Teoria algorytmów i obliczeń – Projekt – Etap 3

Błażej Bobko, Jakub Gocławski, Patryk Kujawski, Radosław Kutkowski

Wydział Matematyki i Nauk Informacyjnych, Politechnika Warszawska

1 Opis zadania

Zadanie polega na znalezieniu deterministycznego automatu skończonego, będącego rekonstrukcją struktury automatu na podstawie relacji indukowanej przez język. Na wejściu otrzymujemy automat, który pozwala sprawdzić czy: $\forall_{x,y\in\Sigma^*}\ xR_Ly$ i na podstawie odpowiedzi otrzymanych przez powyższą funkcję tworzymy automat wynikowy. Automat ten próbujemy znaleźć za pomocą kolejnych optymalizacji za pomocą algorytmu PSO.

2 Dokumentacja algorytmu

PSO (*Particle Swarm Optimization*) jest metodą obliczeniową polegającą na iteracyjnych próbach ulepszenia rozwiązania problemu optymalizacji poprzez przeszukiwanie przestrzeni rozwiązań w wielu różnych punktach jednocześnie. Potencjalne rozwiązania, nazwane dalej cząsteczkami, są porównywane za pomocą pewnej funkcji dopasowania a następnie "przesuwają się" w przestrzeni rozwiązań. Nowa pozycja jest dobierana na podstawie kilku czynników m.in. najlepszego znanego globalnie rozwiązania oraz najlepszego rozwiązania unikalnego dla danej cząstki.

2.1 Dane wejściowe PSO

- N – ilość cząsteczek
- Dim – wymiar przestrzeni przeszukiwania
- $LowerBound$ – wektor długości Dim zawierający w i-tej komórce największą wartość współrzędnej możliwą w i-tym wymiarze przestrzeni rozwiązań
- $UpperBound$ – wektor długości Dim zawierający w i-tej komórce największą wartość współrzędnej możliwą w i-tym wymiarze w przestrzeni rozwiązań
- $MaxEpochs$ – Liczba iteracji po których zwracamy najlepsze znalezione rozwiązanie
- $VelocityWeight$ – Waga jaką cząsteczka przywiązuje dla swojej aktualnej prędkości
- $CognitiveWeight$ – Waga jaką cząsteczka przywiązuje do najlepszego rozwiązania jakie napotkała
- $SocialWeight$ – Waga jaką cząsteczka przywiązuje do najlepszego rozwiązania jakie znaleziono w toku działania PSO
- $Fitness$ – Funkcja $R^{Dim} \to R$, zwracająca dla danego rozwiązania pewną wielkość opisującą to jak bardzo bliskie optymalnemu jest to rozwiązanie. Wartość tej funkcji chcemy minimalizować/maksymalizować.

Pseudokod algorytmu PSO

Sposób działania PSO przedstawiony jest za pomocą Algorytmu 1 na stronie 2.

2.2 Zbiór treningowy i testowy

Utworzenie zbiorów treningowego i testowego nastąpi poprzez wygenerowanie odpowiednio dużych zbiorów słów. Zbiór treningowy zawierać będzie wszystkie pary słów krótkich (długości nie większej niż parametr $c = 4$) nad danym alfabetem i losowe pary słów długich. Słowa długie zostaną wygenerowane w sposób losowy oraz poprzez (wielokrotną) konkatenację słów krótkich, a następnie w sposób losowy dobrane w pary. Zbiór testowy będzie zawierał wszystkie wygenerowane pary słów długich, które nie weszły w skład zbioru testowego.

Algorytm 1 PSO

```
1: procedure PSO
2:     BestSocialValue ← ∞
3:     stwórz N cząsteczek
4:     for i := 0 to N do
5:         stwórz wektor Location, w taki sposób, że
6:         for j := 0 to Dim do
7:             Location_i(j) ← Random(LowerBound(j), UpperBound(j))
8:         end for
9:         BestCognitive_i ← Location_i
10:        BestCognitiveValue_i ← Fitness(Location_i)
11:        if BestCognitiveValue_i < BestSocialValue then
12:            BestSocialValue ← BestCognitiveValue_i
13:            BestSocial ← BestCognitive_i
14:        end if
15:        ustal wektor Velocity_i w losowy sposób, tak, że
16:        for j := 0 to Dim do
17:            size ← |UpperBound(j) − LowerBound(j)|
18:            Velocity_i(j) ← Random(−size, size)
19:        end for
20:    end for
21:    for k := 0 to MaxEpochs do
22:        for i := 0 to N do
23:            for j := 0 to Dim do
24:                RandomSocial = Random(0, 1)
25:                RandomCognitive = Random(0, 1)
26:                Velocity_i(j) ← VelocityWeight ∗ Velocity_i(j) + SocialWeight ∗ RandomSocial ∗
   (BestSocial(j) − Location_i(j)) + CognitiveWeight ∗ RandomCognitive ∗ (BestCognitive_i(j) −
   Location_i(j))
27:                Location_i(j) ← Location_i(j) + Velocity_i(j)
28:            end for
29:            fitness ← oblicz wartoć Fitness(Location_i)
30:            if fitness < BestCognitiveValue_i then
31:                BestCognitive_i ← Location_i
32:                BestCognitiveValue_i ← fitness
33:                if BestCognitiveValue_i < BestSocialValue then
34:                    BestSocial ← BestCognitive_i
35:                    BestSocialValue ← BestCognitiveValue_i
36:                end if
37:            end if
38:        end for
39:    end for
40:    return BestSocial
41: end procedure
```

2.3 Dane wejściowe algorytmu

- A – automat wejściowy, podlegający rekonstrukcji
- M – liczba stanów automatu
- Al – wielkość alfabetu
- $MaxEpochs$ – liczba iteracji od ostatniego zaktualizowania najlepszego kandydata po których kończymy działanie algorytmu
- $MaxState$ – maksymalna liczba stanów jaką chcemy rozpatrzyć
- $LastBestCount$ – liczba cząsteczek stworzonych na podstawie najlepszego rozwiązania z poprzedniej iteracji (dla $LastBestCount = 0$ rozwiązania poprzedniej iteracji nie będą brane pod uwagę w inicjalizowaniu PSO w iteracji następnej)
- $DeathProbablility$ – szansa na zniszczenie cząsteczki i zastąpienie jej inną, losową (dla $DeathProbability = 0$ funkcjonalność umierania cząsteczek jest wyłączona)
- P – zbiór parametrów wywołania PSO

2.4 Działanie algorytmu

Dla każdej pary słów w zbiorze par treningowych wywoływane jest działanie automatu A. Wyniki zapisywane są w słowniku, gdzie kluczem jest para sprawdzanych słów, a wartością wyniki obliczeń automatu A. Słownik ten posłuży do sprawdzania współczynników błędu automatów konstruowanych w toku działania algorytmu PSO.

Analogicznie wygenerowany zostanie zbiór testowy, służący do mierzenia skuteczności automatu, po zakończeniu obliczeń.

Wywołujemy algorytm PSO dla każdego całkowitego i z przedziału $[2, MaxState]$ oraz ilości cząstek równej $P.N + LastBestCount$. Zmienna i określa liczbę stanów jaką ma mieć automat wynikowy. Cząsteczka zawiera automat, który zapisany jest w postaci macierzy $M * Al$ liczb zmiennopozycyjnych (z zakresu $[0; MaxState - 1 + \epsilon]$). Funkcja $Fitness$ ma do niego dostęp i ma możliwość uruchomienia na nim obliczeń. W trakcie przeprowadzania obliczeń, wartości zmiennopozycyjne są zaokrąglane w dół do liczby całkowitej, na potrzeby przypisania ich do konkretnego stanu.

Następnie dla każdej pary słów w zbiorze treningowych par sprawdzane jest czy oba słowa kończą działanie automatu w tym samym stanie. Na podstawie wyników przetwarzania wstępnego sprawdzamy, czy wynik ten jest błędem. Zwracana jest liczba napotkanych błędów.

W każdej iteracji prócz pierwszej do zbioru losowych cząstek dodawane są cząstki konstruowane na podstawie wyniku poprzedniej iteracji. Są to macierze z dodatkowym wierszem odpowiadającym nieosiągalnemu przez automat stanowi, wiersz ten wypełniany jest losowo, tak samo początkowy wektor prędkości cząsteczki również jest losowy.

W każdym kroku algorytmu PSO każda cząsteczka ma szansę „zginąć", w takim wypadku nadawany jej jest losową prędkość i losowa pozycja w przestrzeni rozwiązań.Prawdopodobieństwo to opisane jest zmienną $DeathProbability$. Rozwiązanie to ma na celu minimalizację wpływu minimów lokalnych na proces przeszukiwania przestrzeni.

Zmianie uległy również warunki zakończenia działania instancji PSO. PSO zwraca najlepszą wartość nie po danej ilości kroków, a po $MaxEpochs$ kroków od ostatniego zaktualizowania najlepszego kandydata na rozwiązanie. Nie spowoduje to nieskończonych obliczeń przy asymptotycznej zbieżności kolejnych kandydatów do optymalnego rozwiązania ze względu na to, że funkcja $Fitness$ przyjmuje wartości naturalne.

3 Dokumentacja techniczna rozwiązania

3.1 Szczegóły implementacyjne

Rozwiązanie zostało zaimplementowane w języku *C#* z wykorzystaniem technologii *WPF* do stworzenia środowiska graficznego (*GUI*). Nie były wykorzystywane żadne dodatkowe, zewnętrzne biblioteki.

Projekt został podzielony na 5 modułów opisanych poniżej.

UserInterface Moduł zawiera implementację *GUI* w technologii *WPF* oraz klasy pomocnicze wspomagające interakcję z użytkownikiem, m.in. w celu wczytania automatu z pliku lub wprowadzenia parametrów obliczeń.

TestGenerator Moduł zawiera klasę *TestSets*, służącą do przechowywania danych zbiorów: treningowego oraz testowego. Klasa ta potrafi także generować te zbiory, a także zapisywać je do pliku i wczytywać z pliku. Na podstawie danych zawartych w tej klasie można odpowiedzieć na pytanie, czy dwa słowa są w relacji.

PSO Moduł zawiera klasę *MachinePSO* zawierającą główną pętlę algorytmu *PSO* oraz klasę *Particle* będącą reprezentacją pojedynczej cząsteczki roju wykorzystywanego przez *MachinePSO*.

LanguageProcessor Moduł zawiera klasę *Machine* będącą reprezentacją odtwarzanego automatu, klasę *Alphabet* zawierającą dostępne litery alfabetu wraz z funkcjami ułatwiającymi konwersję ich formatu oraz klasę pomocniczą *Extensions*.

LanguageProcessorTests Moduł zawiera testy jednostkowe najważniejszych modułów rozwiązania.

3.2 Interfejs użytkownika

Po uruchomieniu programu możliwa jest modyfikacja głównych parametrów obliczeń. Oprócz podstawowych parametrów dokładniej opisanych w dokumencie z I etapu obecne są także dwa dodatkowe:

- Liczba cząsteczek – parametr PSO, liczba cząsteczek w roju
- Waga prędkości – parametr PSO, wpływ poprzedniej prędkości podczas ustalania kolejnej
- Waga lokalna – parametr PSO, wplyw najlepszego rozwiązania danej cząsteczki podczas ustalania nowej prędkości
- Waga globalna – parametr PSO, wplyw najlepszego rozwiązania z całego roju podczas ustalania nowej prędkości
- Szansa śmierci cząsteczki – cząsteczka z podanym prawdopodobieństwem może umrzeć w czasie każdego wykonywanego kroku i zostać zastąpioną nową, losową cząsteczką
- Cząsteczki przekazane do następnej iteracji – liczba cząsteczek z n-tej iteracji, które zostaną przekazane do $n + 1$-szej iteracji, gdzie n, to liczba stanów poszukiwanego automatu
- Maksymalna liczba stanów – maksymalna liczba stanów rekonstruowanego automatu

Parametry należy zatwierdzić przyciskiem *Zatwierdź parametry*. Następnie należy kliknąć przycisk *Wczytaj automat*. Pokaże się okno wyboru pliku, w którym należy wskazać plik w zdefiniowanym w zadaniu formacie. Po wczytaniu pliku należy kliknąć przycisk *Generuj zbiory*.

Pojawi się nowe okno umożliwiające albo wczytanie wygenerowanych wcześniej testów z pliku albo wygenerowanie nowych zbiorów testowych i treningowych.

Podczas generowania zbiorów testowych można ustalić parametry:

- Maksymalna długość „krótkiego” słowa – zostaną wygenerowane wszystkie pary słów o długości nie większej od podanego parametru

– Liczba długich słów (zbiór treningowy) – liczba par losowych słów dłuższych niż parametr określony powyżej
– Rozmiar zbioru testowego – liczba par dłuższych słów, nie biorących udziału w procesie optymalizacji w trakcie trwania PSO

Po wygenerowaniu zbiorów lub wczytaniu ich z pliku, należy nacisnąć przycisk *Akceptuj*. Po powrocie do głównego okna programu, aktywny staje się przycisk *Rozpocznij PSO*; jego kliknięcie powoduje pojawienie się nowego okna, w którym można rozpocząć obliczenia. Aby to zrobić, należy kliknąć przycisk *Start*. W trakcie trwania obliczeń można śledzić ich postęp. Po zakończeniu działania algorytmu pojawi się okno z komunikatem. Następnie można zapisać logi z obliczeń za pomocą przycisku *Stwórz log*. Można także podejrzeć różnice pomiędzy znalezionym, a poszukiwanym automatem, za pomocą przycisku *Porównaj automaty*.

3.3 Dodatkowe funkcjonalności

Zapisywanie i wczytywanie zbiorów testowych Raz wygenerowane pozwalają znacznie przyspieszyć porównywanie wydajności programu dla różnych parametrów oraz zapewniają, że próby odtwarzania będą dotyczyć dokładnie tego samego automatu.

Równoległe wykonywanie obliczeń Główna pętla programu została zaimplementowana z wykorzystaniem wątków za pomocą dostępnej w $C\#$ klasy $Task$.

Zapisywanie logów Po wykonaniu obliczeń możliwe jest zapisanie do pliku pełnego logu zawierającego wyniki dla każdej testowanej pary słów.

4 Raport z testów

4.1 Parametry obliczeń

- Liczba liter w alfabecie: 5
- Liczba cząsteczek: 20 (liczba wystarczająca, aby otrzymać interesujące wyniki, ale nie nazbyt duża, by nie spowolnić obliczeń)
- Waga prędkości (inertia weight): 0,729 (jedna z wartości polecanej dla PSO)
- Waga lokalna: (cognitive weight): 1,49445 (jedna z wartości polecanej dla PSO)
- Waga globalna: (social weight): 1,49445 (jedna z wartości polecanej dla PSO)
- Szansa śmierci cząsteczki: 0,01 (wybrana przez nas wartość, większa zaburzy algorytm, a mniejsza spowoduje znikome znaczenie tego usprawnienia)
- Cząsteczki przekazane do następnej iteracji: 4 (wybrana przez nas wartość)
- Maksymalna liczba stanów: (różne wartości w części A oraz B)

4.2 Testy dla stałej c=4

Obliczenia były wykonywane dla stałej $c = 5$. A zatem:

- Liczba wszystkich słów krótkich: 780
- Liczba wszystkich permutacji słów krótkich: 303 810
- A zatem rozmiar zbioru treningowego: 607 620
- Rozmiar zbioru testowego: 607 620

4.3 Testy dla stałej c=5

Nie udało się wykonać obliczeń dla stałej $c = 5$, gdyż taka wartość ma następujące konsekwencje:

- Liczba wszystkich słów krótkich: 3 905
- Liczba wszystkich permutacji słów krótkich: 7 622 560
- A zatem rozmiar zbioru treningowego: 15 245 120
- Rozmiar zbioru testowego: 15 245 120

Powoduje to problemy z pamięcią oraz wydajnością. Sam proces generowania zbiorów o takiej liczności wymaga w szczytowym momencie 10,5 GB pamięci RAM. Wymaga to bardzo mocnego komputera oraz wykonywania w środowisku 64-bitowym. Niestety nawet po wygenerowaniu tak ogromnych zbiorów obliczenia trwają bardzo długo. Próbowaliśmy uruchomić obliczenia dla Automatu nr 2 z podpunktu A, klasa 5-stanowa. Po 25 minutach obliczeń (z wykorzystaniem 11 GB pamięci RAM), przetwarzany był ciągle automat o 2 stanach, z błędem 45%.

Przerwaliśmy obliczenia, nie widząc perspektyw na ich ukończenie, a woleliśmy skupić się na dokładniejszych testach dla stałej $c = 4$.

4.4 A. Rekonstrukcja automatów

4.5 B. Aproksymacja automatów

Tablica 1. Aproksymacja za pomocą automatu o 6 stanach

Błąd „krótkich" słów	Błąd „długich słów"	Błąd łączny	Błąd zbioru testowego	Czas
21,80	21,91	21,85	21,94	15:23
20,78	20,54	20,66	20,73	14:34
20,65	20,61	20,63	20,71	14:51
20,93	20,91	20,92	21,00	17:08
21,23	21,17	21,20	21,12	14:56

Tablica 2. Aproksymacja za pomocą automatu o 8 stanach

Błąd „krótkich" słów	Błąd „długich słów"	Błąd łączny	Błąd zbioru testowego	Czas
18,34	18,23	18,29	18,26	13:36
18,18	18,44	18,31	18,42	20:21
17,62	17,48	17,56	17,46	23:11
18,73	18,93	18,83	18,88	21:15
17,91	18,31	18,11	18,26	19:09

Aproksymacja automatów 20 stanowych