3SAT - algorytm genetyczny vs DPLL

Raport projektu z przedmiotu Inteligencja Obliczeniowa

Krzysztof Łozowski
255157

24 stycznia 2018

1 Wprowadzenie

Celem projektu jest zaprezentowanie i porównanie czasów działania algorytmu genetycznego oraz algorytmu DPLL dla zagadnienia 3SAT.
Przed rozpoczęciem analizy problemu podane zostaną podstawowe definicje dotyczące wspomnianego zagadnienia.

Definicja 1.1 (KPN)
Formuła ϕ jest w koniunkcyjnej postaci normalnej jeśli jest ona koninkcją klauzul, z których każda jest alternatywą literałów, tzn. jest ona postaci

$$(p_{11} \vee \ldots \vee p_{1k_1}) \ \wedge \ (p_{21} \vee \ldots \vee p_{2k_2}) \ \wedge \ldots \wedge \ (p_{n1} \vee \ldots \vee p_{nk_n})$$

gdzie każde p_{ij} jest literałem.

Definicja 1.2 (k-SAT)
Formuła logiczna postaci KPN, w której każda klauzula składa się z nie więcej niż k literałów.

Zagadnienia 1-SAT oraz 2-SAT można rozwiązać w deterministycznym czasie wielomianowym P.
Rozważane przez nas zagadnienie 3-SAT jest już z kolei NP-zupełne, czyli takie, że każdy problem z klasy NP jest do niego redukowalny przy pomocy redukcji w czasie wielomianowym.
W projekcie rozważane są jedynie formuły rozwiązywalne, ponieważ jego głównym założeniem jest porównanie czasów działania algorytmów a nie problem rozwiązywalności formuł.

Repozytorium projektu dostępne jest pod adresem
`https://github.com/lozovsky/GAvsDPLL`.

2 Algorytm Genetyczny

W projekcie użyty został algorytm genetyczny, którego implementacja w języku **R** znajduje się w paczce **genalg**.

```
install.packages("genalg")
```

Na początku istotnym problemem był dobór odpowiednich parametrów dla algorytmu.

2.1 Dane i ich obróbka

W celu doboru odpowiednich parametrów zdecydowałem się na porównanie spełnialności skryptu dla wielu formuł.
Wybraną paczką danych została paczka o nazwie **CBS_k3_n100_m403_b10** dostępna pod adresem `http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`.
Pliki te znajdują się w folderze B403.

2.1.1 Wstępna obróbka danych

Następną czynnością do wykonania była obróbka tych danych. Proces ten został podzielony na następujące etapy:

1. Usunięcie zbędnych wierszy z plików
2. Usunięcie liczby 0 z końca każdego wiersza
3. Podział danych na kolumny z indeksami oraz wartościami logicznymi
4. Zamiana minusów na poprawione wartości logiczne

Dwa pierwsze podpunkty związane były stricte z operacjami na plikach, a ponieważ było ich aż 1000, to ręczna obróbka każdego z nich zajęłaby zbyt wiele czasu.
Można było jednak zauważyć, że w każdym z tych plików do wykonania były dokładnie te same operacje, więc przy użyciu komend w edytorze VIM:

```
:set hidden
:args *
:argdo %s/  0//g
:argdo 1,4d
:wa
```

Pliki te miały zagwarantowaną rozwiązywalność. W celu wygenerowania mniejszych plików (wprawdzie bez gwarancji rozwiązywalności, lecz zapewne część będzie rozwiązywalna), zostały stworzone odpowiednie foldery

```
$ echo B10/ B20/ B50/ B100/ B200/ B300/ | xargs -n 1 cp -rf B403/
```

a następnie przeprowadzono redukcję do odpowiedniej liczby literałów

```
:set hidden
:args B10/*
:argdo 11,403d
:wa
```

Operacje dla pozostałych folderów były niemal identyczne, nie ma sensu powielać poleceń.

2.1.2 Dalsza obróbka danych w skrypcie

Aby dane nadawały się do użytku dla funkcji fitness, należało zrobić jeszcze dwie rzeczy:

1. Odseparować indeks zmiennej od jej wartości logicznej
2. Pozbyć się minusa przy numerze indeksu, jednocześnie zmieniając wartość logiczną tej zmiennej na przeciwną

Za pierwszą z nich odpowiada funkcja *separateIndexFromValue*:

```
separateIndexFromValue <- function(data_frame_to_clear){
  number_of_rows = length(data_frame_to_clear[,1])
  DT <- data.frame(
                   V1_index = data_frame_to_clear[,1],
                   V1_value = matrix(1,number_of_rows),
                   V2_index = data_frame_to_clear[,2],
                   v2_value = matrix(1,number_of_rows),
                   V3_index = data_frame_to_clear[,3],
                   V3_value = matrix(1,number_of_rows)
                   )
  return(DT)
}
```

Za drugą natomiast odpowiada funkcja $fixMinusAndValue$:

```
separateIndexFromValue <- function(data_frame_to_clear){
fixMinusAndValue <- function(DT){
  for (i in 1:length(DT[,1])){
    for (j in seq(1, 6, 2)){
      if (DT[i,j] < 0){
        DT[i,j] = abs(DT[i,j])
        DT[i,j+1] = 0
      }
    }
  }
  return(DT)
}
```

Obie funkcje wywoływane są przez funkcję *cleanData*, dzięki czemu dane są gotowe do pracy z algorytmem genetycznym.

```
cleanData <- function(data_frame_to_clear){
  DT = separateIndexFromValue(data_frame_to_clear)
  DT = fixMinusAndValue(DT)
}
```

2.2 Funkcja fitness

Chcemy, by funkcja fitness przyporządkowywała lepszym wynikom coraz mniejsze wartości. Każdy literał, który jest prawdziwy zmniejsza ocenę o 1. Najlepsza ocena możliwa do uzyskania równa się liczbie literałów danej formuły.

```
fitnessFunc <- function(chromosome = c(), DT = cleanDataFrame){
  value = 0
  number_of_rows = length(DT[,1])
  for (i in 1:number_of_rows){
      if (chromosome[DT[i,1]] == DT[i,2] || chromosome[DT[i,3]] == DT[i,4]
       || chromosome[DT[i,5]] == DT[i,6]){
        value = value -1
      }
  }

  return(value)
}
```

2.3 Dobór parametrów

Wszystkie logi dostępne w repozytorium na GitHubie

2.3.1 10 literałów

Dla niewielkiej liczby literałów wystarczyły również stosunkowo niewielkie parametry $PopulationSize = 10$ oraz $NumberOfGenerations = 20$.

```
[1] "Liczba prawidlowych wynikow"
[1] 964
[1] "Liczba nieprawidlowych wynikow"
[1] 36
[1] "Sredni czas wykonania"
[1] 0.2175534 (s)
```

Występowanie wyników nieprawidłowych może być spowodowane skracaniem oryginalnych formuł.
Dobrane parametry są zadowalające.

2.3.2 20 literałów

Population Size = 10, Number of Generations = 20

```
[1] "Liczba prawidlowych wynikow"
[1] 881
[1] "Liczba nieprawidlowych wynikow"
[1] 119
[1] "Sredni czas wykonania"
[1] 0.332041 (s)
```

Population Size = 20, Number of Generations = 20

```
[1] "Liczba prawidlowych wynikow"
[1] 998
[1] "Liczba nieprawidlowych wynikow"
[1] 2
[1] "Sredni czas wykonania"
[1] 0.7549588 (s)
```

Czas wykonania wzrasta ponad dwukrotnie, lecz liczba zadowalających wyników jest niemal perfekcyjna.

2.3.3 50 literałów

Population Size = 10, Number of Generations = 20

```
[1] "Liczba prawidlowych wynikow"
[1] 414
[1] "Liczba nieprawidlowych wynikow"
[1] 586
[1] "Sredni czas wykonania"
[1] 0.7309625 (s)
```

Population Size = 50, Number of Generations = 20

```
[1] "Liczba prawidlowych wynikow"
[1] 991
[1] "Liczba nieprawidlowych wynikow"
[1] 9
[1] "Sredni czas wykonania"
[1] 3.221467 (s)
```

Znaczące zwiększenie zwiększa precyzję jak i czas wykonania skryptu.

2.3.4 100 literałów

Population Size = 10, Number of Generations = 20

```
[1] "Liczba prawidlowych wynikow"
[1] 39
[1] "Liczba nieprawidlowych wynikow"
[1] 961
[1] "Sredni czas wykonania"
[1] 1.244902 (s)
```

Population Size = 50, Number of Generations = 20

```
[1] "Liczba prawidlowych wynikow"
[1] 560
[1] "Liczba nieprawidlowych wynikow"
[1] 440
[1] "Sredni czas wykonania"
[1] 5.719035 (s)
```

Proponowany Population Size = 100

2.3.5 200 literałów

Population Size = 10, Number of Generations = 20

```
[1] "Liczba prawidlowych wynikow"
[1] 0
[1] "Liczba nieprawidlowych wynikow"
[1] 10000
[1] "Sredni czas wykonania"
[1] NaN
```

Średni czas jest liczony jedynie dla prawidłowych wyników

Population Size = 50, Number of Generations = 20

```
[1] "Liczba prawidlowych wynikow"
[1] 1
[1] "Liczba nieprawidlowych wynikow"
[1] 999
[1] "Sredni czas wykonania"
[1] 10.602 (s)
```

Population Size = 150, Number of Generations = 50

```
[1] "Liczba prawidlowych wynikow"
[1] 102
[1] "Liczba nieprawidlowych wynikow"
[1] 153
[1] "Sredni czas wykonania"
[1] 1.2398 (mins)
```

Ze względu na znaczny wzrost czasu wykonania (głównie przez zwiększenie liczby generacji) zmniejszona została liczba testowanych formuł.
Liczba wyników prawidłowych pozostaje niezadowalająca.
Proponowane: Population Size = 300, Number of Generations = 100

2.3.6 300 literałów

Population Size = 10, Number of Generations = 20

```
[1] "Liczba prawidlowych wynikow"
[1] 0
[1] "Liczba nieprawidlowych wynikow"
[1] 1000
[1] "Sredni czas wykonania"
[1] NaN
```

Population Size = 150, Number of Generations = 50

```
[1] "Liczba prawidlowych wynikow"
[1] 2
[1] "Liczba nieprawidlowych wynikow"
[1] 184
[1] "Sredni czas wykonania"
[1] 2.0705 (mins)
```

O wiele za małe wartości testowe.
Proponowane: Population Size = 450, Number of Generations = 100

2.3.7 403 literały

Population Size = 150, Number of Generations = 50

```
[1] "Liczba prawidlowych wynikow"
[1] 0
[1] "Liczba nieprawidlowych wynikow"
[1] 143
[1] "Sredni czas wykonania"
[1] 2.0705 (mins)
```

O wiele za małe wartości testowe.
Proponowane: Population Size = 600, Number of Generations = 100

3 DPLL

Użyta implementacja algorytmu dostępna jest pod adresem `https://github.com/ashwinkachhara/3dpll`.
Użyty język: Python

4 Porównanie

Dla obu algorytmów użyto tych samych danych testowych.
Otrzymano następujące wyniki:

Literały	GA	DPLL
10	0.140135[s]	0.022[s]
20	1.36323[s]	0.026[s]
50	18.94206[s]	0.039[s]
100	39.86838[s]	0.108[s]
200	1.831965[min]	4.287[s]
300	4.528744[min]	0.995[s]
400	16.28383[min]	7[min]17.359[s]
403	12.86907[min]	13[min]12.759[s]