Sprawozdanie

Algorytmy optymalizacji Dyskretnej
Laboratorium - Lista 4

Paweł Stanik

18. January 2025

0.1 Przedmowa

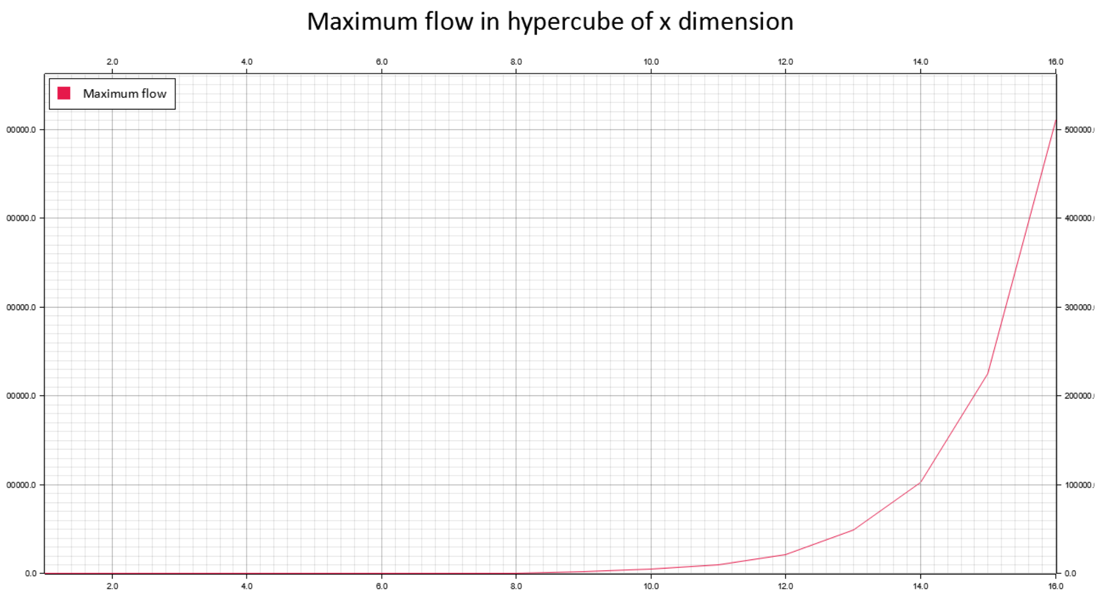
Zadanie polega na implementacji i przetestowaniu kilku algorytmów wyznaczania największych przepływów w grafach. Algorytmy zostały zaimplementowane w języku rust, wykresy wykonano za pomocą biblioteki plotters.

1 Algorytm Edmondsa - Karpa

Moja implementacja wykorzystuje BFS z kolejką oraz tablicę `PREV` oznaczającą poprzednika danego wierzchołka w BFS pozwalającą na znalezienie ścieżki do wierzchołka końca, a następnie za pomocą tej ścieżki modyfikujemy sieć resydualną. Bfs jest wykonywany w sieci resydualniej wielokrotnie, do póki istnieje ścieżka z wierzchołka startu do wierzchołka końca.

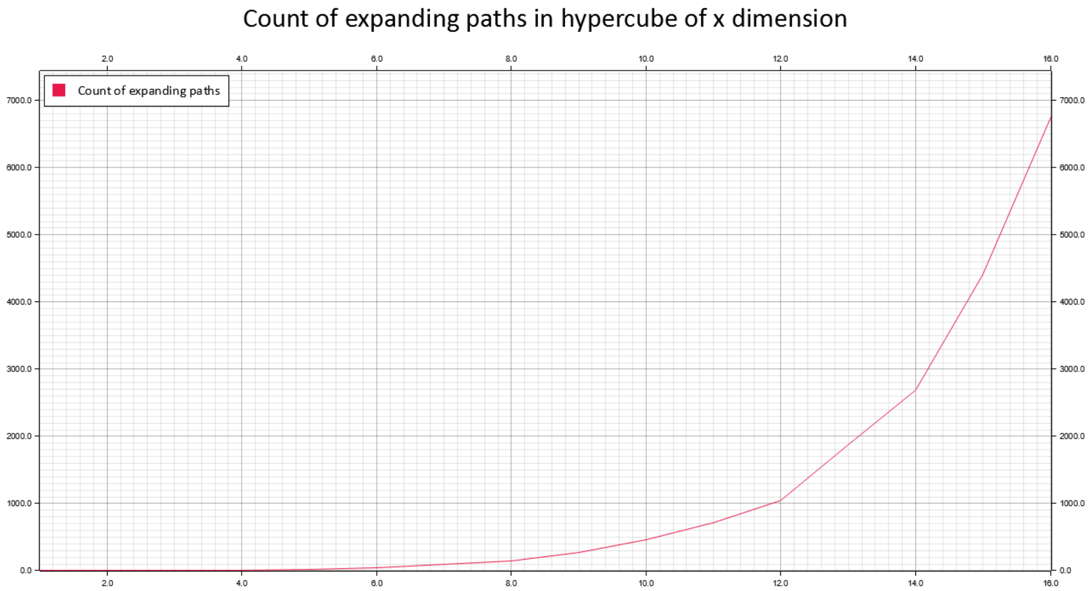
Złożoność algorytmu to $O(VE^2)$, znajdywanie ścieżki powiększającej ma złożoność $O(E)$, długość ścieżki nie może maleć, ale może się utrzymywać na tym samym poziomie przez co najwyżej $O(E)$ kroków, a maksymalna długość ścieżki jest $O(V)$.

1.1 Wykresy



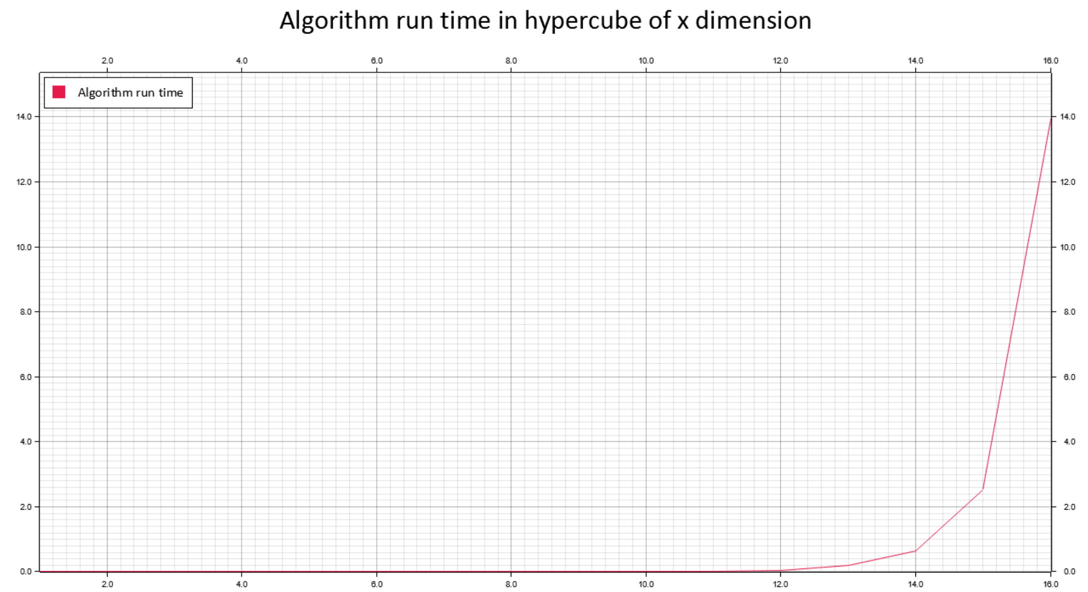
Rysunek 1: Wielkość maksymalnego przepływu

Wiemy że k wymiarowa hiperkostka ma $k \cdot 2^{k-1}$ krawędzi możemy więc przypuszczać że przepływ będzie w jakiś sposób zależny od liczby krawędzi i rzeczywiście obserwujemy eksponencjalny wzrost maksymalnego przepływu wraz ze wzrostem k.



Rysunek 2: Ilość wyznaczonych ścieżek powiększających

Wiemy że ilość ścieżek powiększających jest rzędu $O(V \cdot E)$ spodziewamy się więc również wzrostu eksponencjalnego i zaobserwowany wykres znowu na taki wygląda.



Rysunek 3: Czas działania algorytmu

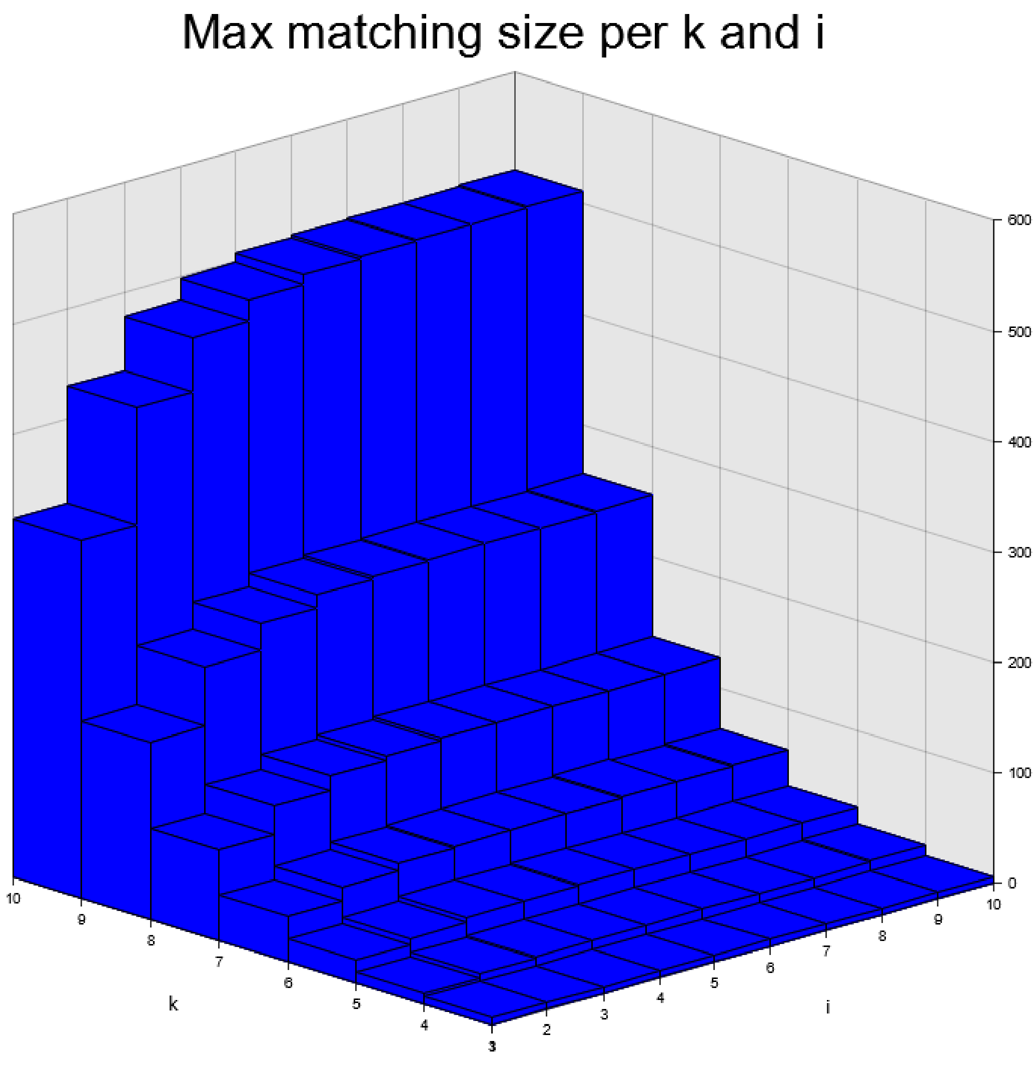
Jako że znamy złożoność czasową algorytmu $O(VE^2)$, wiemy że możemy się spodziewać wzrostu czasu na poziome 2^{2k} co zgadza się z obserwacjami.

2 Algorytm Wyznaczania Skojarzeń

Moja implementacja wykorzystuje zmodyfikowany algorytm BFS z kolejką oraz tablicą `PREV` do wyznaczania ścieżek powiększających, oraz tablicę `MATCHING` do zapisywania skojarzeń wierzchołków, modyfikacja polega na „zmuszeniu” algorytmu po przejściu po krawędzi skojarzenia w co drugiej iteracji.

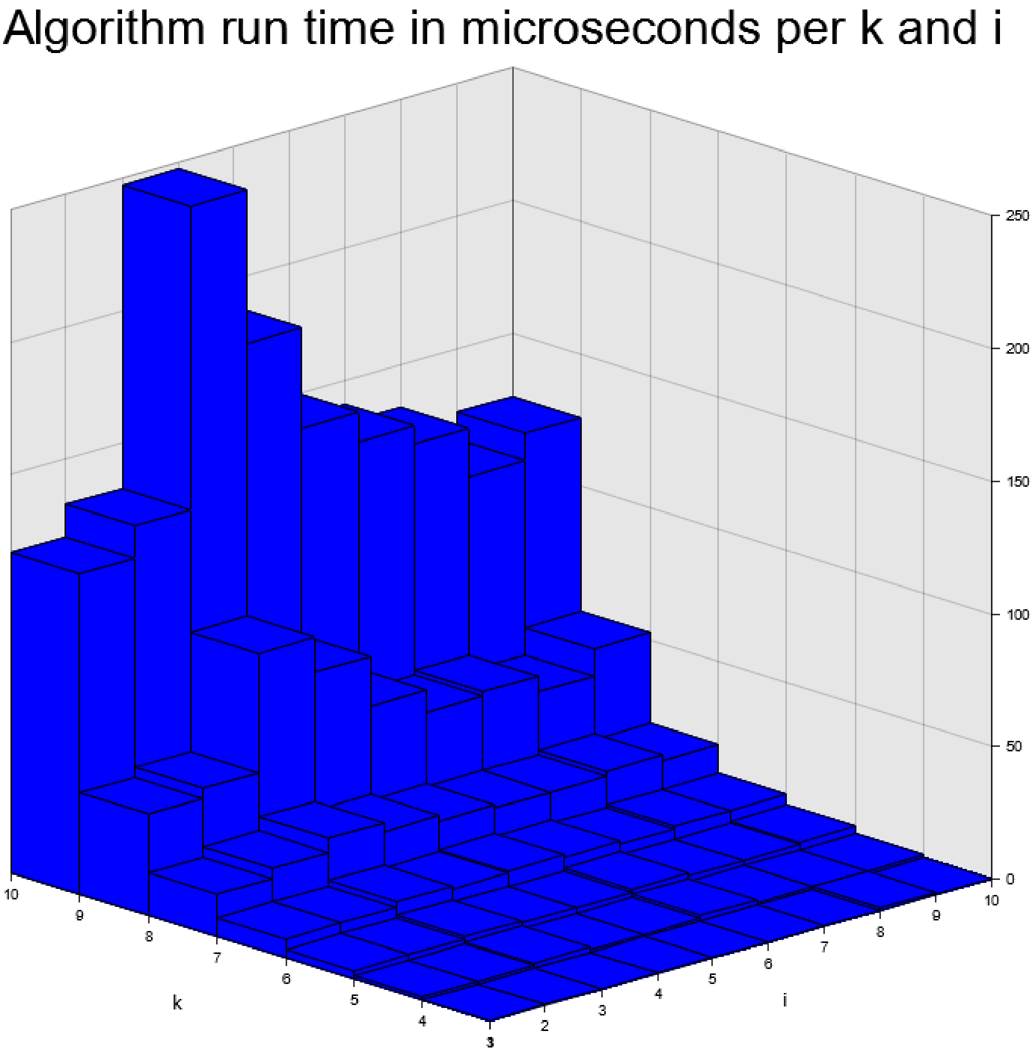
Złożoność algorytmu to $O\left(E\sqrt{V}\right)$, BFS posiada złożoność $O(E)$ i jest wykonywany $O\left(\sqrt{V}\right)$ razy.

2.1 Wykresy



Rysunek 4: Wielkość maksymalnego matchingu w stosunku do i oraz k

Obserwujemy eksponencjalny wzrost rozmiaru matchingu wraz ze wzrostem k, co jest spowodowane eksponencjalnym wzrostem ilości wierzchołków, obserwujemy również logarytmiczny wzrost wielkości matchingu wraz ze wzrostem i. Co ciekawe już dla $i = 4$ matching obejmuje niemal 100% wierzchołków. Wydaje mi się że jest to w jakiś sposób powiązane z rozkładem geometrycznym zmiennej losowej.



Rysunek 5: Czas działania algorytmu w stosunku do i oraz k

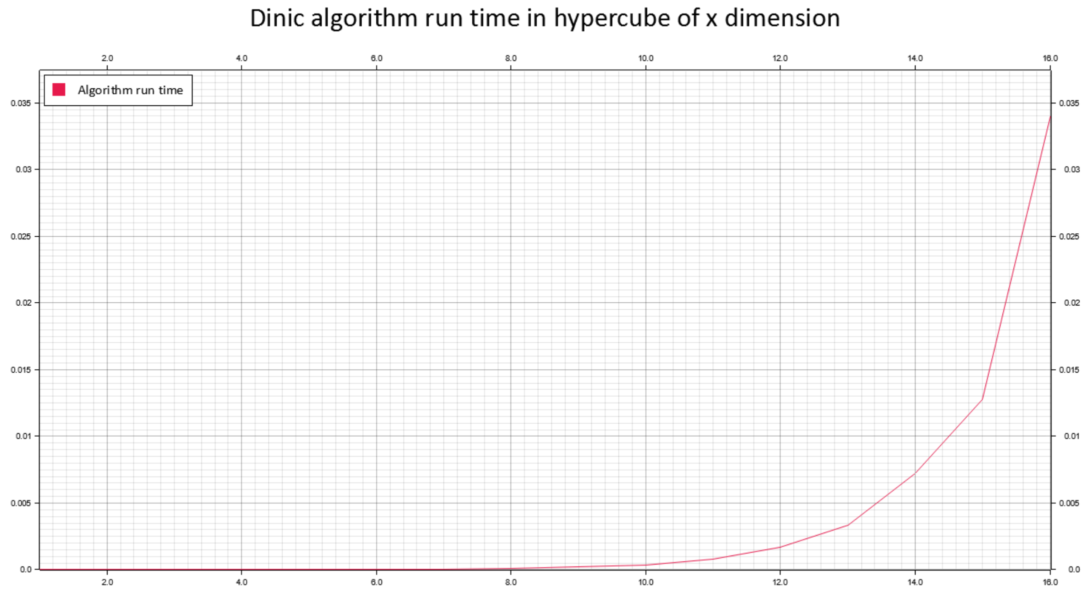
Obserwujemy eksponencjalny wzrost czasu działania wraz ze wzrostem k, co ciekawe wykres ze wzrostem i zachowuje się specyficznie osiągając maksimum dla $i = 3$, jest to spowodowane faktem że dla małych i statystycznie rzadkie są długie ścieżki powiększające w sieci, z kolei dla dużych i istnieje duże prawdopodobieństwo znalezienia krótkiej ścieżki powiększającej, co powoduje że dla wartości pomiędzy czas działania jest największy.

3 Algorytm Dinitza

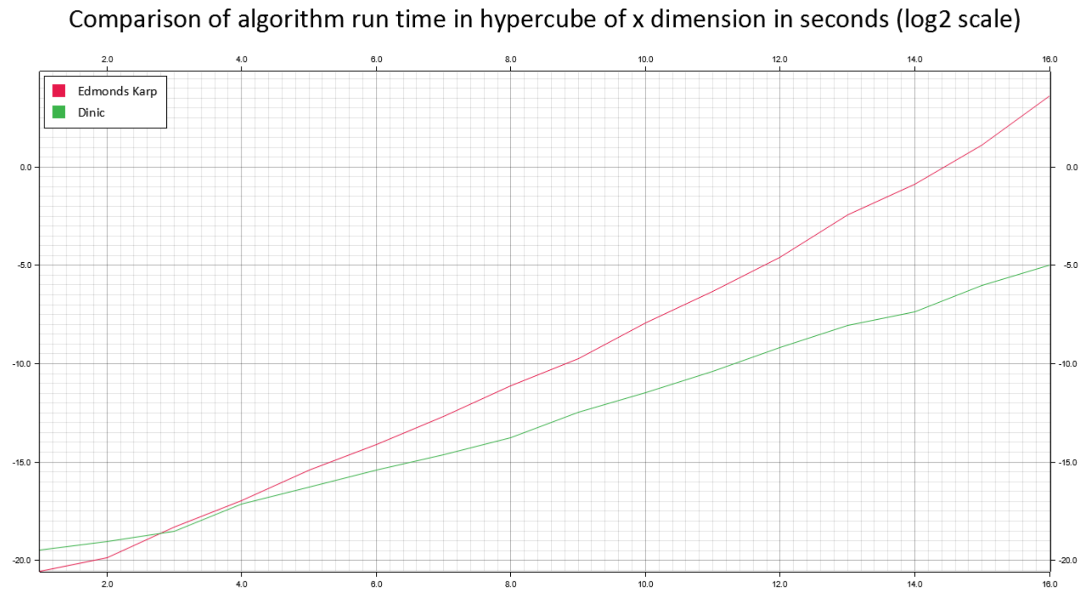
Algorytm Dinica to modyfikacja algorytmu znajdowania maksymalnego przepływu w sieci. Wykorzystuje on graf poziomowy aby zredukować poruszanie się po krawędziach które nie doprowadzą do źródła.

Moja implementacja wykorzystuje algorytm BFS z kolejką i tablicą `LEVEl` do wyznaczania grafu poziomowego i algorytm DFS z tablicą `PATH` do wyznaczania ścieżek powiększających w grafie poziomowym.

Algorytm działa w czasie $O(V^2E)$, wyznaczanie grafu poziomowego zajmuje $O(E)$ a wyznaczenie przepływu blokującego w grafie poziomowym zajmuje $O(VE)$.



Rysunek 6: Czas działania algorytmu Dinitza



Rysunek 7:

Jak widzimy algorytm Dinitza jest magnitudy szybszy dla większych grafów, dla małych grafów jest jednak nieco wolniejszy. Obliczenia wykazują że algorytm Dinica jest $\approx 2^{0.5x-2}$ razy szybszy. Widzimy że jest on znacznym usprawnieniem algorytmu Edmondsa - Karpa

4 Modele programowania liniowego

4.1 Max Flow

1. Konwersje rozpoczynamy od zamiany grafu na postać macierzy sąsiedztwa
2. Ustalamy zmienne decyzyjne jako macierz o rozmiarach macierzy sąsiedztwa gdzie każdy element jest większy od zera
3. Ustalamy ograniczenie: wartość zmiennej decyzyjnej nie może być większa od odpowiadającej jej wartości w macierzy sąsiedztwa
4. Ustalamy ograniczenie: ilość przepływu wpływająca do wierzchołka musi być taka sama jak ilość wypływająca tzn. suma wartości i-tej kolumny musi być równa sumie i-tego wiersza (z pominięciem pierwszego i ostatniego - start i sink)
5. Ustalamy funkcję celu jako ilość przepływu wypływającą z 1 wierzchołka (suma 1 wiersza)

4.2 Max Flow

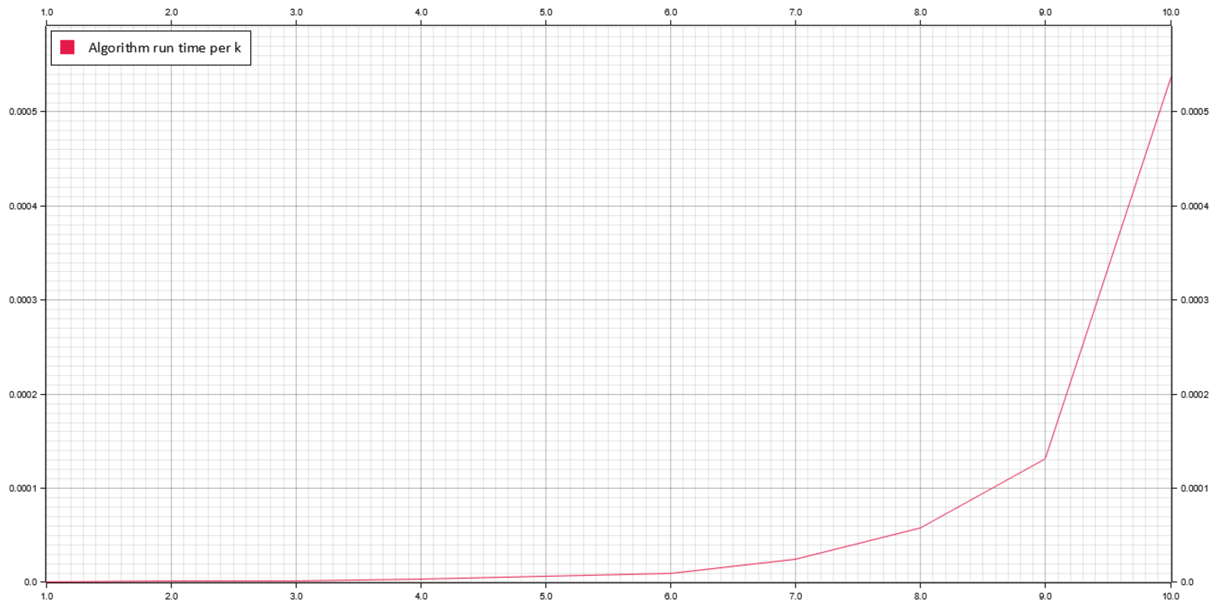
1. Konwersje rozpoczynamy od zamiany grafu na postać macierzy sąsiedztwa
2. Ustalamy zmienne decyzyjne jako macierz o rozmiarach macierzy sąsiedztwa gdzie każdy element jest równy 1 lub 0
3. Ustalamy ograniczenie: wartość zmiennej decyzyjnej nie może być większa od odpowiadającej jej wartości w macierzy sąsiedztwa
4. Ustalamy ograniczenie: jeden wierzchołek może być połączony co najwyżej z jednym wierzchołkiem, tzn. suma danego wiersza ≤ 1 i suma danej kolumny ≤ 1
5. Ustalamy funkcję celu jako ilość przepływu wypływającą z 1 wierzchołka (suma 1 wiersza)

4.3 Eksperymenty

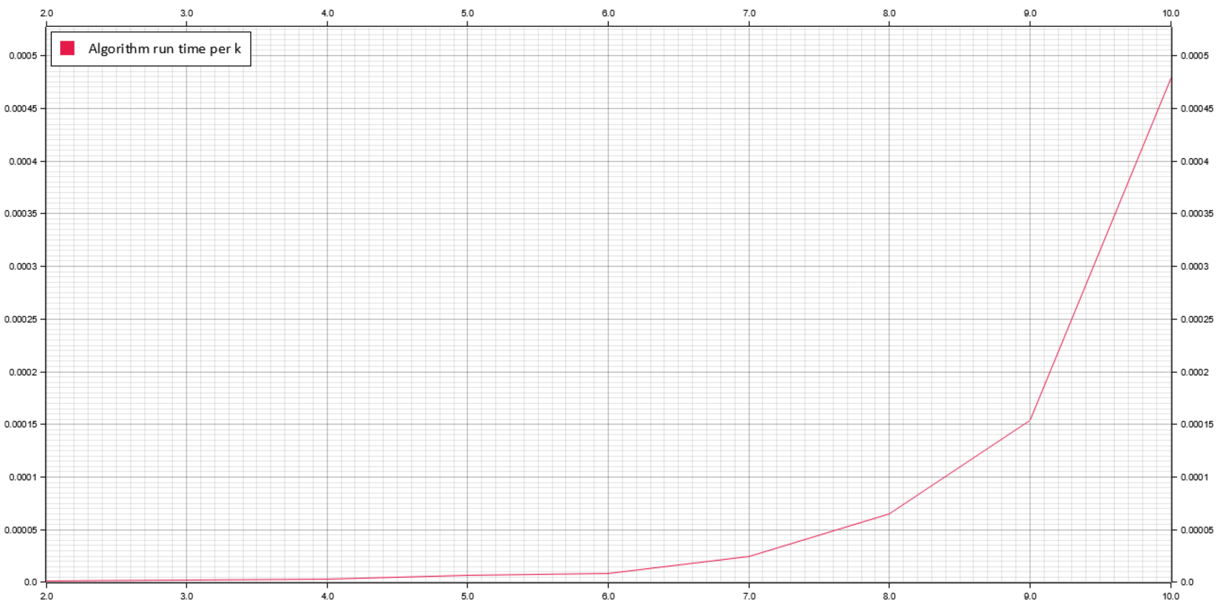
Testy wykazały że solver zwraca te same obliczane wartości co algorytmy, i wyznacza te same rozwiązanie, działa przy tym znacznie wolniej od algorytmów (możliwe że jest to spowodowane po części parsowaniem wygenerowanej macierzy)

5 Dodatkowe wykresy

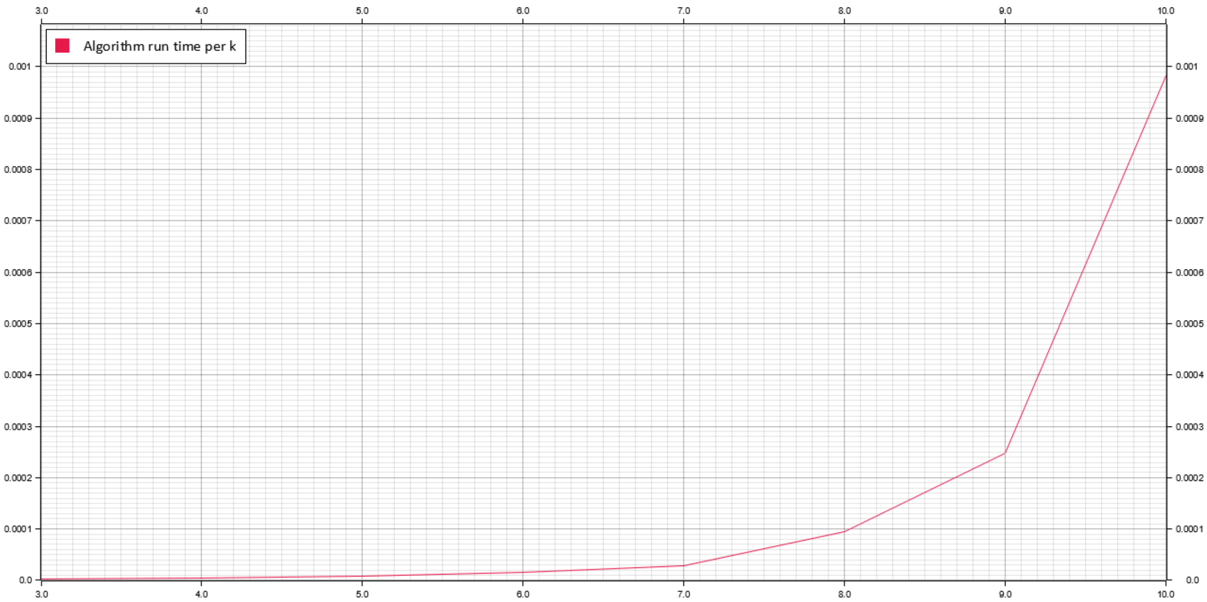
Algorithm run time in in bipartite graph of i = 1



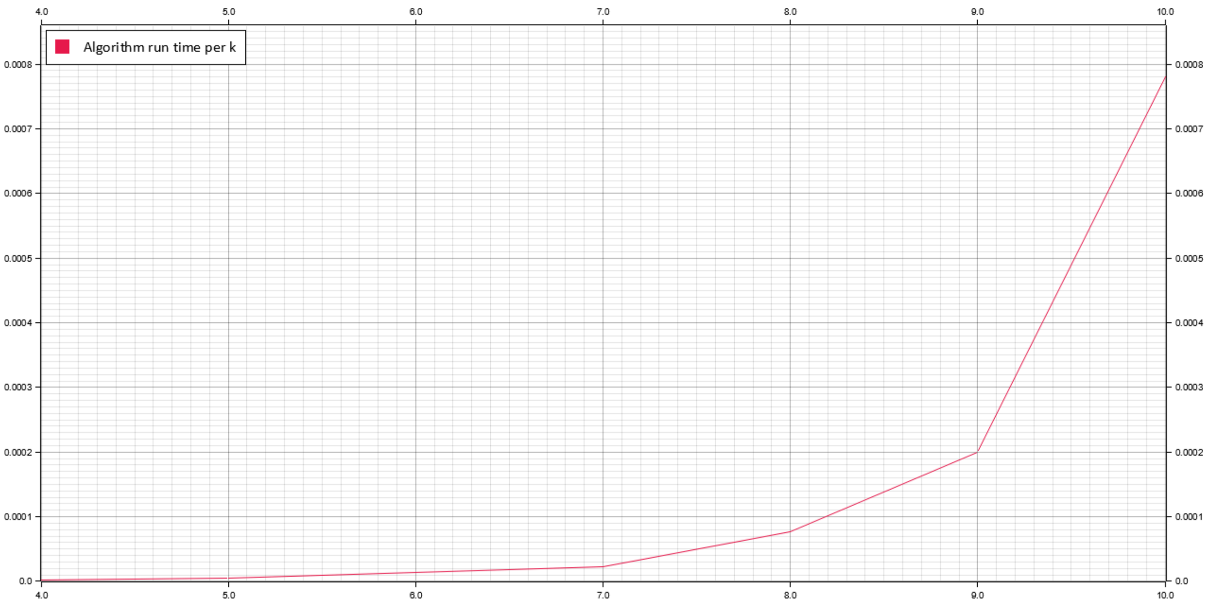
Algorithm run time in in bipartite graph of i = 2



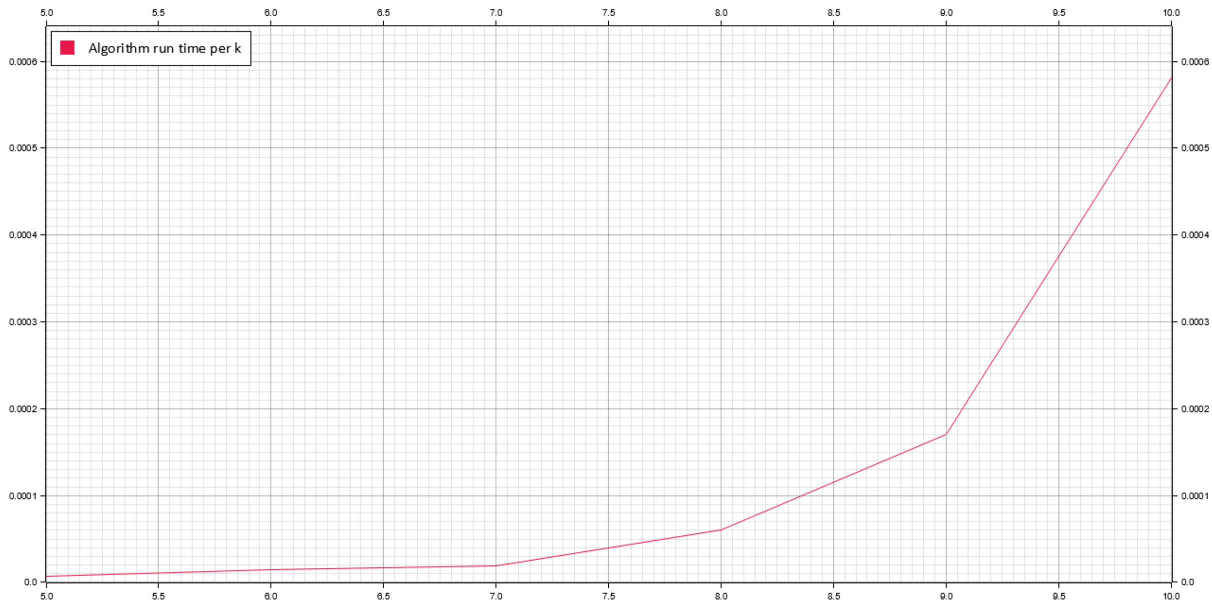
Algorithm run time in in bipartite graph of i = 3



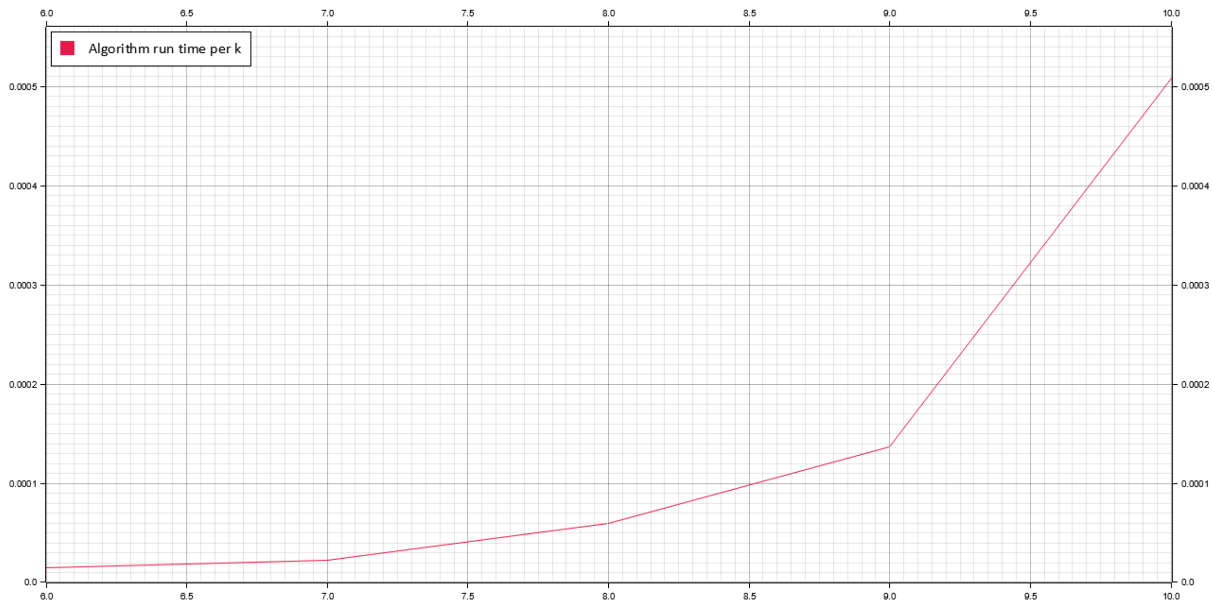
Algorithm run time in in bipartite graph of i = 4



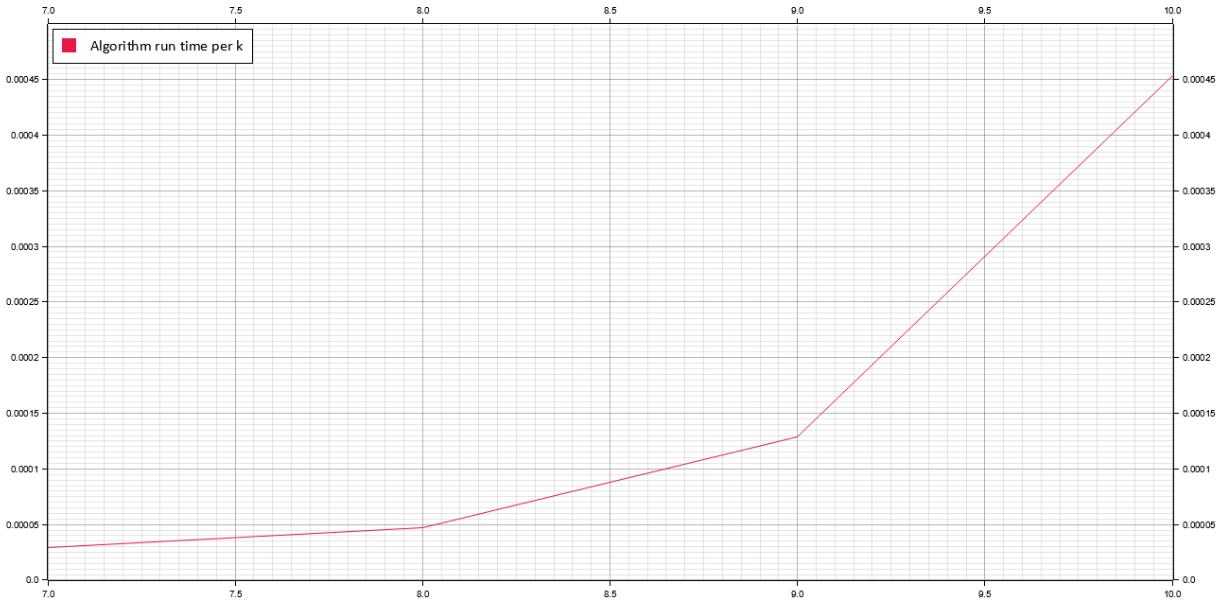
Algorithm run time in in bipartite graph of i = 5



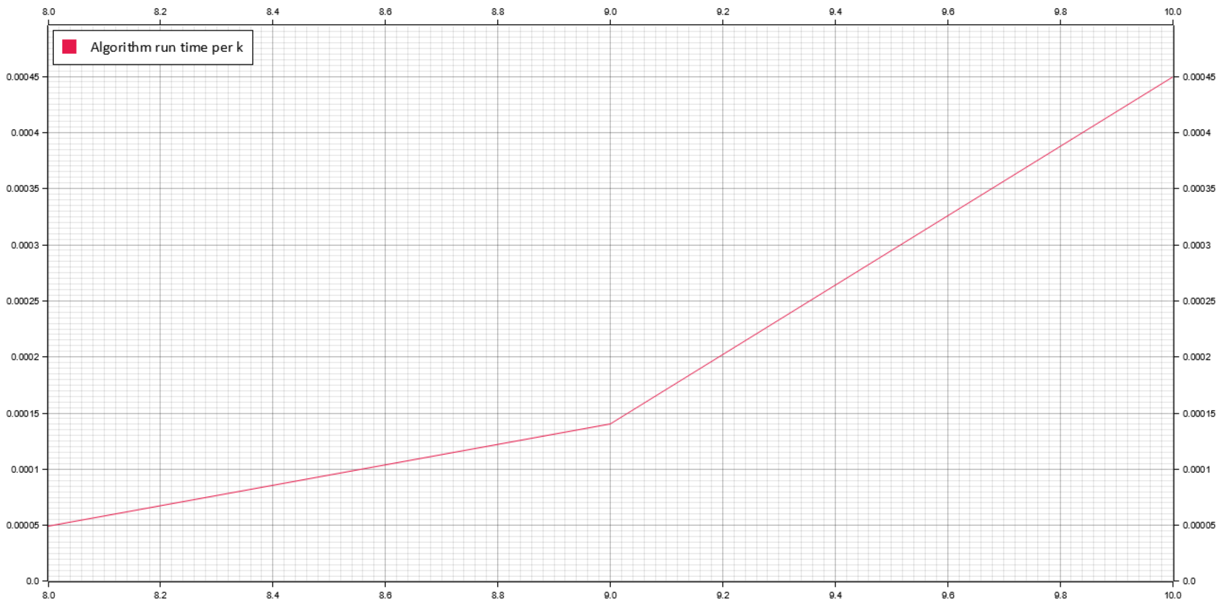
Algorithm run time in in bipartite graph of i = 6



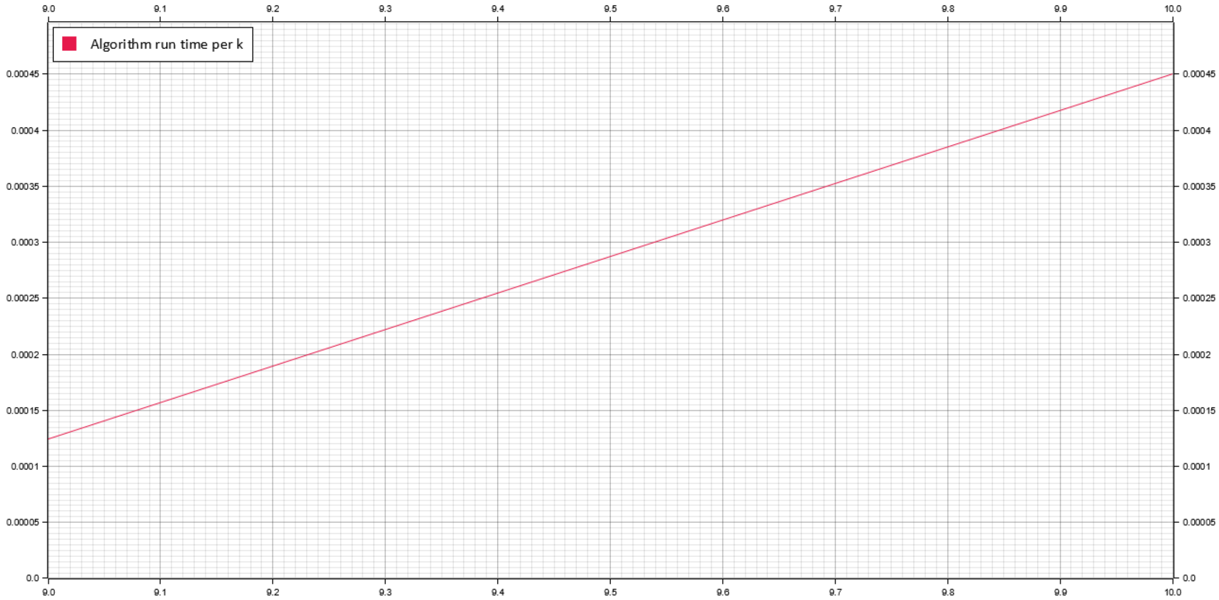
Algorithm run time in in bipartite graph of i = 7



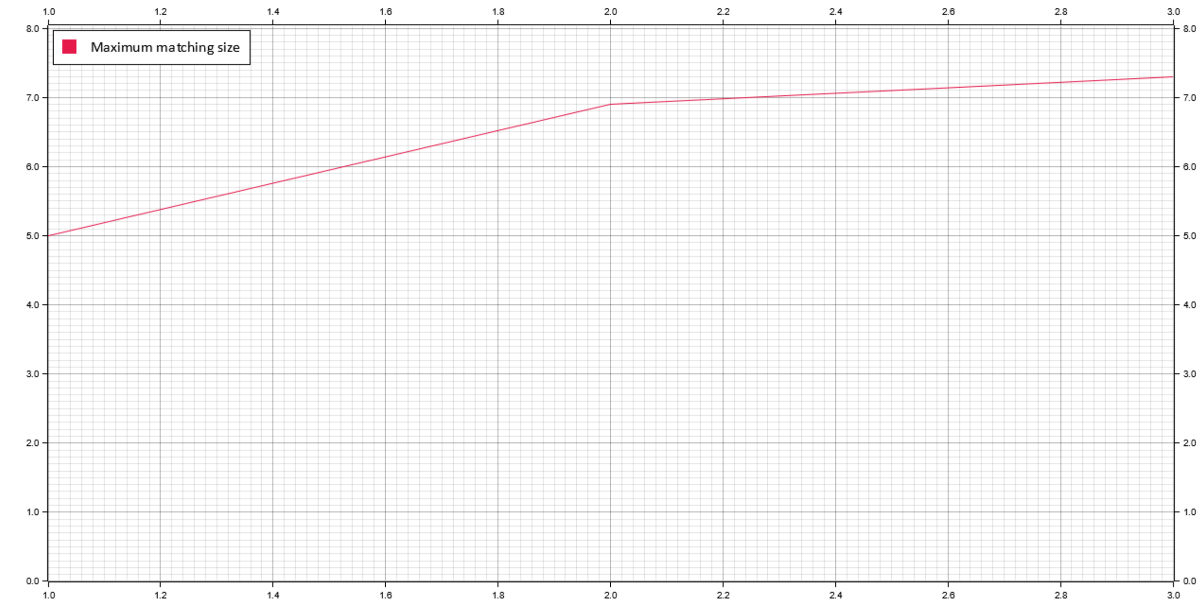
Algorithm run time in in bipartite graph of i = 8



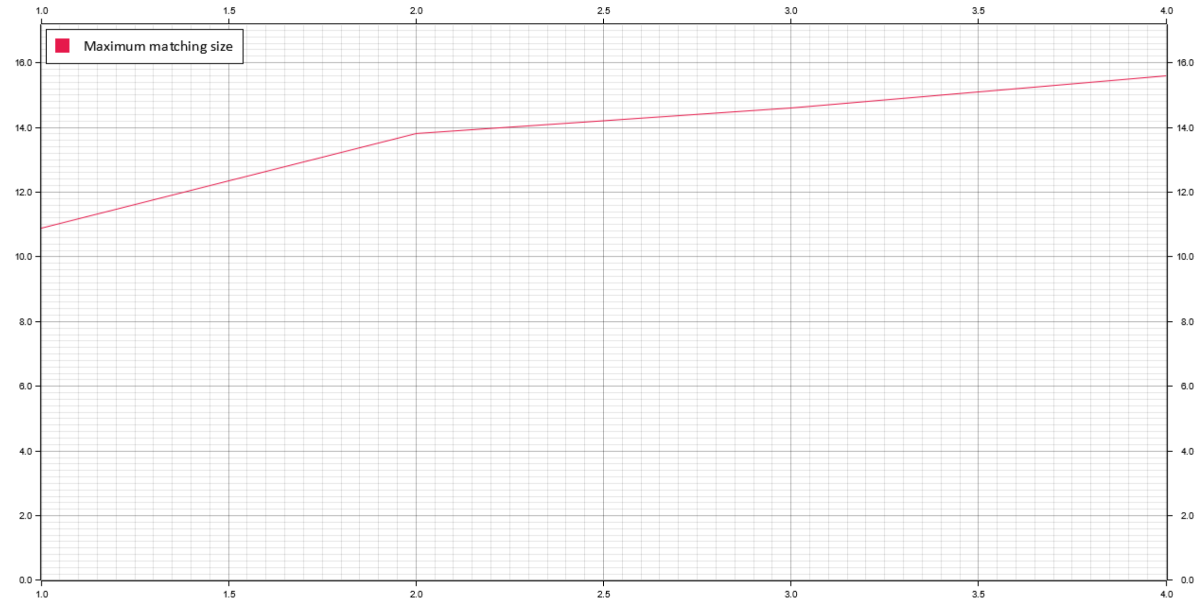
Algorithm run time in in bipartite graph of i = 9



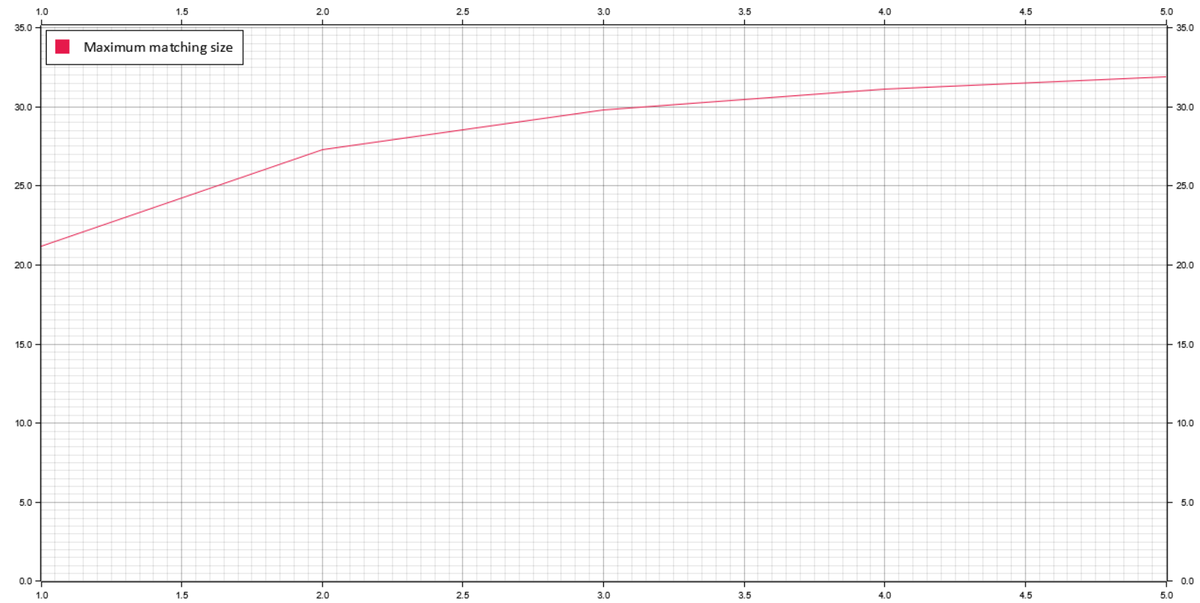
Maximum matching size in bipartite graph of k = 3



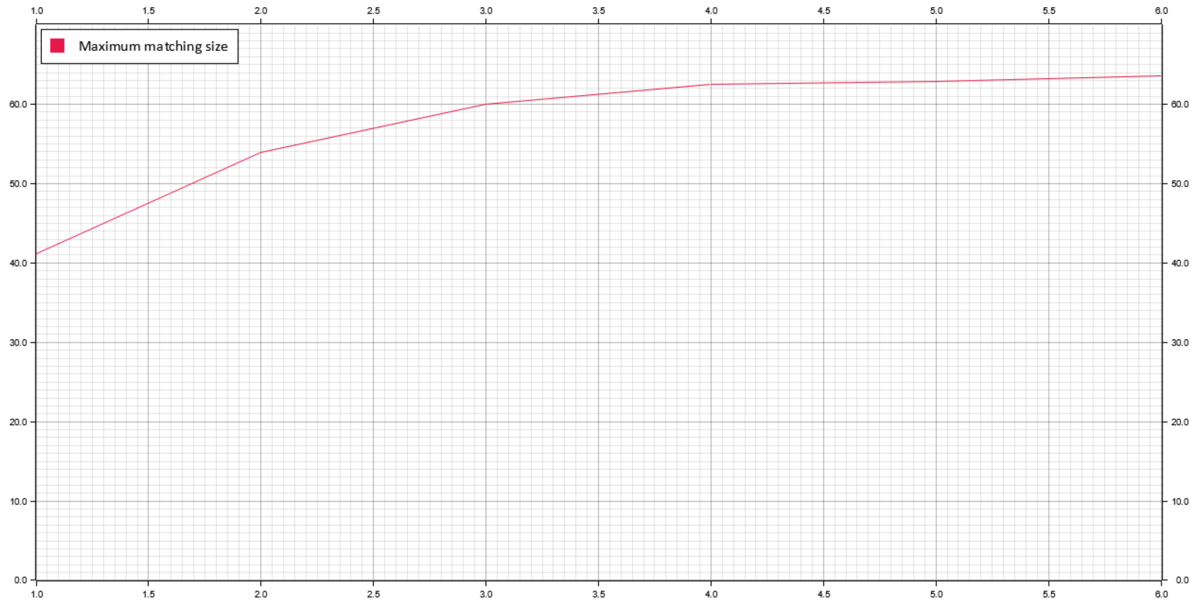
Maximum matching size in bipartite graph of k = 4



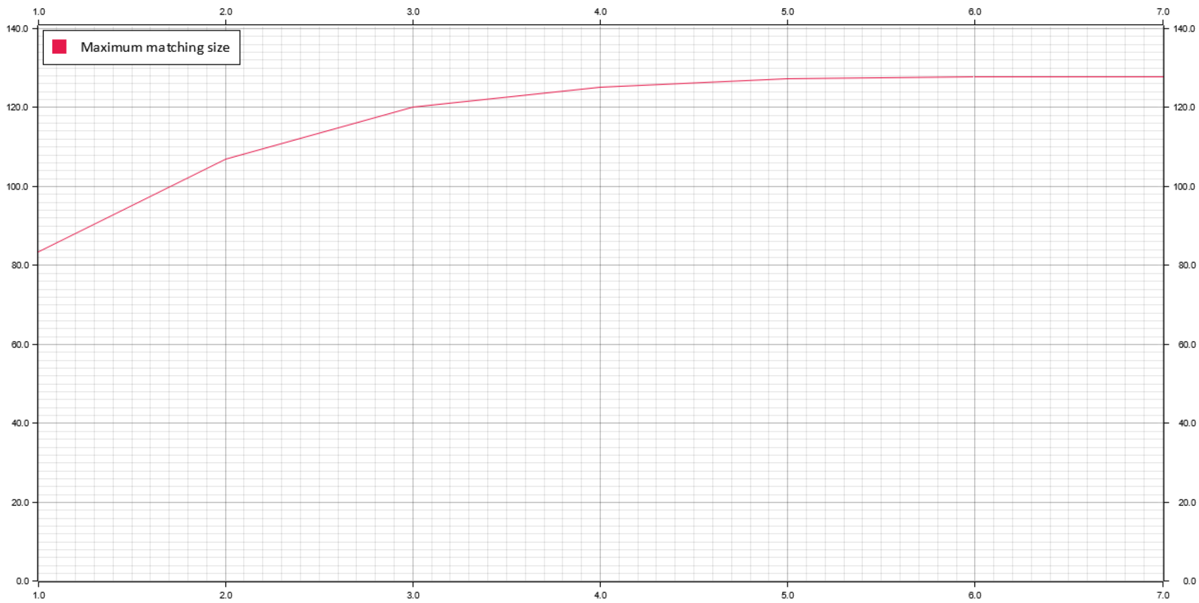
Maximum matching size in bipartite graph of k = 5



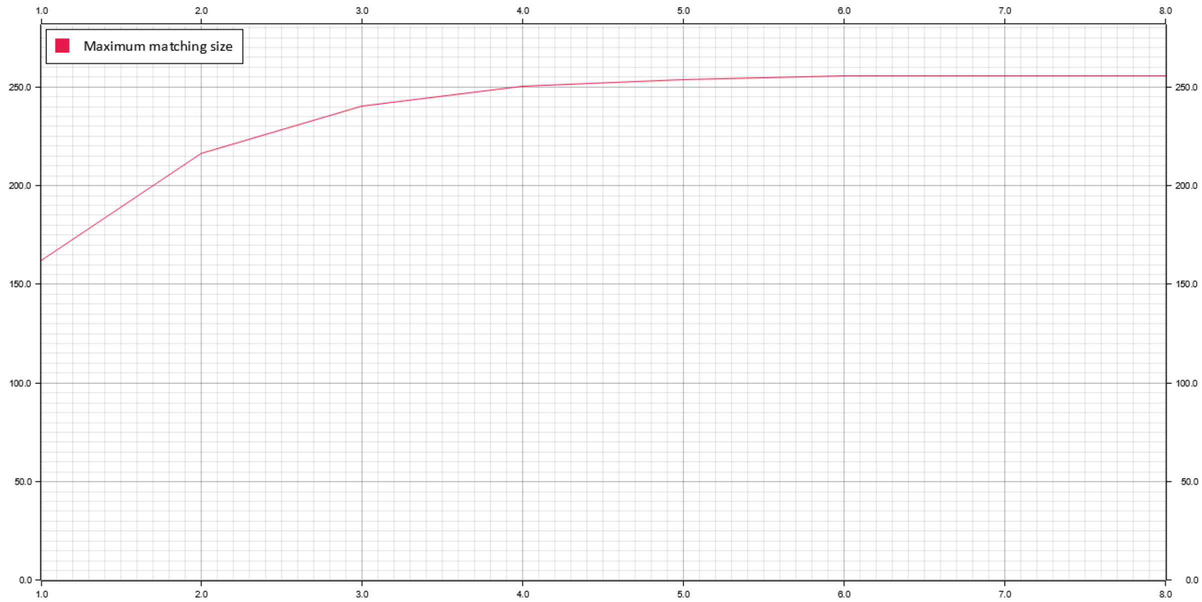
Maximum matching size in bipartite graph of k = 6



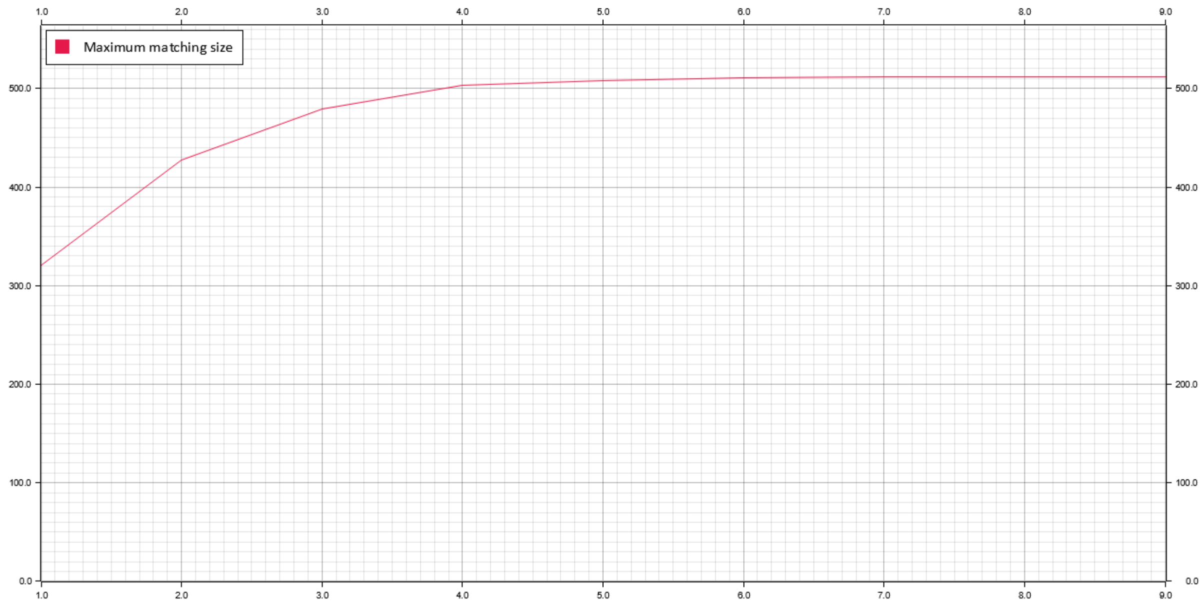
Maximum matching size in bipartite graph of k = 7

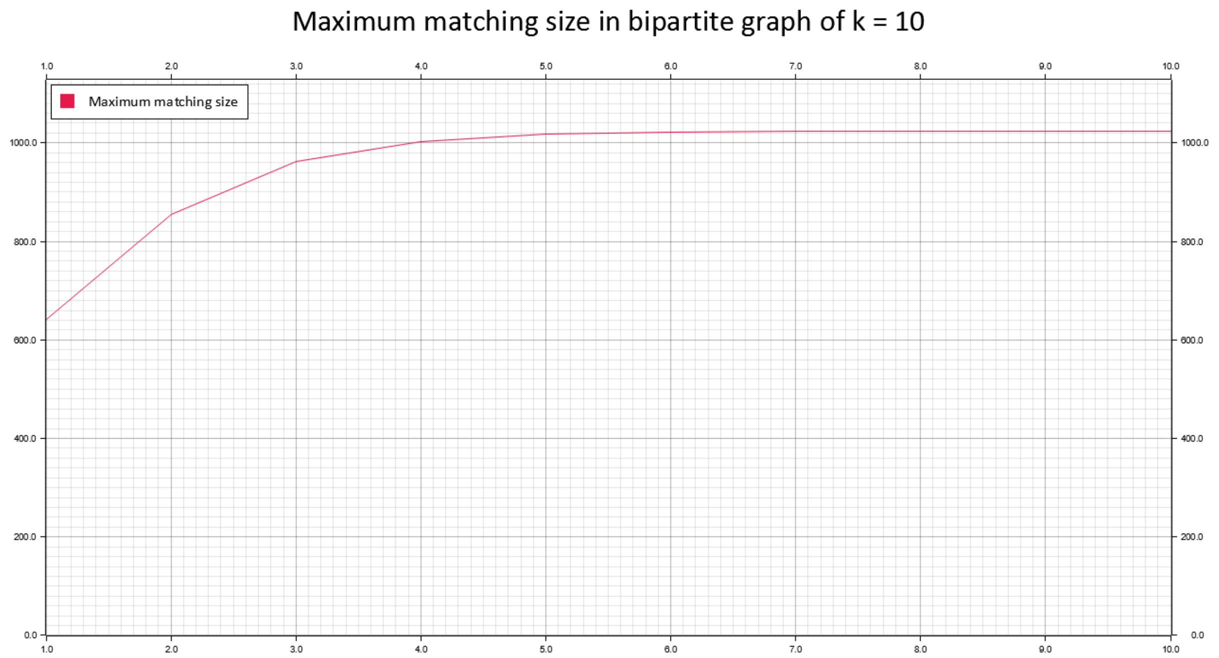


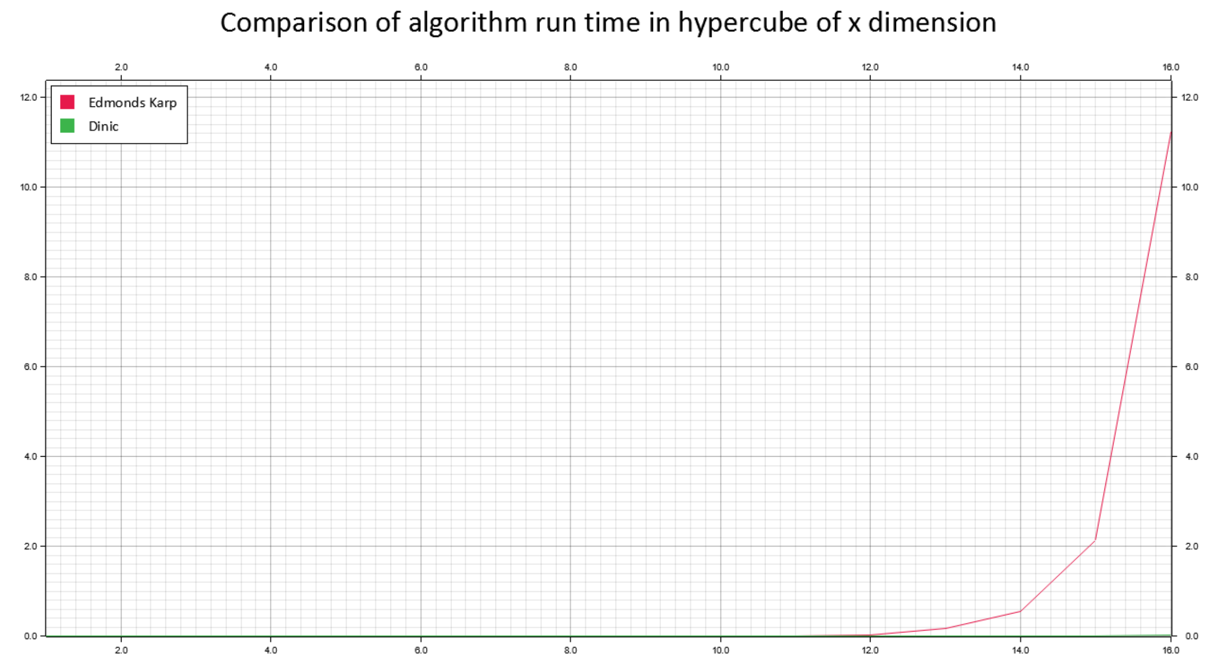
Maximum matching size in bipartite graph of k = 8



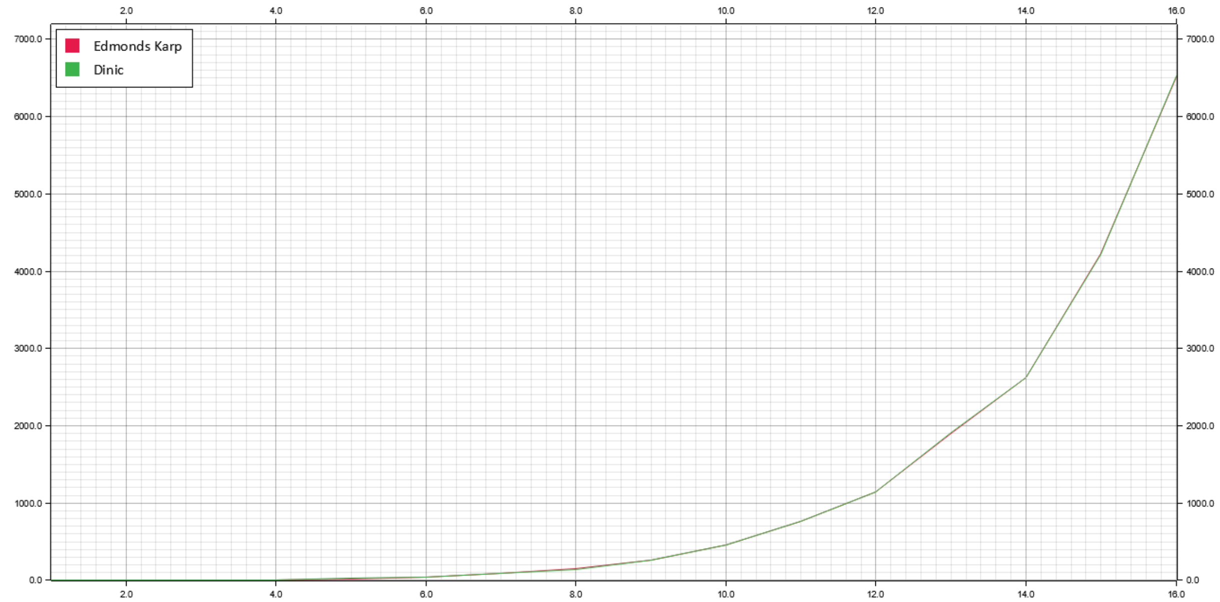
Maximum matching size in bipartite graph of k = 9




Maximum matching size in bipartite graph of k = 10
Maximum matching size


Comparison of algorithm run time in hypercube of x dimension
Edmonds Karp
Dinic

Comparison of count of expanding paths in hypercube of x dimension



Comparison of maximum flow in hypercube of x dimension