Raport z wykonania zadania

Maciej Ciepiela

December 26, 2024

1 Część A

1.1 Opis rozwiązania

Do generowania zróżnicowanych grafów stworzyłem klase RandomGraph, która automatycznie inicjalizuje się na podaną liczbę wierzchołków. Po czym losuje liczbę krawędzi z przedziału od $n-2$ do n^2, gdzie n to liczba wierzchołków. Następnie losuje wierzchołki, które połączy krawędź. Liczba krawędzi może wydawać się duża, ale jest celowa ze względu na potrzeby zadania i fakt, że możemy wylosować połączone już wierzchołki, a co za tym idzie dodamy mniej faktycznych krawędzi.

1.2 Algorytm

Algorithm 1 Losowanie krawędzi grafu

```
G = (V, E) - zainicjonowany graf
r ← losowa liczba z przedziału [|V| − 2, |V|^2]
for i ← 1 to r do
    u ← losowy wierzchołek z przedziału [0, |V| − 1]
    v ← losowy wierzchołek z przedziału [0, |V| − 1]
    if u ≠ v and (v, u) ∉ E and (u, v) ∉ E then
        dodaj (u, v) do krawędzi
        dodaj v do sąsiadów u
        dodaj u do sąsiadów v
        zwiększ licznik krawędzi o 1
    end if
end for
```

2 Część F

2.1 Opis rozwiązania

Do rozwiązania części F wykorzystałem biblioteke z3 i zawarte w niej funkcje oraz solver dla SMT. Dla każdego wierchołka tworze zmienną boolowską, która określe przynależność wierzchołka do vertex cover. Następnie dla każdej krawędzi tworze klauzulę, która mówi, że przynajmniej jeden z końców krawędzi jest pokryty, tj. przynajmniej jeden wierzchołek jest w vertex cover. Na koniec dodaje ograniczenie, że suma wszystkich wybranych wierzchołków w vertex cover jest mniejsza bądź równa podanemu k. Potem wykorzystując Solver dostępny w bibliotece z3 sprawdzam czy istnieje taki vertex cover, który spełnia warunki.

2.2 Algorytm

Algorithm 2 Obliczanie vertex cover z wykorzystaniem smt-solvera

```
1:  G = (V, E) ← graf
2:  k ← przybliżony rozmiar vertex cover
3:  solver ← nowy Solver() z biblioteki z3
4:  vertex_vars ← słownik zmiennych boolowskich odpowiadających wierzchołkom
5:  for (u, v) in E do
6:      solver.add(Or(vertex_vars[u], vertex_vars[v]))
7:  end for
8:  solver.add(Sum([If(vertex_vars[v], 1, 0) for v in V]) ≤ k)
9:  solver.check()
10: if solver.check() == sat then
11:     model ← solver.model()
12:     vertex_cover ← [v for v in vertex_vars jeśli model[vertex_vars[v]]]
13:     return vertex_cover
14: else
15:     return None
16: end if
```

3 Część H

3.1 Opis rozwiązania

Wykorzystując dodatkową funkcje wykonałem pomiary czasów działania algorytmów na tym samym zbiorze wylosowanych grafów. Zbiór ten zawierał 120 grafów po 3 o ilości wierzchołków równej od 1 do 40. (Ze zwględu na słabą wydajność brute force dla większych grafów i możliwości urządzenia). Ze względu na długi czas działania wyniki zapisałem w osobnych plikach, a następnie przedstawiłem w postaci wykresów, aby lepiej zobrazować różnice pomiędzy algorytmami.

3.2 Wyniki

Uzyskane wyniki przedstawiłem w postaci wykresów, aby lepiej zauważyć zachodzące zależności.

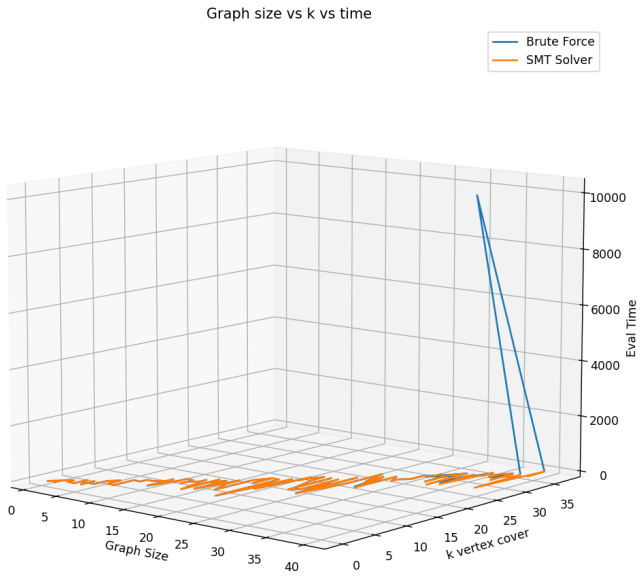

Figure 1: Porównanie działania algorytmów w 3d

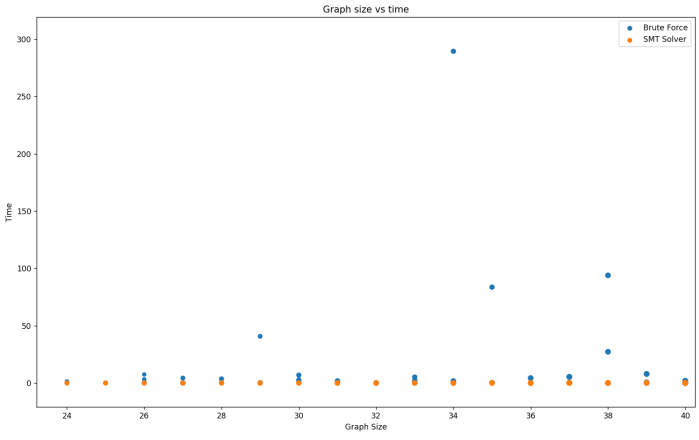

Figure 2: Zbliżenie na różnice

3.3 Wnioski

Jak widać na wykresie, algorytm brute force jest znacznie wolniejszy od algorytmu z wykorzystaniem smt-solvera. Natomiast rozpatrując każdy algorytm z osobna można zauważyć, że dla brute force rozmiar grafów nie masz aż tak dużego znaczenia, tym co wpływa na jego wydajność jest rozmiar szukanego vertex cover (rozmiar punktów na wykresie). Z kolei dla smt-solvera większe znaczenie ma rozmiar grafu niż szukany vertex cover, jednak nie wpływa to znacząco na jego czas działania. Na podstawie tych obserwacji można wywnioskować, że algorytm z wykorzystaniem smt-solvera jest lepszym rozwiązaniem dla tego problemu. Należy też brać lekką poprawkę na ograniczenia urządzenia i fakt, że podczas obliczeń mogły wystąpić pewne błędy spowodowane czynnikami zewnętrznymi.

4 Część I

4.1 Opis rozwiązania

W celu rozwiazania tego zadania, dla każdego algorytmu z osobna, empirycznie generowałem coraz większy graf i mierzyłem czas działania programu, dopóki nie natrafiłem na wynik większy od 2 minut. Starałem się również sprawdzać, aby generowany graf miał stosunkowo duży przybliżony rozmiar vertex cover, żeby był to bardziej realny przypadek i móc dokładniej określić najlepsze wyniki. Przy doborze rozmiaru sugerowałem się także wynikami z części H.

4.2 Wyniki

Algorytm	Rozmiar Grafu	Vertex Cover	Czas działania (sek.)
Brute force	30	17	130
SMT-solver	2400	2392	129

Table 1: Tabela przybliżonych wyników dla części I

4.3 Wnioski

W trakie mierzenia brute force pojawiły się pewne problemy z realnym wynikiem ze względu na fakt, że trzeba zasymulować dokładnie najgorszy przypadek, sugeruje to, że czas działania nie jest dużo zależny od liczby wierzchołków lub rozmiaru vertex cover, a bardziej od samego rozkładu grafu i poprawnych w nim wierzchołków. Zatem można stwierdzić, że brute force mógłby szybko sobie poradzić z bardzo dużym grafem, ale zależałoby to od jego struktury i ustawienia wierzchołków z vertex cover. Natomiast widać, że smt-solver jest bardziej stabilny i niezależny od tego, jak zbudowany jest graf, sprawiając, że jest on lepszym rozwiązaniem dla tego problemu.