Algorytmika Praca domowa nr 2

Kamil Pilkiewicz

Kwiecień 2024

Zadanie 2

Treść

Głównym celem zadania będzie zaimplementowanie kilku metod, z których każda będzie implementowała algorytm dla znajdowania naliczniejszego skojarzenia w grafie dwudzielnym. Każda taka metoda powinna odczytywać graf z pliku, dokonać obliczeń, a następnie zwrócić wynik. Czas obliczeń każdej metody może zostać zmierzony przy użyciu biblioteki time.

a

Zaimplementuj metodę HK, która implementuje algorytm Hopcrofta-Karpa. Algorytm Hopcrofta-Karpa znajduje skojarzenie w grafach dwudzielnych w czasie O(|E|·p|V |), zaś jego opis można znaleźć np. tu: $https://en.wikipedia.org/wiki/Hopcroft-Karp_algorithm$ Twoja implementacja powinna być efektywna, t.j. faktycznie działać w czasie O(|E|p|V |).

b

Korzystając z biblioteki python-mip, zaimplementuj metodę, która rozwiązuje problem skojarzeń dwudzielnych za pomocą programu liniowego. Będziemy testować dwa warianty tej metody, różniące się wyłącznie zastosowanym solverem. Niech metoda LPcbc oznacza wariant używający domyślny solver CBC, natomiast jako LPgur oznaczymy wariant używający solvera GUROBI.

c

Porównaj działanie metod HK, LPcbc oraz LPgur na grafach losowych 300 × 300 wierzchołków przy prawdopodobieństwie krawędzi rosnącym od 0 do 1. Narysuj stosowny wykres używając biblioteki pyplot.

d

Dla dowolnego n podaj przykład grafu o 2n wierzchołkach, dla którego algorytm Hopcrofta-Karpa wykona $\Omega(\sqrt{n})$ faz, czyli zadziała w czasie $\Omega(|E|p|V|)$.

e

Porównaj czasy działania metod HK, LPcbc i LPgur na instancjach trudnych dla algorytmu Hopcrofta-Karpa. Narysuj stosowny wykres czasów działania przy rosnącym rozmiarze instancji n, kończąc na n rzędu 500000.

f

Zaproponuj jak najlepszą heurystykę która na tych instancjach działa wydajniej niż HK. Zaimplementuj ją nazywając stosowną metodę AlternativeToHK. Czy potrafisz znaleźć trudny przykład dla Twojej heurystyki ?

f*

Alternatywnie, metoda AlternativeToHK może implementować algorytm rangowy z pracy 'Online bipartite matching in offline time', który również znajduje skojarzenie w grafie dwudzielnym w czasie O(|E|p|V |), ale w zupełnie inny sposób niż algorytm Hopcrofta-Karpa. Algorytm rangowy jest ciekawy i prosty w implementacji. Pracę 'Online bipartite matching in offline time', gdzie można go znaleźć, udostępnimy na platformie moodle. Współautorka tej pracy chętnie wytłumaczy ten algorytm chętnym w ramach konsultacji.

g

Przetestuj metody HK, LPcbc, LPgur i AlternativeToHK na instancjach własnego pomysłu. Narysuj stosowne wykresy porównawcze. Czy na podstawie testów możesz wywnioskować coś na temat asymptotycznych czasów działania zastosowanych solverów?

Rozwiązanie

Praktycznie wszystko jest w notebook, włącznie z wyjaśnieniami. Postanowiłem jedynie wykresy zamieścić tu, wnioski, opisy konstrukcji.

Uwagi

Uwagi:

- Pomyliłem nazwy solverów: PLgur z LPgur, PLcbc z LPcbc. Przepraszam.
- HK nie zamienia U z V, co mogłoby zmiejszyć czas czasami. Ale Pani Zych powiedziała, że to nie problem.
- Ta reprezentacja grafu jest w porządku - tak twierdzi Pani Zych.
- Reprezentacja grafu w pliku:

  U

  V

  m

  W kolejnych m wierszach będą krawędzie, np. "0 3" oznacza krawędź od 0 z 'U' do 3 z 'V'.

Opis API

Definicja algorytmów rozwiązujących matching grafów dwudzielnych: Funkcje są postaci: def nazwa_(U, V, E, debug=False), np. def HK_(U, V, E) gdzie:

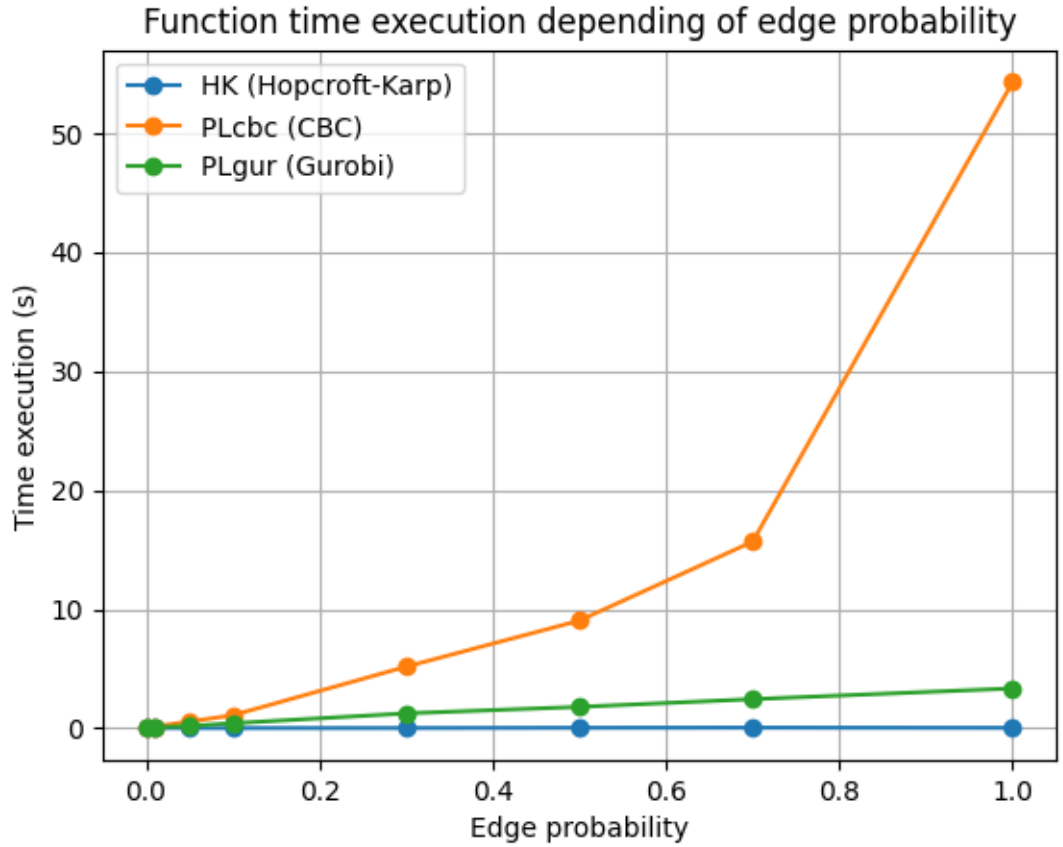
- U to liczba wierzchołków w częsci lewej 'U', zakładamy, że numery wierzchołków to 0, 1, ..., U - 1,
- V to liczba wierzchołków w 'V', zakładamy, że numery wierzchołków to 0, 1, ..., V - 1,
- E to krawędzie między U oraz V, są w postaci listy sąsiedztwa, ma ona dokładnie U elemtów, każdy element to np.array([int]), gdzie liczby w elemencie to numery wierzchołków w 'V' do których istnieje krawędź z odpowiedniego wierzchołka z 'U',
- debug = True oznacza wersję testową, będą wyświetlały się informacje diagnostyczne, debug = False oznacza, że nie będą wyświetlały się informacje diagnostyczne(oprócz tych z modeli CBC, GUROBI).

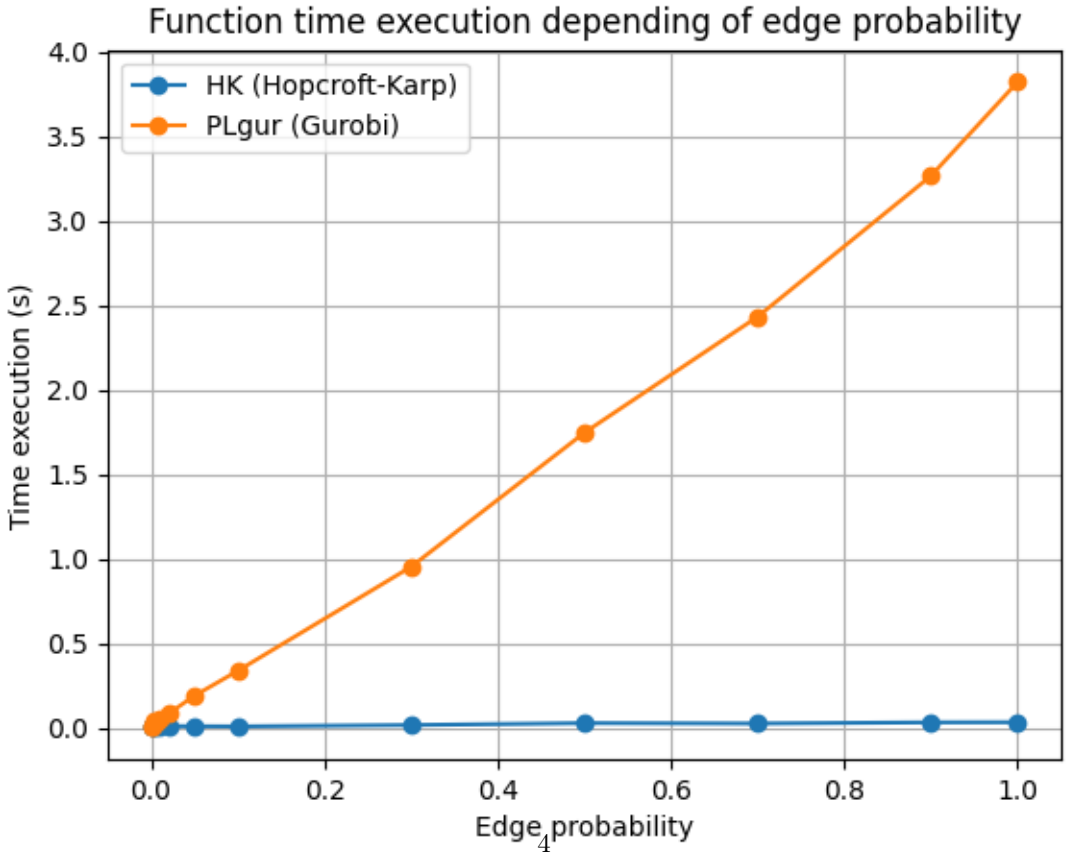
Po nazwie jest znak '_', który oznacza, że API funkcji jest jak wyżej. Zrobiłem tak, ponieważ w opisie zadania jest wymóg, aby czytać graf z pliku. Te funkcje są postaci: def nazwa(), np. def HK(). Są zamieszczone na końcu.

Generatorka grafu z prawdopodobieństwem krawędzi p

Korzystam z numpy, dzięki czemu jest szybciej. generateEdgesWithProbability - to nazwa tej funkcji.

c - wykresy


Function time execution depending of edge probability
HK (Hopcroft-Karp)
PLcbc (CBC)
PLgur (Gurobi)
Time execution (s)
0
10
20
30
40
50
0.0
0.2
0.4
0.6
0.8
1.0
Edge probability


Function time execution depending of edge probability
HK (Hopcroft-Karp)
PLgur (Gurobi)
Time execution (s)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0.0
0.2
0.4
0.6
0.8
1.0
Edge probability

d

Opis konstrukcji grafu: różne spójne składowe wcale nie są zupełnie niezależne w algorytmie HK. Wystarczy aby graf miał spójne składowe w postaci rozłącznych wierzchołkowo ścieżek, niech te ścieżki mają 1, 3, 5, 7, ..., (2 * d + 1) krawędzi, niech kolejność krawędzi w listach sąsiedztwa najpierw zablokuje w 1 fazie HK wewnętrzne krawędzie, czyli każdą ścieżkę oprócz 1 można powiększyć ścieżką powiększającą alternującą o 1. Przykład: takim grafem jest graf dla którego n = U = V = 10 oraz E = [array([0]), array([2, 1]), array([2]), array([4, 3]), array([5, 4]), array([5]), array([7, 6]), array([8, 7]), array([9, 8]), array([9])]. Tutaj wierzchołki na krótszej ścieżce mają mniejsze numery, pierwszy wierzchołek na ścieżce należy do 'V', potem do 'U', itd., ostatni należy do 'U'. Każdy należący do 'U' na ścieżce, oprócz ostatniego, najpierw łączy się z wierzchołkiem należącym do 'V' o większym numerze, a potem mniejszym. Ostatni z 'U' łączy się z tylko 1 wierzchołkiem. Więcej informacji o konstrukcji będzie widać w samym algorytmie. Szczegół dla ustalonego n: suma wierzchołków z 'U' z takich ścieżek nie musi wynosić dokładnie n, ale będzie tego rzędu. Budujemy największy taki graf jaki jest możliwy, a pozostałe wierzchołki ustawiamy jako rozłączne ze wszystkim.

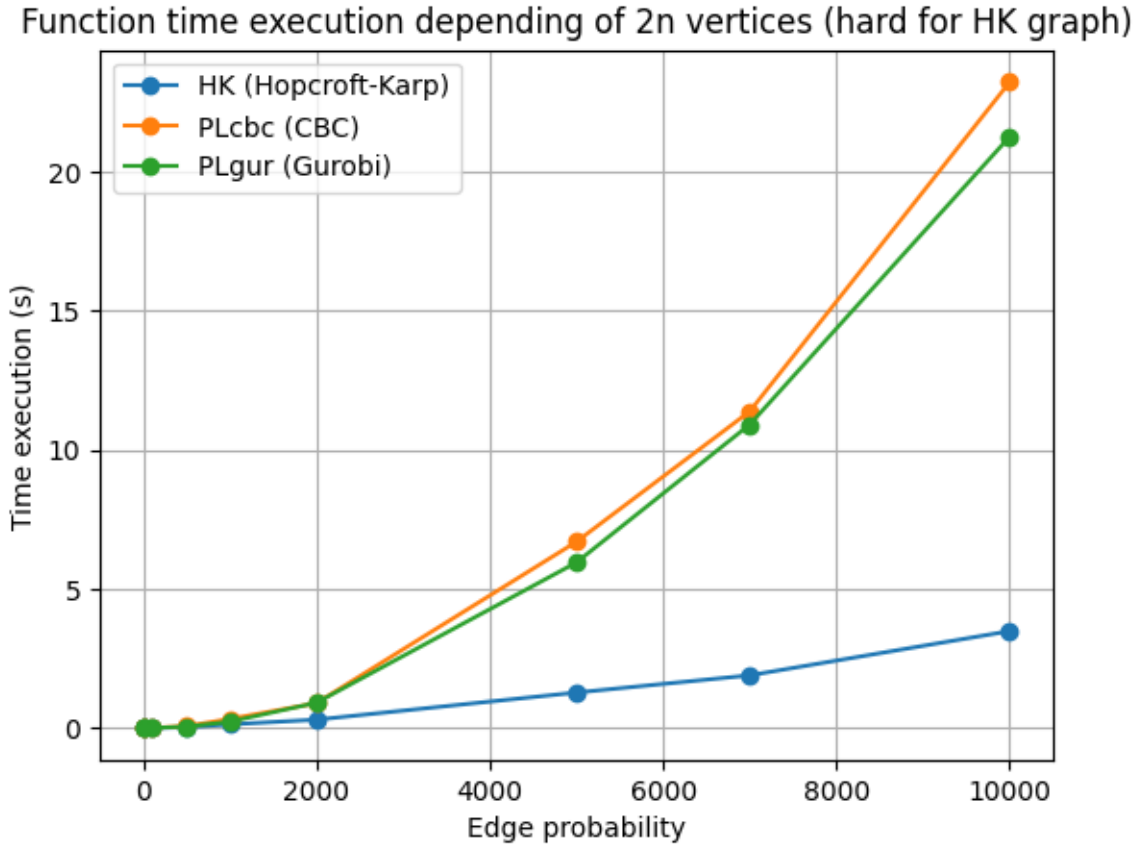
Czemu to działa? Po pierwsze ścieżka o (2 * k + 1) krawędziach ma k wierzchołków z 'U' i tyle samo z 'V'. Szukamy więc d największego, które spełnia 1 + 2 + ... + d <= n. Ale 1 + 2 + ... + d = $\theta(d^2)$, więc 1 + 2 + ... + d = $\theta(n)$ (różnica = $O(\sqrt(n))$). Ograniczmy się więc do przypadku, gdy 1 + 2 + ... + d = n. Zauważmy: |E| = O(V) Po przejściu 1 fazy każdą ścieżkę (oprócz 1-szej) można rozszerzyć o 1, tylko na 1 sposób - ścieżka powiększająca zaczyna się na ostatnim wierzchołku z 'U'. Ale BFS sprawi, że w danej fazie tylko 1 ścieżkę można rozszerzyć - w 2 fazie o długości 3, w 3 fazie - o długości 5 itd., bo w każdej fazie pozostałe ścieżki niepowiększone są dłuższe od tej ścieżki. Zatem faz będzie d, czyli $\sqrt(V)$. To nie oznacza jeszcze, że długość fazy zajmie O(|E|). Ale tak będzie, bo tak się składa, że ograniczając się do faz od (d/2) do (3/4 d) faz długość ścieżki powiększającej dla fazy będzie >= (2 * d + 1) / 2 >= d, ścieżek po których przejdzie jest co najmniej (d/4) (ostatnie ścieżki), więc długość tych faz >= $\Omega(d^2)$, tych faz jest $\theta(d)$, co daje złożoność $\theta(d^(3/2) = \theta(|E| \cdot \sqrt(V)))$ - tak miało być.

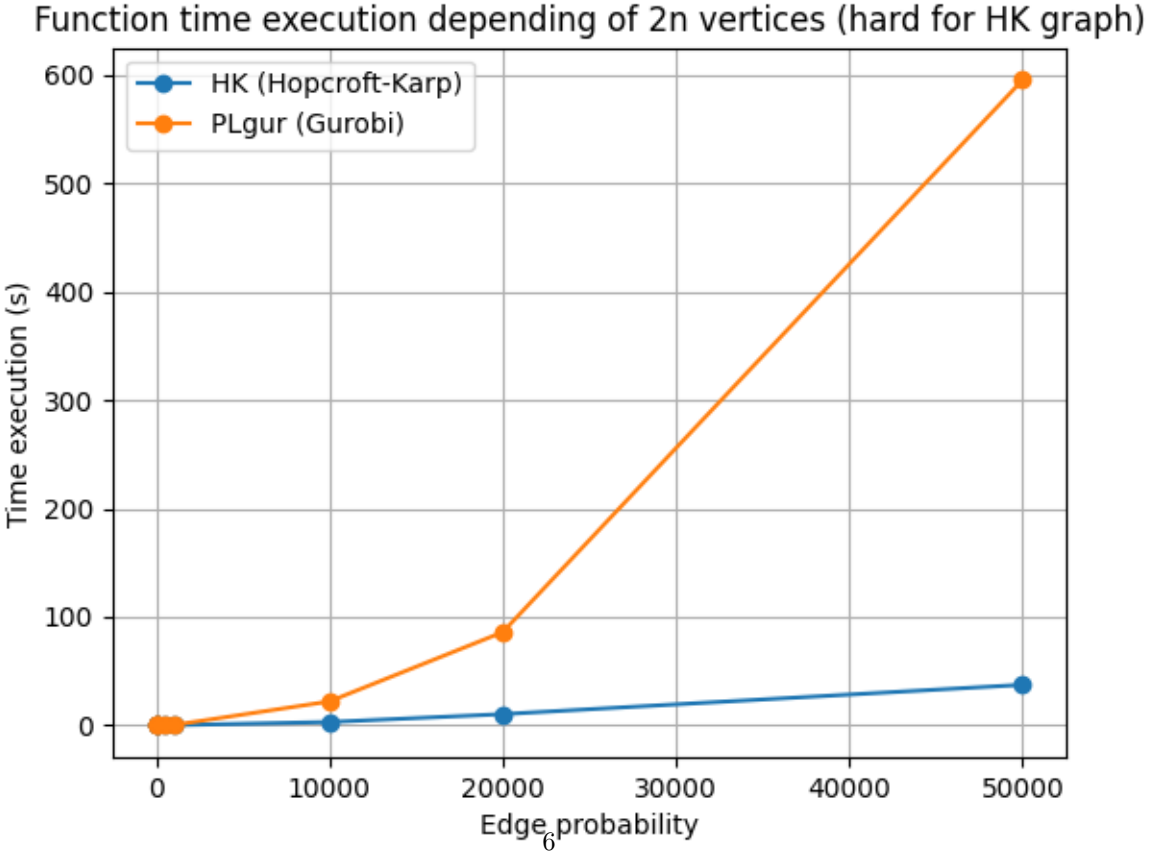
f

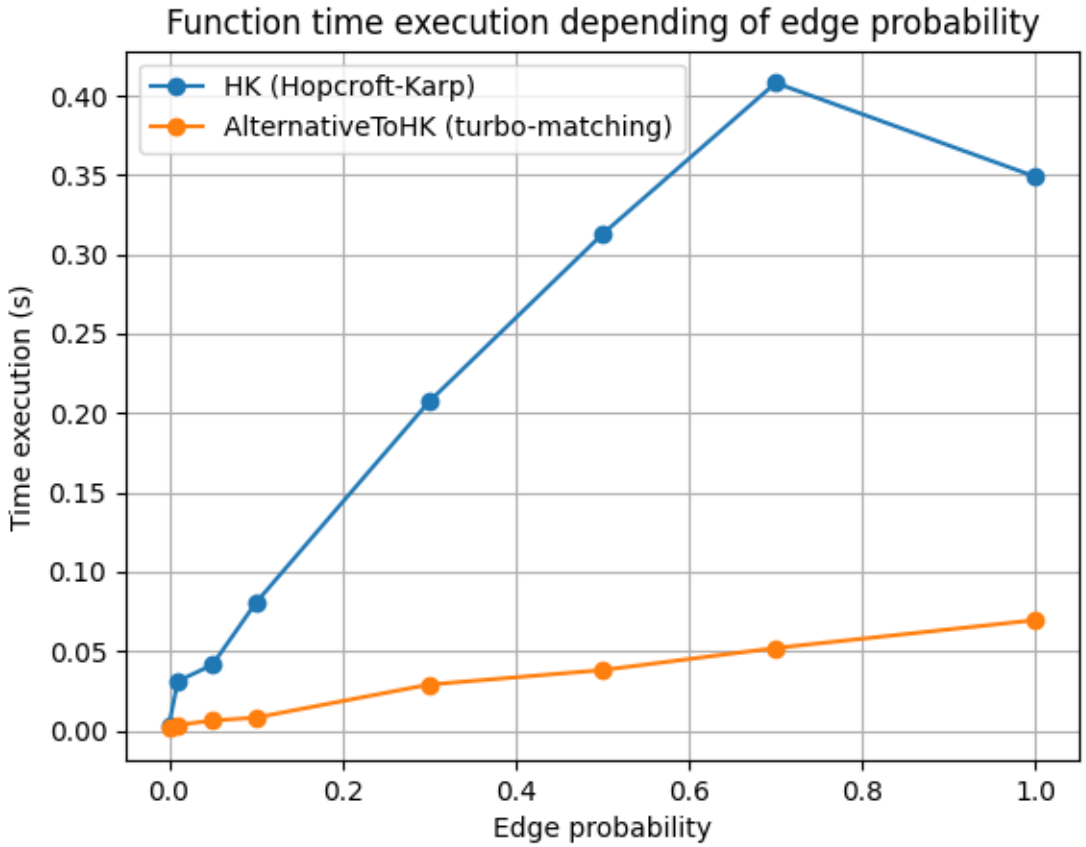
Algorytm AlternativeToHK_ to implementacja typowego algorytmu turbo-matching, czyli "polskiego" tricku na olimpiadę informatyczną na znajdowanie ścieżek powiększających alternujących tak, aby w iteracji próby powiększenia nie odwiedzać wierzchołka więcej niż raz. To heurystyczny algorytm, został spisany przez Jakuba Bachurskiego w

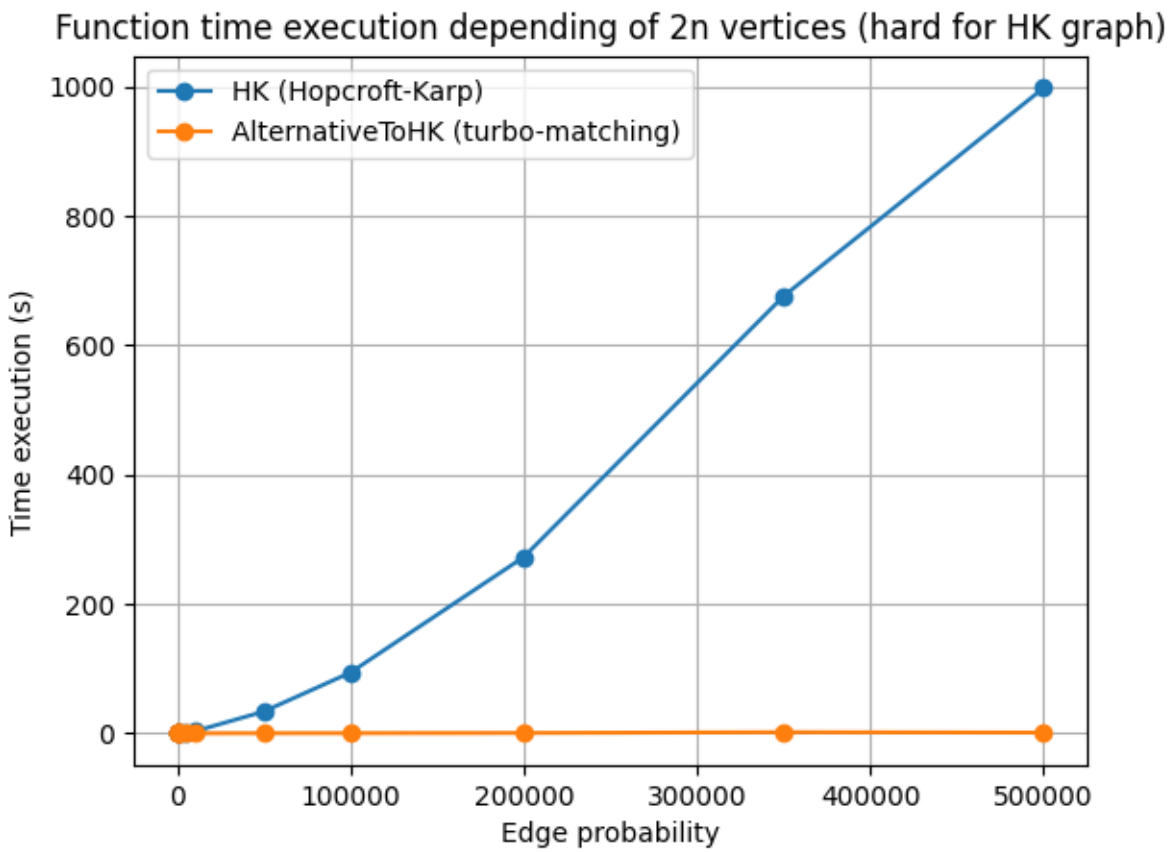
"*https* : *//kubin.w.staszic.waw.pl/static/pages/esoterica/matching/matching.pdf*".

e - wykresy


Function time execution depending of 2n vertices (hard for HK graph)
HK (Hopcroft-Karp)
PLcbc (CBC)
PLgur (Gurobi)
Time execution (s)
0
5
10
15
20
0
2000
4000
6000
8000
10000
Edge probability


Function time execution depending of 2n vertices (hard for HK graph)
HK (Hopcroft-Karp)
PLgur (Gurobi)
Time execution (s)
0
100
200
300
400
500
600
0
10000
20000
30000
40000
50000
Edge probability


Function time execution depending of edge probability
HK (Hopcroft-Karp)
AlternativeToHK (turbo-matching)
Time execution (s)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.0
0.2
0.4
0.6
0.8
1.0
Edge probability



g - wnioski

Nie mam pomysłu na kolejny graf który byłby ciekawy. Zatem pozostanę przy grafie pesymistycznym dla HK oraz randomowym. Wnioski:

- Wykres dla "Big test for pessimistic (HK and Aternative)": Alternative (turbo-matching) to pozioma kreska na wykresie, podczas gdy czas HK rośnie aż do 1000s. Wiemy, że złożoność HK na tym przykładzie to $\theta(E\sqrt{V}) = \theta(V\sqrt{V})$. V rzędu 500 000, czyli ilość obliczeń jest rzędu 500 000 000. Stąd aż tyle czasu. Na 1 sekundę przypada około 500 000 obliczeń na moim komputerze. Czas AlternativeToHK wynosi co najmniej omega(V), z wykresu wydaje się, że może maks 10 sekund trwają obliczenia, czyli obstawiam że efektywny czas to O(ElogV). Ale uwaga: prosta analiza pokazuje, że w tym przykładzie przejdzie po prostu $\sqrt{V}$ razy po kolejnych ścieżkach, tylko 1 iteracja będzie, więc ten przykład niewiele mówi: złożoność to $\theta(E)$.

- Wykres dla "Simple test for small random graphs": To szczególny graf i bardzo łatwo znaleźć krawędź gdy jest super gęsty(w szczególności dla p = 1 będzie tylko 1 iteracja, od razu zmatchuje HK każdą krawędź, stąd spadek w czasie działania niż dla p = 0.7). Zatem porównajmy czas dla p = 0.7. Dla HK to 0.4s, dla AlternativeToHK: 0.05s. W praktyce 8 razy lepszy. W typowym przypadku wydaje się więc, że złoność jest bliska liniowej dla turbo-matching. W praktyce O(ElogV) wydaje się być dobrym ograniczeniem. Oczywiście teoretycznie możliwe, że istnieje kontrprzykład i złożoność wyniesie $\theta(EV)$ (bo to heurystyczny algorytm), ale bardzo ciężko wymyślić. Tak więc turbo-matching działa.

- Porównajmy czas działania CBC, GUROBI i HK. Jak widać zawsze solvery PL są wolniejsze. Zawsze czas HK $<$ czas PLgur $<$ czas PLcbc. Wykresy porównujące te 3 algorytmy na raz pokazują, że w instancji trudnej dla HK - GUROBI działa podobnie do CBC, ale dla losowego grafu GUROBI jest wyraźnie szybszy(55s CBC - 4s GUROBI). Te wykresy nie pozwalają porównać jak bardzo szybki jest HK - to zbyt duża różnica.

- Porównajmy czas działania GUROBI i HK. Na losowym grafie to HK jest dużo szybsze na losowym grafie: 0.4s do 0.01s, czyli 40 razy szybszy. Warto zwrócić uwagę, że tu p = 1, czyli HK ma jedną fazę i czas $\theta(E)$. Dla przypadku trudnego dla HK, czas GUROBI - 600s, do HK - 50s, czyli 12 razy szybszy. Jest to dla n = 50000, czyli $\sqrt{(V)}$ to około 200. Zatem GUROBI wcale nie działa tak źle: tu rzędu $O(V^{1/4})$ wolniejszy. No więc obstawiam, że GUROBI to około $O(E * V^{3/4})$. Ma to sens: 40 razy wolniejszy jest GUROBI dla p = 1, czyli V = 300, $V^{3/4)} > 40$.
- Ostatecznie: CBC jest dużo wolniejszy od GUROBI, GUROBI to tak powiedzmy $O(E * V^{3/4})$, HK to $O(E \cdot \sqrt{V})$, AlternativeToHK to około O(ElogV).