odbioru.

Bardzo trudno było zdecydować się na realizację tylko jednej metody, czy to selekcji czy krzyżowania, w związku z tym dla tych operacji postanowiliśmy zrealizować dwie metody realizacji. Pozwala to na lepsze zbadanie problemu i zwiększa szansę na jak najlepsze działanie algorytmu. Ponadto w poleceniu było napisane, że można zawrzeć ewentualne ulepszenia, więc postanowiliśmy nie tylko je napisać, ale również zaimplementować i dopiero ulepszony algorytm poddać testom. Na potrzeby testów został zaimplementowany również algorytm brutalny, tworzący wszystkie możliwe kombinacje genów i wybierający najlepiej przystosowanego osobnika. Poniżej znajduję się opis najważniejszych składowych algorytmu genetycznego:

Generacja pierwszego pokolenia – wektor losowych wartości logicznej sprawiał problemy, bo zdarzały się przypadki, że dla wszystkich osobników funkcja przystosowania osiągała wartość minimalną (wszystkie fenotypy przekraczały dozwoloną objętość), dlatego zdecydowaliśmy się na wybieranie z wyznaczonych losowo osobników te, dla których funkcja celu jest większa od 0.

Krzyżowanie - Zaimplementowane zostały dwie metody: krzyżowanie jednopunktowe oraz wybór genu na danym miejscu od losowego z dwojga rodziców. Podczas testów wstępnych krzyżowanie jednopunktowe wypadło zdecydowanie gorzej, więc zdecydowaliśmy się nie używać go do dalszych testów.

Funkcja przystosowania - Wartość funkcji celu jest sumą wartości przedmiotów z danego fenotypu. Im większa tym lepsza, a jeśli wartość ta jest większa od objętości plecaka, to 0, czyli najgorsza z możliwych wartości.

Selekcja – Do selekcji zdecydowaliśmy się wybrać dwa algorytmy : koła ruletki oraz n najlepszych. Oba wykazywały ciekawe właściwości, więc oba zostały poddane testom. Początkowo selekcja, nie działała tak jakbyśmy sobie tego życzyli, gdyż zdarzało się, że kolejne pokolenia były gorzej przystosowane od poprzednich. W związku z tym postanowiliśmy ją usprawnić i zdecydowaliśmy się, że będziemy zapewniać miejsce w kolejnym pokoleniu najlepiej przystosowanemu osobnikowi z danego pokolenia oraz zablokujemy możliwość mutacji tego osobnika. Daję to pewność, że najlepszy osobnik z kolejnego pokolenia będzie co najmniej tak dobry jak najlepszy z poprzedniego pokolenia.

Warunek Stopu – Algorytm kończy się w przypadku gdy 50 pokoleń z rzędu najlepiej przystosowany osobnik będzie miał tą samą wartość funkcji przystosowania.

Testy

Testy zostały przeprowadzone z wykorzystaniem różnych rodzajów selekcji, różnych rozmiarów problemu oraz różnych wielkości populacji. Poniżej przedstawione są tabele i wykresy z przeprowadzonych eksperymentów. Każdy typ algorytmu dla każdego rozmiaru problemu i populacji został przetestowany 10 razy. Powyżej 25 przedmiotów algorytm brutalny wyrzucał błąd braku pamięci. Średnia ”dobroć” – oznacza stosunek średniego wyniku to najlepszego możliwego do uzyskania

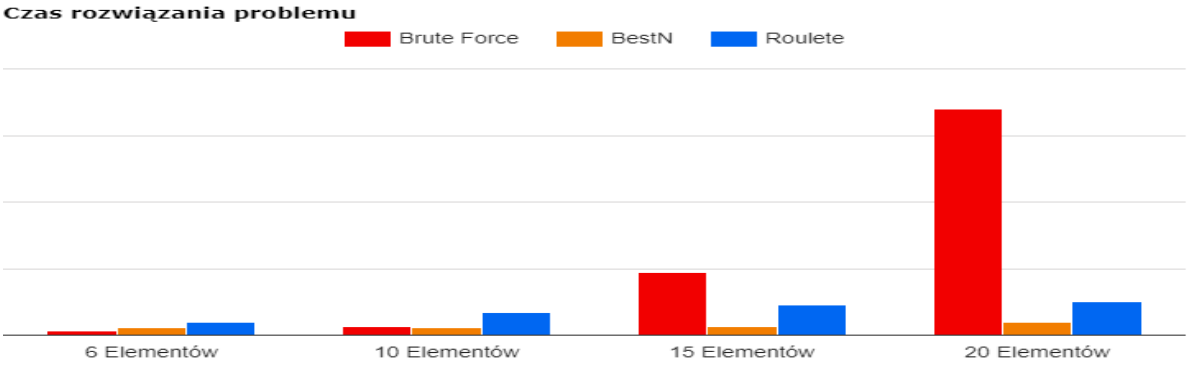
Typ algorytmu	Rozmiar populacji	Średni czas w milisekundach	Średni Wynik	najlepszych wyników	Średnia „dobroć”
6 przedmiotów do wyboru					
Brute	64	59	14	100%	100%
BestN	4	90	10.1	40%	72%
Roulette	4	187	14	100%	100%
BestN	7	110	12.8	70%	91%
Roulette	7	192	14	100%	100%
15 przedmiotów do wyboru					
Brute	2^15	950	25	100%	100%
BestN	16	137	22.5	10%	90%
Roulette	16	461	24.4	60%	98%
BestN	8	105	21.2	20%	85%
Roulette	8	290	24	40%	96%
25 przedmiotów do wyboru					
Brute	2^25	49030	47	100%	100%
BestN	40	336	44.1	10%	94%
Roulette	40	1128	44.9	0%	96%
BestN	100	482	46.1	40%	98%
Roulette	100	2834	44.3	0%	94%

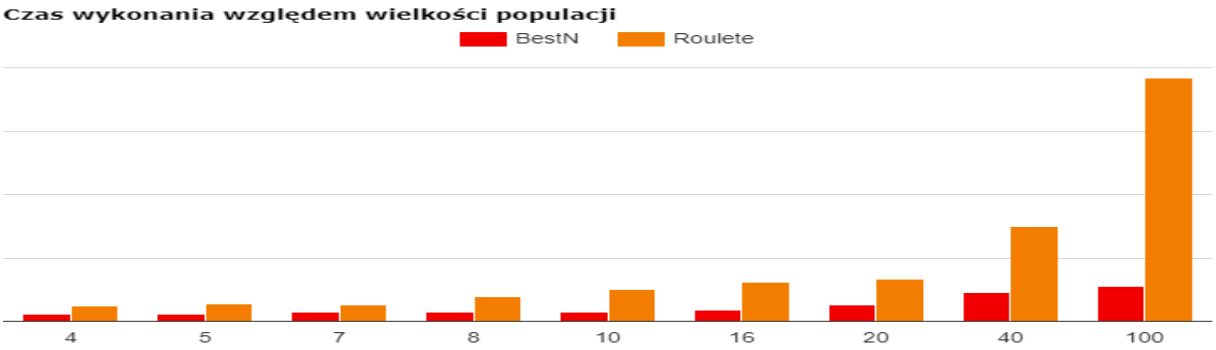
Analizując dane z tabeli można stwierdzić, że przy małych problemach zdecydowanie najlepiej sprawdza się algorytm brutalny. Ponadto selekcja metodą koła ruletki działa wolniej niż n najlepszych, lecz jeśli zależy nam bardziej na wyniku niż czasie, to lepiej wybrać właśnie ją, bo ma lepszy średni wynik i częściej znajduję najlepszy wynik.

Przy większych problemach wnioski z poprzedniego paragrafu potwierdzają się, jeśli chodzi o porównanie dwóch metod selekcji. Największa zmianą jest czas algorytmu brutalnego, gdyż algorytmy genetyczne biją go na głowę, przy bardzo zbliżonych wynikach dopasowania.

Przy dużych rozmiarach problemu różnica czasowa między algorytmem brutalnym a genetycznymi zwiększa się znacząco. Ciekawą obserwacją jest to, że w tym przypadku selekcja n najlepszych sprawdza się lepiej, szczególnie jeśli chodzi o liczbę maksymalnych wyników.

W tabelach nie zostały umieszczone liczby iteracji, gdyż nie różniły się one dla metod selekcji , a jedynie zwiększały się proporcjonalnie do rozmiarów problemu, co było oczywiste. Można przytoczyć dla ciekawskich liczby: dla małych problemów jest to około 50 iteracji a dla dużych nawet do 100





Wykresy w jeszcze przyjemniejszy dla oka sposób pokazują, że algorytmy genetyczne zdecydowanie szybciej rozwiązują problemy niż rozwiązania brutalne, ale dopiero gdy ich rozmiary będą odpowiednio duże. Można też zauważyć wyraźną różnicę w czasie wykonania algorytmów genetycznych w zależności od populacji. Metoda n najlepszych jest znacząco szybsza, a różnica między kołem ruletki zwiększa się wraz ze wzrostem liczby osobników w populacji. Podsumowując jeśli zależy nam tylko na czasie wybieramy selekcję n najlepszych. Dla małych i średnich problemów możemy rozważać jeszcze koło ruletki jeśli zależy nam nie tylko na czasie, ale również na wyniku bardzo zbliżonym do najlepszego. W przypadku problemów dużych warto wybrać selekcję n najlepszych bo wygrywa czasowo, a jest niemal identyczna, a nawet czasem lepsza jeśli chodzi o maksymalizację wyniku.

Uruchomienie programu

Program można uruchomić w konsoli. Wymagane jest posiadanie zainstalowanego programu Apache Maven. W tym celu należy użyć poniższych poleceń będąc w katalogu głównym (ten w którym jest plik pom.xml:

- mvn compile
- mvn exec:java -Dexec.mainClass="Main" -Dexec.args="packSize popSize selMode input.txt"

Argumenty to odpowiednio: Pojemność plecaka, rozmiar populacji, typ selekcji (R – koło ruletki, B – n najlepszych) oraz plik z danymi wejściowymi. Pliki przykładowe w katalogsrc/main/resources/Inputs