Rozwiązywanie 3-SAT - algorytm DPLL vs strategie ewolucyjne

Grams, Stanisław (251000)
MFI UG
Inteligencja Obliczeniowa

26 stycznia 2020

1 Wstęp

Tematem projektu jest porównanie wydajności jednego algorytmu i dwóch heurystyk pozwalających na rozwiązywanie problemów 3-SAT. Problem spełnialności (SAT) jest fundamentalnym problemem NP-zupełnym - czyli takim do którego każdy problem z klasy NP można zredukować w czasie wielmianowym. Odpowiednio szybki algorytm SAT mógłby zostać zastosowany m.in. w planowaniu w czasie rzeczywistym, obliczeniach charakterystycznych dla elektroniki czy astronautyki, gdzie znaleźć należy optymalne rozwiązanie problemu klasy NP-zupełnego.

1.1 Narzędzia

Programy zaimplementowano od podstaw w języku *Python 3*[1] oraz użyto nastepujących bibliotek zewnętrznych:

- matplotlib[2] - biblioteka pozwalająca na stworzenie wykresów z poziomu pythona
- pandas[3] - biblioteka pozwalająca na łatwą kontrolę nad plikami csv

1.2 Dane

Dane wejściowe pochodzą ze strony internetowej *cs.ubc.ca/ hoos/SATLIB/benchm.html*[4] - agregującej dane wejściowe dla problemów typu SAT. Plik importowany przez program zdefiniowany jest w specyficznym dla problemów SAT formacie DIMACS[5].

1.3 Zaimplementowane oraz użyte algorytmy

- (**DPLL**): Współczesny algorytm DPLL[6][7]
- (**SGA**): Standardowy algorytm genetyczny z możliwością elityzmu[8]
- (**IAGA**): Adaptacyjny algorytm genetyczny zaimplementowany na podstawie publikacji[9]

Testy zostały wykonane dla zmiennych $L = 20..90$ i klauzul $M = 10..449$.

2 SGA: Standardowy algorytm genetyczny

Zaimplementowano standardowy algorytm genetyczny oraz przeprowadzono testy. Zadziwiająco, dla dowolnej liczby zmiennych oraz klauzul najlepszymi okazały się poniższe parametry, oraz to własnie ich użyto w celu dalszej ewaluacji rozwiązania.

[1] https://www.python.org/
[2] https://matplotlib.org/
[3] https://pandas.pydata.org/
[4] https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html
[5] https://logic.pdmi.ras.ru/~basolver/dimacs.html
[6] https://en.wikipedia.org/wiki/DPLL_algorithm
[7] https://www.win.tue.nl/~jschmalt/teaching/2IMF20/BMC-lecture_slides_SAT.pdf
[8] https://pl.wikipedia.org/wiki/Algorytm_genetyczny
[9] https://www.atlantis-press.com/journals/ijcis/25888772/view&sa=D&ust=1579988144931000

nazwa współczynnika	wartość
crossover rate	0.90
mutation rate	0.08
population size	50
generations	1000
elitism	True

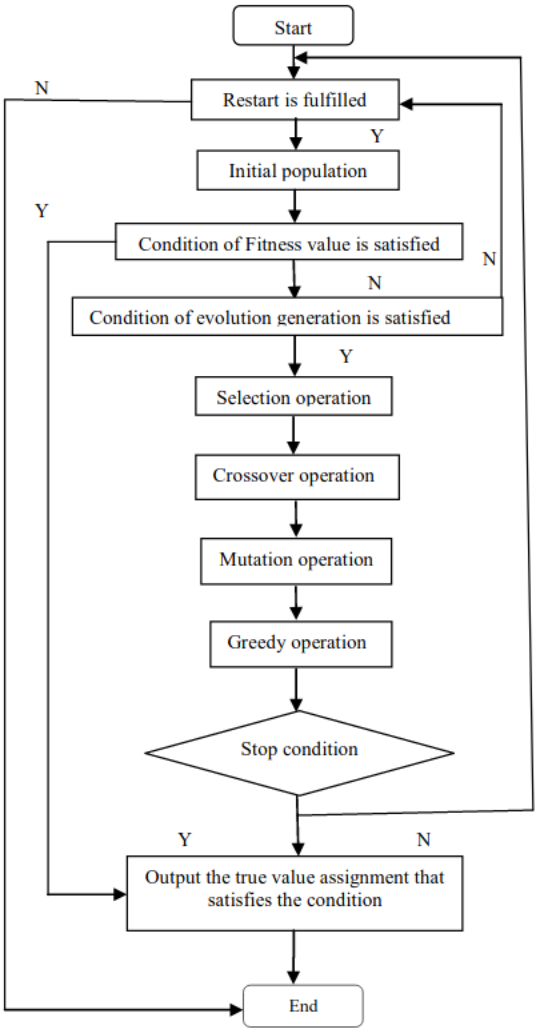
Funkcję fitness w przypadku **obu algorytmów genetycznych** można określić poniższym wzorem:

$$F = \frac{S}{C}$$

gdzie S oznacza ilość spełnionych klauzul przez chromosom, a C oznacza ilość wszystkich klauzul.

3 IAGA: Improved Adaptive Genetic Algorithm

Głównymi róznicami między typowym algorytmem genetycznym rozwiązującym problemy 3-SAT a zaimplementowanym ulepszonym, adaptywnym algorytmem genetycznym wynikają wprost z nazwy publikacji, w której się ukazał *An Improved Adaptive Genetic Algorithm for Solving 3-SAT Problems Based on Effective Restart and Greedy Strategy.*



Flow chart heurystyki **IAGA**

Heurystyka ta wykazuje się trzema głównymi zmianami w stosunku do standardowego algorytmu genetycznego:

1. adaptywnie w trakcie ewolucji, na podstawie poczynionych wyborów wylicza prawdopodobieństwo operacji crossover i mutate
2. dodaje operacje greedy - zachłanną strategię poszukiwania optymalnego rozwiązania

3. dodaje operacje restart - rozwiązanie problemu wpadania w lokalne minimum przy kolejnych ewolucjach

Algorytm ten dodaje również następujące nowe zmienne, które pozwalają zmieniać jego zachowanie. Poniżej przedstawiono wartości, które sprawdziły się najlepiej.

nazwa współczynnika	**wartość**
restart ratio	100
fzero	0.75

4 Opis eksperymentów

4.1 Konfiguracja środowiska

Uruchomienia programu wykonane zostały na nastepującej konfiguracji sprzętoweji:

- jednostka centralna Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz
- 12GB pamięci ram DDR4

Program uruchamiano na systemie Arch Linux z jądrem Linux 5.4.13-arch-1-1.

4.2 Przebieg eksperymentu i opis programu

Program *main.py* uruchamia każdy z wcześniej wspomnianych algorytmów na określonych danych wejściowych.

```
$ python main.py -f ../data/test/M100_L50.cnf -o csvs/M100_L50
isFileValid == True
testing DPLL...
testing SGA...
testing IAGA...
```

Przykładowe wywołanie programu *main.py*

Wywołanie to przyjmuje nastepujące parametry:

- f: plik z danymi w formacie DIMACS
- o: katalog docelowy dla plików wyjściowych w formacie csv
- i: liczba iteracji/wykonań pojedynczego algorytmu

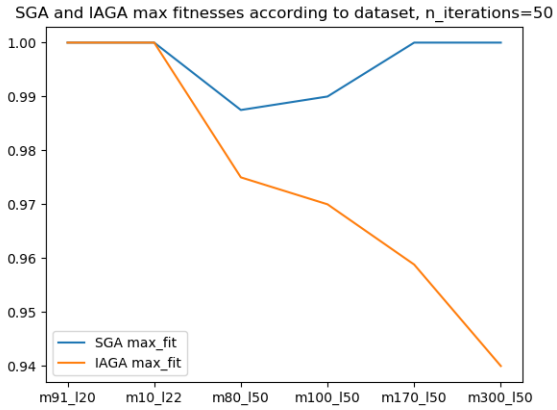
```
# sjg @ t460s in ~/git/ci/09-3sat_project/src on git:master x [23:23:27]
$ python main.py
Usage: python main.py -f /path/to/DIMACS_file.cnf -o /path/to/output_dir/ -i iterations
```

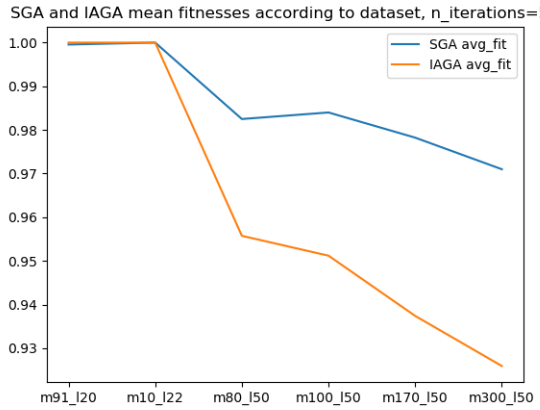
Szereg wywołań programu *main.py* stworzył poszczególne pliki opisujące wyniki pojedynczego eksperymentu.

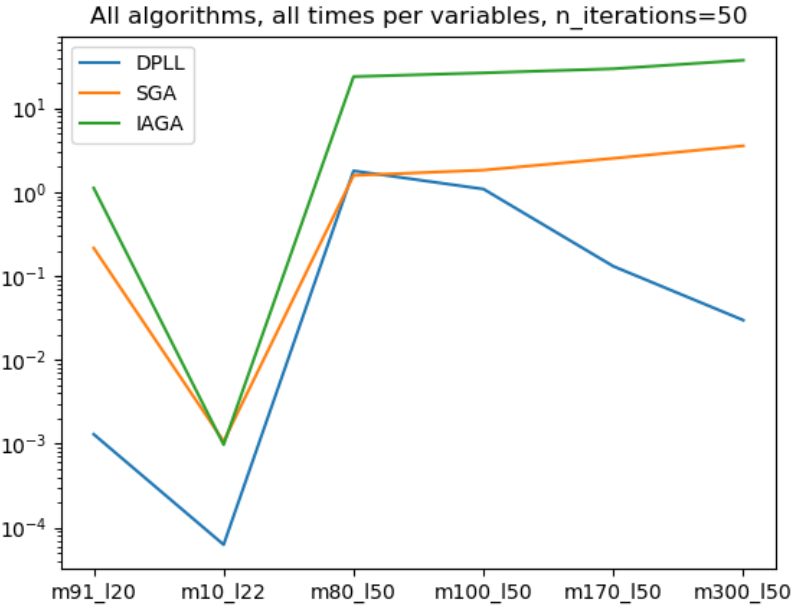
```
# sjg @ t460s in ~/git/ci/09-3sat_project/report on git:master x [23:26:54]
$ ls ../src/csvs/M300_L50
DPLL_M300_L50-20200125214008-results.csv  IAGA_M300_L50-20200125214008-results.csv  SGA_M300_L50-20200125214008-results.csv
```

Przy użyciu zaimplementowanego skryptu *graph.py* stworzono wykresy będące podstawą analizy wyników.
Wykonanie algorytmów w wielu iteracjach pozwoliło na uśrednienie wyników pomiarów czasu zaprezentowanych na wykresach

5 Wyniki







6 Interpretacja wyników

6.1 Wnioski

1. Zgodnie z oczekiwaniami algorytm **DPLL** zawsze gdy było to możliwe (dane wejściowe SAT) zwracał prawidłowe rozwiązanie w krótkim czasie

2. Na czas wykonywania algorytmów wpływ ma nie tylko ilość klauzul ale również ilość zmiennych

3. Zaimplementowany algorytm **SGA** okazał się równy bądź nawet lepszy od danych przedstawionych w *An Improved Adaptive Genetic Algorithm for Solving 3-SAT Problems Based on Effective Restart and Greedy Strategy*

4. Gdy algorytm **IAGA** nie był wstanie znaleźć spełnialnego rowiązania dla dużej ilości zmiennych $L >= 50$, bądź dla dużej ilości klauzul $M >= 150$ czas jego wykonania **rósł ze względu na ilość wykonywanych restartów**.

5. Algorytm **IAGA** zwraca zdecydowanie więcej rozwiązań spełniających oraz zdecydowanie szybciej dla datasetu, który również jest spełnialny przez algorytm **SGA**.

6. Zachodzi następująca prawidłowość: im mniejsza ilość zmiennych w problemie, tym krótszy czas wykonania algorytmu.

6.2 Możliwe błędy

Niestety, publikacja naukowa, na której oparto implementacje algorytmu IAGA nie jest wystarczająco pełnowartościowa. Na szczególną uwagę zasługują niejasne rozdziały na temat zastosowanych strategii - w szczególności algorytm strategii wyboru oraz strategii zachłannej.
W związku z niewystarczającym opisem słownym, wymagane były eksperymentalne zaimplementowalnie strategii zachłannej na podstawie artykułu. Jednym z przykładów może być podatność na występowanie dzielenia przez zero w opisie funkcji adaptycznie liczącej współczynniki mutacji i rekombinacji.

$$p_c = \begin{cases} 0.8 * \dfrac{f_{\max} - \overline{f}}{\overline{f} - f_{\min} + \lambda} & , \ \dfrac{f_{\max} - \overline{f}}{\overline{f} - f_{\min} + \lambda} < 1 and M_1 > M_2 \\ p_1 - \dfrac{0.3 * \left(f^{'} - f_{\min}\right)}{f_{\max} - f_{\min}} & , \ otherwise \end{cases} \tag{4}$$

$$p_m = \begin{cases} 0.1 * \dfrac{f_{\max} - \overline{f}}{\overline{f} - f_{\min} + \lambda} & , \ \dfrac{f_{\max} - \overline{f}}{\overline{f} - f_{\min} + \lambda} < 1 and M_1 > M_2 \\ p_2 - \dfrac{0.09 * \left(f_c - f_{\min}\right)}{f_{\max} - f_{\min}} & , \ otherwise \end{cases} \tag{5}$$

When $g > 0.75 * G$;

$$p_c = \begin{cases} 0.8 * \dfrac{f_o - f_{\max}}{f_o - \overline{f}} & , \ \dfrac{f_{\max} - \overline{f}}{\overline{f} - f_{\min} + \lambda} < 1 and M_1 > M_2 \\ p_1 - \dfrac{0.3 * \left(f_0 - f_{\max}\right)}{f_o - f^{'}} & , \ otherwise \end{cases} \tag{6}$$

$$p_m = \begin{cases} 0.1 * \dfrac{f_o - f_{\max}}{f_o - \overline{f}} & , \ \dfrac{f_{\max} - \overline{f}}{\overline{f} - f_{\min} + \lambda} < 1 and M_1 > M_2 \\ p_2 - \dfrac{0.09 * \left(f_0 - f_{\max}\right)}{f_o - f_c} & , \ otherwise \end{cases} \tag{7}$$