

Raport z wykonania zadania

Maciej Ciepiela

December 24, 2024

1 Część A

1.1 Opis rozwiązania

Do generowania zróżnicowanych grafów stworzyłem klasę 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

```
1:  $G = (V, E)$  - zainicjonowany graf
2:  $r \leftarrow$  losowa liczba z przedziału  $[|V| - 2, |V|^2]$ 
3: for  $i \leftarrow 1$  to  $r$  do
4:    $u \leftarrow$  losowy wierzchołek z przedziału  $[0, |V| - 1]$ 
5:    $v \leftarrow$  losowy wierzchołek z przedziału  $[0, |V| - 1]$ 
6:   if  $u \neq v$  and  $(v, u) \notin E$  and  $(u, v) \notin E$  then
7:     dodaj  $(u, v)$  do krawędzi
8:     dodaj  $v$  do sąsiadów  $u$ 
9:     dodaj  $u$  do sąsiadów  $v$ 
10:    zwiększ licznik krawędzi o 1
11:   end if
12: end for
```

2 Część F

2.1 Opis rozwiązania

Do rozwiązania części F wykorzystałem bibliotekę z3 i zawarte w niej funkcje oraz solver dla SMT. Dla każdego wierzchołka tworzę zmienną boolowską, która określa przynależność wierzchołka do vertex cover. Następnie dla każdej krawędzi tworzę 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:  $G = (V, E) \leftarrow$  graf
2:  $k \leftarrow$  przybliżony rozmiar vertex cover
3:  $solver \leftarrow$  nowy Solver() z biblioteki z3
4:  $vertex\_vars \leftarrow$  zmienne boolowskie odpowiadające 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  $V$ ])  $\leq k$ )
9:  $solver.check()$ 
10: if  $solver.check() == \text{sat}$  then
11:    $model \leftarrow solver.model()$ 
12:    $vertex\_cover \leftarrow [v$  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 funkcję wykonałem pomiary czasów działania algorytmów na tym samym zbiorze wylosowanych grafów. Zbiór ten zawierał 100 grafów o maksymalnie 40 wierzchołkach. (Ze względu na słabą wydajność brute force dla większych grafów i możliwości urządzenia).

3.2 Wyniki

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

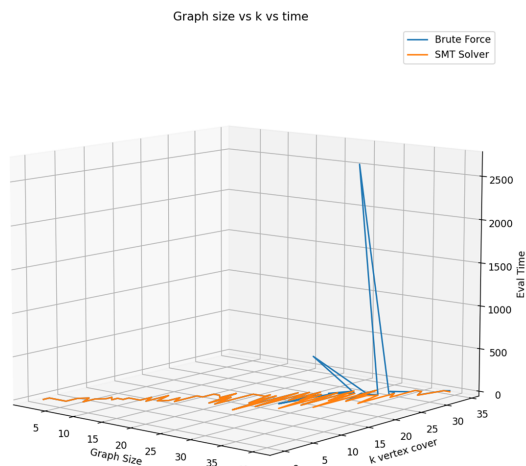


Figure 1: Porównanie działania algorytmów w 3d

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 ma aż tak dużego znaczenia, tym co wpływa na jego wydajność jest rozmiar szukanego vertex cover. 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 rozwiązania 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 trakcie 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, co sugeruje, że jest to lepsze rozwiązanie dla tego problemu.