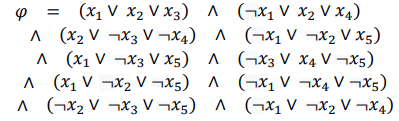
**Rozwiązywanie problemu 3SAT – Wiktoria Wolnik**

**Specyfikacja problemu**

W logice i informatyce problem spełnialności boolowskiej (czasami nazywany problemem spełnialności zdań i w skrócie SATISFIABILITY , SAT lub B-SAT ) jest problemem ustalenia, czy istnieje interpretacja , która spełnia daną formułę Boole'a . Innymi słowy, pyta, czy zmienne danej formuły Boole'a mogą być konsekwentnie zastępowane wartościami PRAWDA lub FAŁSZ w taki sposób, że formuła ma wartość PRAWDA. W takim przypadku formuła nazywa się „satisfiable”.

**Jak wyglądają omawiane tautologie?**

****

**Jak zamierzam znaleźć najlepsze rozwiązanie?**

Zamierzam wypróbować 3 różne metod poszukiwania takowych rozwiązań: metoda Brute force, Algorytm genetyczny z własną ewolucją (zastosowanie elitaryzmu przy wyborze następnych pokoleń) oraz roju cząstek.

**Trochę o kodzie:)**

**Jak wyglądają moje formuły?**

Każda z reguły OR składa się z 3 elementów reprezentowanych za pomocą słownika. X\_n oznacza liczbę (której przypisana jest wartość 0 albo 1), a ‘negative’ oznacza czy przed tą liczbą jest minus (jeśli jest wtedy rzeczywista wartość tej liczby jest odwrotna).

[[[{'x\_n': 9, 'negative': 0}, {'x\_n': 1, 'negative': 1}, {'x\_n': 7, 'negative': 0}], [{'x\_n': 2, 'negative': 1}, {'x\_n': 7, 'negative': 0}, {'x\_n': 9, 'negative': 0}]], [[{'x\_n': 8, 'negative': 1}, {'x\_n': 3, 'negative': 0}, {'x\_n': 5, 'negative': 1}], [{'x\_n': 4, 'negative': 1}, {'x\_n': 2, 'negative': 1}, {'x\_n': 0, 'negative': 1}]]]

**Funkcja fitness:**

Moja funkcja fitness przechodzi po wszystkich elementach i sprawdza czy chociaż jedna z trzech wewnętrznych wartości jest 1 (bo rozpatrujemy wewnętrzne nawiasy (x1 or x2 or x3)), jeśli jest to dodajemy do sumy +1, jeśli nie, przechodzimy do następnej formuły.

Wynik który zwraca funkcja fitness to ilość „prawdziwych” tautologii (x1 or x2 or x3). To znaczy, że maksymalna (najlepsza) wartość fitness wynosi tyle ile jest łącznie takich nawiasów w całej formule.

Nie stosuje systemu kar, ponieważ przy tym problemie ujemne punkty nie są istotne i nie prowadziłyby do lepszego rozwiązania. Jedyna „kara” jest taka, że za fałszywy nawias punktów nie dodajemy.

Moje **chromosomy** to listy o długości n (gdzie n wynosi maksymalną wartość x\_n) wypełnione 0 i 1. To znaczy, że jeśli na pozycji index=5 mamy 1 to x\_5=prawda.

**Kod funkcji fitness:**

Obraz zawierający tekst

Opis wygenerowany automatycznie

**Brute Force**

Pierwsza metoda którą omówię to „Brute Force”, funkcja ta „brutalnie” szuka rozwiązań. Funkcja nie korzysta w żadnym stopniu z rozwoju genetycznego. Generuje ona bowiem każdy możliwy chromosom. Po kolei przechodzi po wszystkich chromosomach szukając maksymalnej wartości funkcji fitness. Jeśli funkcja znajdzie taki wynik (będzie on równy limitowi funkcji fitness, czyli sumie nawiasów wewnętrznych) to algorytm kończy działanie i wyświetla wynik. Jeśli nie to idzie dalej, a po zakończeniu przedstawia wynik najbardziej zbliżony do maksimum.

Przykład produkowania „chromosomów” przez „Brute force” dla 4 zmiennych:

[0, 0, 0, 0]

[0, 0, 0, 1]

…

[1, 1, 0, 1]

[1, 1, 1, 0]

[1, 1, 1, 1]

**Kod funkcji Brute force:**

Obraz zawierający tekst

Opis wygenerowany automatycznie

**Algorytm genetyczny**

Druga metoda, którą zdecydowałam się przetestować to „algorytm genetyczny”.

**Jak to działa?**

1. Losowana jest pewna populacja początkowa.
2. Populacja poddawana jest ocenie (selekcja). Najlepiej przystosowane osobniki biorą udział w procesie „reprodukcji”.
3. Genotypy wybranych osobników poddawane są operatorom ewolucyjnym:
   1. Są ze sobą kojarzone poprzez złączanie genotypów rodziców (krzyżowanie),
   2. przeprowadzana jest „mutacja”, czyli wprowadzenie drobnych losowych zmian, na poziomie „probability”.
4. Rodzi się drugie (kolejne) pokolenie. Aby utrzymać stałą liczbę osobników w populacji niektóre są powielane, a inne usuwane. Jeżeli nie znaleziono dostatecznie dobrego rozwiązania, algorytm powraca do kroku drugiego. W przeciwnym wypadku wybieramy najlepszego osobnika z populacji - jego genotyp to uzyskany wynik.

**Moja implementacja**

Zdecydowałam się nie korzystać z wbudowanej funkcji pygad, tylko napisać własne założenia ewolucji. Stosuję metodę elitarnego wybierania osobników z populacji. Po przejściu kodu przez zadane pokolenie sortuje populacje według wyniku z funkcji fitness i wybieram tylko parę najlepszych osobników.

**Kod funkcji ewolucji:**

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

**Tworzenie własnych wyrażeń do problemu 3 SAT:**

Do x\_n przypisuje randomową wartość z zakresu od 0 do liczby którą wybrałam jako długość chromosomu, a do „negative” przypisuje randomowo wartość 0 (fałsz) lub 1 (prawda).

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

**Jak wygląda zapis takiego problemu?**

Przykładowa tautologia do sprawdzenia:

[(x\_88 or x\_77 or x\_7) and (x\_12 or x\_5 or x\_68) and (x\_66 or x\_63 or x\_49) and (x\_46 or x\_99 or x\_50) and (x\_17 or x\_86 or x\_11) and (x\_28 or x\_2 or x\_52) and (x\_57 or x\_50 or x\_92) and (x\_0 or x\_40 or x\_10) and (x\_35 or x\_67 or x\_96) and (x\_62 or x\_87 or x\_60) and (x\_64 or x\_43 or x\_39)]

**Rój cząstek – PSO**

Ostatnią metodą którą przygotowałam do przeprowadzenia eksperymentów jest rój cząstek (Particle Swarm Optimization).

**Jak to działa?**

Optymalizacja roju cząstek jest metodą obliczeniową, która optymalizuje problem poprzez iteracyjne próby ulepszenia kandydującego rozwiązania w odniesieniu do danej miary jakości. Na ruch każdej cząstki wpływa jej najlepiej znana lokalna pozycja, ale jest ona również kierowana w kierunku najbardziej znanych pozycji w przestrzeni poszukiwań, które są aktualizowane w miarę znajdowania lepszych pozycji przez inne cząstki. Oczekuje się, że przesunie to rój w kierunku najlepszych rozwiązań.

\*Cały kod roju cząstek zawarty jest w pliku project1/pso.py

**Eksperymenty:**

**Eksperyment 1 – proste formuły**

W tym eksperymencie bierzemy pod uwagę tautologie wykorzystujące 10 różnych zmiennych i 30 klauzulach (klauzula = nawias wewnętrzny (x1 or x2 or x3).

|  |  |  |
| --- | --- | --- |
|  | **Brute force** | **Algorytm genetyczny** |
| **Średni czas** | 0.0021660327911376953 | 0.002000093460083008 |
| **Oczekiwana najlepsza wartość fitness** | 30 | 30 |
| **Najlepsza wartość fitness** | 30 | 30 |

Jak widzimy dla stosunkowo „małych” formuł wyniki obu metod są porównywalne. Brute force w praktycznie identycznym czasie jak algorytm genetyczny znajduje najlepsze rozwiązanie.

**Eksperyment 2 – skomplikowane formuły**

Dla porównania przeprowadziłam eksperyment dla tych samych „metod rozwiązujących” przy zdecydowanie bardziej złożonych tautologiach. Są one złożone z 30 różnych zmiennych i zawierają 90 klauzul:

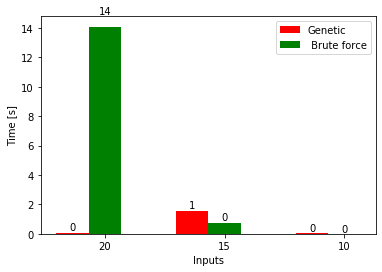
|  |  |  |
| --- | --- | --- |
|  | **Brute force** | **Algorytm genetyczny** |
| **Średni czas** | 645.4906756877899 | 0.6595320701599121 |
| **Oczekiwana najlepsza wartość fitness** | 90 | 90 |
| **Najlepsza wartość fitness** | 90 | 90 |

Jak widzimy wraz ze zwiększeniem skomplikowania problemu pojawiły się nam spore różnice w szybkości znajdowania rozwiązania. W tym przypadku algorytm genetyczny znalazł wynik 1000 razy szybciej niż algorytm Brute force.

**Podsumowanie eksperymentu 1 i 2:**

|  |  |
| --- | --- |
| **Brute force** | **Algorytm genetyczny** |
| + Dobrze radzi sobie przy małej złożoności problemu  + Algorytm przechodzi po wszystkich możliwych konfiguracjach, więc jeśli istnieje rozwiązanie to na pewno je znajdzie  - Przy złożonych zagadnieniach proces szukania jest zazwyczaj bardzo długi (zdecydowanie dłuższy niż innych algorytmów) | + Dobrze radzi sobie przy małej złożoności problemu  - Istnieje niewielka szansa, że mimo iż istnieje rozwiązanie to algorytm ten go nie znajdzie  + Czas poszukiwania rozwiązania przy nawet bardzo złożonych problemach jest niewielki |

Dla lepszego zobrazowania stworzyłam dodatkowo wykres pokazujący ile czasu potrzebują oba algorytmy aby znaleźć rozwiązanie dla problemu o 10, 15 i 20 zmiennych:



**Eksperyment 4 – co na to rój cząstek?**

Przeprowadziłam te same formuły przez algorytm PSO aby sprawdzić jak otrzymanego wyniki będą wyglądały w odniesieniu do algorytmu genetycznego i algorytmu brute force. W tabeli poniżej przedstawiam uzyskane dane:

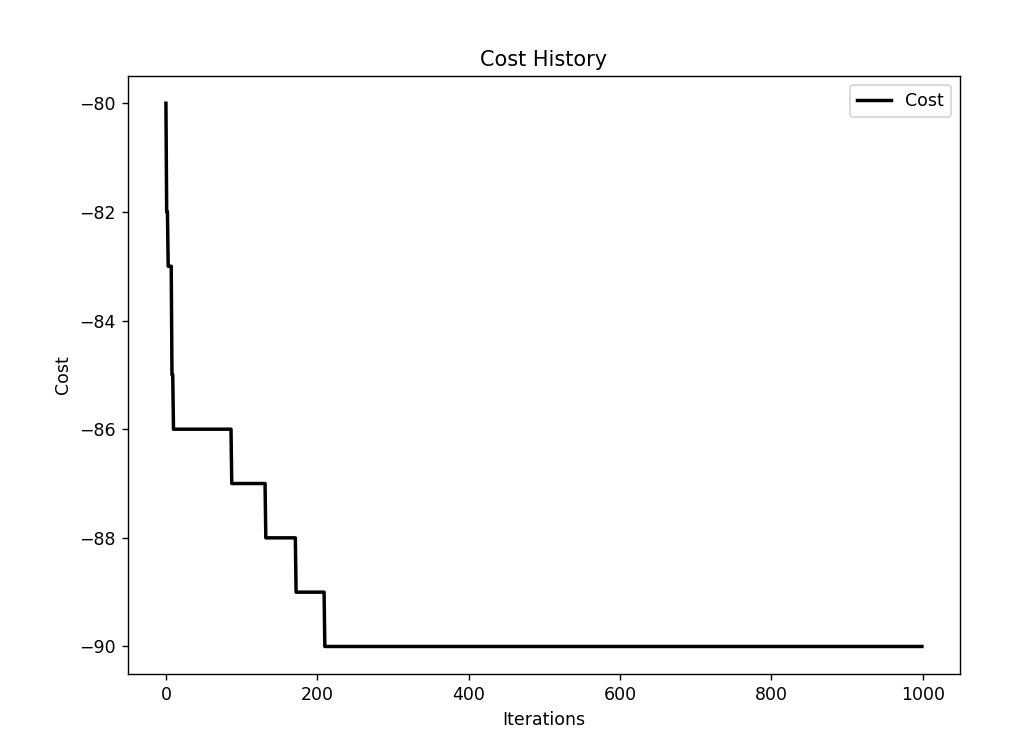
|  |  |  |
| --- | --- | --- |
|  | **Prosta formuła - 30 klauzul** | **Złożona formuła – 90 klauzul** |
| **Czas** | 0.23960399627685547 | 1.199157247543335 |
| **Oczekiwany wynik** | 30 | 90 |
| **Uzyskany wynik** | 30 | 90 |

Jak widzimy algorytm PSO działa zdecydowanie krócej niż omawiany wcześniej brute force, jego wyniki czasowe są zbliżone do algorytmu genetycznego.

A więc który z nich jest lepszy? PSO czy algorytm genetyczny?

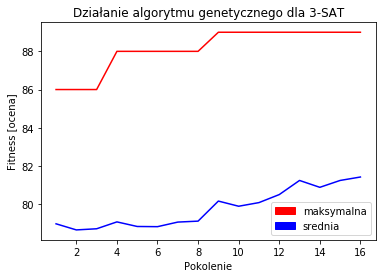
Uważam, że mimo iż wyniki algorytmu genetycznego są trochę lepsze to rój cząstek lepiej sprawdził się w przypadku rozwiązywania problemu 3SAT. Ponieważ, odsetek wyników błędnych gdzie program nie znajdował najlepszego rozwiązania był większy dla algorytmu genetycznego.

\*Dodatkowo zobrazowałam na wykresie jak przebiega proces znajdowania maksymalnego wyniku w złożonym problemie 90 klauzul:

****

**Eksperyment 5 – zmiana średniego wyniku wyznaczanego przez algorytm genetyczny**

W tym eksperymencie chciałam pokazać jak zmienia się średnia w algorytmie genetycznym. Dzięki zastosowaniu elitaryzmu podczas selekcji wybieranych rodziców możemy zauważyć, że średni wynik funkcji fitness stopniowo rośnie [niebieska linia]. Na wykresie zamieściłam też poglądowo jak zmienia się aktualna maksymalna otrzymana wartość [czerwona linia].

****

**Podsumowanie:**

Wszystkie zaprezentowane algorytmy były w stanie odnaleźć oczekiwane wyniki, głównym wyznacznikiem „lepszości” był czas kompilacji programu.

Uważam, że algorytm brute force świetnie sprawdza się dla mniejszych, prostszych problemów, ponieważ gwarantuje on wtedy znalezienie najlepszego rozwiązania w stosunkowo krótkim czasie.

Jeśli chodzi o rozwiązywanie trudniejszych problemów nie mogę określić, który z dwóch pozostałych algorytmów jest efektywniejszy. U mnie sprawdził się lepiej rój cząstek, ale dla innego zagadnienia może to być algorytm genetyczny. Obie te metody działają niezwykle szybko i z bardzo dużą skutecznością i dosłownie detale decydują, który sposób jest lepszy w danym przypadku.