



Temat 5

Sieci samoorganizujące się

Jakub Radzik 260366
Kamil Herbertko 260389
Tomasz Płóciennik 260404

Specjalistyczne technologie
w sieciach informatycznych
nowej generacji

dr inż. Maciej Hojda

Opis słowny problemu



Problem opisany w artykule:

“Problem alokacji przepustowości łącz w sieci komputerowej”

Mamy sieć komputerową, przez którą użytkownicy wysyłają wiele transmisji danych. Węzły sieci (np routery) dzielą swoją przepustowość pomiędzy różne strumienie danych. Zauważamy że użytkownicy działają egoistycznie i maksymalizują własną korzyść, co prowadzi do nierównomiernego obciążenia sieci i wzrostu opóźnień. Problem polega na znalezieniu sposobu na podzielenie tej przepustowości w sposób efektywny dla całej sieci i sprawiedliwie dla wszystkich użytkowników.

Wyjaśnienie znaczenia problemu



Optymalizacja alokacji zasobów jest kluczowe dla współczesnych sieci.

Efektywne zarządzanie zasobami pozwala nam zyskać:

- **poprawę jakości usługi (QoS)** - minimalizacja problemów z przeciążeniem sieci, opóźnieniem; zwiększenie zadowolenia użytkowników końcowych
- **autonomiczną sieć** - sieć zautomatyzowana ma zarządzać sama sobą, a rola człowieka ma być sprowadzona do definiowania sieci a nie sterowania nią,
- **redukcję kosztów operacyjnych** - można lepiej wykorzystać istniejące zasoby sprzętowe zamiast inwestować w droższy sprzęt.

Przykładowe zastosowania



- **Sieci operatorskie** - sieci utrzymywane przez dostawców usług internetowych (ISP), którzy muszą zapewniać odpowiednią jakość usług dla swoich klientów.
- **Centra danych** - gdzie wiele serwerów musi współdzielić ograniczone zasoby sieciowe
- **Sieci mobilne** - utrzymanie stabilnej przepustowości i niskich opóźnień dla użytkowników mobilnych.

Decydenci



Decydenci w problemie optymalnej alokacji przepustowości w sieciach to osoby i instytucje odpowiedzialne za podejmowanie decyzji dotyczących zarządzania zasobami sieciowymi:

- dostawcy usług internetowych ISP
- operatorzy centrów danych
- administratorzy sieci
- producenci sprzętu

Beneficjenci



Beneficjenci efektywnego rozwiązania:

- użytkownicy końcowi
- twórcy aplikacji
- klienci usług chmurowych

Słownik pojęć



Przepustowość – Maksymalna liczba danych, które mogą być przesyłane przez łącze w określonym czasie, wyrażana zazwyczaj w bitach na sekundę (bps), stanowiąca kluczowy parametr wydajności łącza.

Alokacja zasobów – Proces przydzielania ograniczonych zasobów, takich jak przepustowość łącza, między różne aplikacje lub przepływy danych w celu zapewnienia optymalnej wydajności.

Słownik pojęć c.d.



Jakość Usług (Quality of service, QoS) – Zbiór mechanizmów stosowanych w sieci, które mają na celu na przykład minimalizację opóźnień, strat pakietów, aby utrzymać pożądaną jakość transmisji danych.

Maksymalizacja użyteczności sieci (Network Utility Maximization, NUM) – Problem optymalizacji podziału zasobów sieciowych, w którym dąży się do maksymalizacji użyteczności dla użytkowników, uwzględniając ograniczenia sieciowe i ich preferencje.

Słownik pojęć c.d.



Gra alokacji zasobów – Model matematyczny opisujący proces podejmowania decyzji przez uczestników (np. dostawców usług sieciowych) o tym, jak przydzielać zasoby, aby maksymalizować własne korzyści, jednocześnie utrzymując równowagę w sieci.

Równowaga Nasha (Nash Equilibrium) – Sytuacja w grze, w której żaden z graczy nie ma motywacji do jednostronnej zmiany swojej strategii, gdyż każda zmiana prowadziłaby do pogorszenia jego wyniku, zachowując stabilność rozwiązań.

Słownik pojęć c.d.



Dobrobyt społeczny - suma wypłat wszystkich graczy w danym profilu strategii.

Sieci autonomiczne – Systemy sieciowe, które są zdolne do samodzielnego zarządzania swoimi zasobami bez potrzeby ingerencji człowieka, oparte na algorytmach optymalizacji zdecentralizowanej.

Słownik pojęć c.d.



Cena Anarchii – Wskaźnik, który opisuje stosunek między optymalnym społecznym dobrobytem a najgorszym wynikiem, jaki może zostać osiągnięty w równowadze Nasha w danym systemie.

Cena Stabilności – Wskaźnik, który opisuje stosunek między globalnie optymalnym rozwiązaniem a najlepszą możliwą równowagą Nasha, wskazując na potencjalne straty w porównaniu do najbardziej efektywnego scenariusza.



Temat 5

Sieci samoorganizujące się

Jakub Radzik 260366
Kamil Herbertko 260389
Tomasz Płóciennik 260404

Specjalistyczne technologie
w sieciach informatycznych
nowej generacji

dr inż. Maciej Hojda

Matematyczne sformułowanie problemu

- Rozważmy sieć składającą się z L połączeń z określoną pojemnością $c_l > 0$, gdzie $l \in \{1, \dots, L\}$. Oznaczamy $c = [c_1, \dots, c_L]^T$.
- W tej sieci realizowane jest R przepływów pakietów, które wykorzystują pewną ilość połączeń opisaną w macierzy $A = [a_{lr}]$, gdzie $a_{lr} = 1$ oznacza że r -ty przepływ przechodzi przez l -te połączenie, w przeciwnym wypadku 0.
- Każdy przepływ jest opisany przez prędkość transmisji $x_r \geq 0$. Oznaczamy $x = [x_1, \dots, x_R]^T$.
- Dla każdego przepływu powiązana jest miara użyteczności $u_r(x_r)$, która określa tzw. *willingness to pay*.

Zmienne decyzyjne



- Oznaczmy \mathbf{x} jako wektor prędkości transmisji dla przepływów $r \in \{1, \dots, R\}$:

$$\mathbf{x} = [x_1, \dots, x_R]^T$$

Ograniczenia



1. Suma prędkości wszystkich transmisji wykorzystujących dane łącze nie może być większa niż jego pojemność:

$$\mathbf{A} \times \mathbf{x} \leq \mathbf{c}$$

2. Szybkości transmisji muszą być nieujemne:

$$\mathbf{x} \geq 0$$

Funkcja Celu



- Celem jest **maksymalizacja użyteczności sieci**. Czyli szukamy maksymalnego $Q(\mathbf{x})$, gdzie:

$$Q(\mathbf{x}) = \sum_{r=1}^R u_r(x_r)$$

- Rozważane przez nas funkcje użyteczności u_r ograniczamy do podzbioru funkcji izoelastycznych:

$$u_r(x_r) = \begin{cases} w_r \frac{1}{1-\gamma} x_r^{1-\gamma} & \gamma > 0, \gamma \neq 1, \\ w_r \log x_r & \gamma = 1. \end{cases}$$

Problem teorii gier



Graczami w grze są łącza sieciowe $l \in \{1, \dots, L\}$.

Każde łącze zarządza swoją przepustowością c_l i podejmuje decyzje o jej podziale między przepływy x , które przez nie przechodzą.

Problem teorii gier c.d.



Funkcja wypłat $Q_l(S)$ dla gracza l jest obliczana jako wartość użyteczności wszystkich przepływów przechodzących przez jego łącze. Funkcja ta zapisywana jest jako:

$$Q_l(S) = \sum_{r=1}^R a_{lr} b_r u_r \left(\min_{k: a_{kr}=1} s_{kr} \right)$$

Gdzie $b_r \geq 0$ jest wagą przypisaną do r -tego przepływu

Problem teorii gier c.d.



Decyzja l -tego gracza jest nazywana strategią gracza i jest opisywane wektorem $s_l = [s_{l1}, s_{l2}, \dots, s_{lr}]^T$, gdzie s_{lr} jest fragmentem pojemności l -tego łącza przypisanym do r -tego przepływu.

Problem teorii gier c.d.



Ograniczamy wybory gracza tylko do wykonywalnych, czyli takich gdzie suma wszystkich przepustowości przydzielonych do łącza jest mniejsza lub równa pojemności łącza:

$$\sum_{r=1}^R a_{lr} s_{lr} \leq c_l$$

Dodatkowo przepustowość pojedynczego przepływu jest ograniczona przez minimalny przydział wzdłuż ścieżki tego przepływu (zdefiniowanej przez macierz routingu A).



Temat 5

Sieci samoorganizujące się

Jakub Radzik 260366
Kamil Herbertko 260389
Tomasz Płóciennik 260404

Specjalistyczne technologie
w sieciach informatycznych
nowej generacji

dr inż. Maciej Hojda

Wstęp

Problem z artykułu:

- alokacja przepustowości łączy między przepływy
- graczami są połączenia
- gracze rozdzielają swoją przepustowość między przepływy

```

1:  $\mathbf{S}^{(1)} \leftarrow \mathbf{S}^I$ 
2:  $\widehat{\mathcal{R}}_1 \leftarrow \{1, \dots, R\}$ 
3:  $\mathcal{L}_1 \leftarrow \left\{ l : \sum_{r=1}^R a_{lr} \left( \min_{k: a_{kr}=1} s_{kr}^{(1)} \right) = c_l \right\}$ 
4:  $n \leftarrow 2$ 
5: while  $n \leq L$  do
6:    $\widehat{\mathcal{R}}_n \leftarrow \widehat{\mathcal{R}}_{n-1} \setminus \mathcal{R}_{\phi(n-1)}$ ,
     where:

```

$\phi(i) := \min \mathcal{L}_i$,

$$\mathcal{L}_i = \left\{ l : \sum_{r=1}^R a_{lr} \left(\min_{k: a_{kr}=1} s_{kr}^{(i)} \right) = c_l \right\}$$

```

7:   if  $\widehat{\mathcal{R}}_n = \emptyset$  then
8:      $\mathbf{S}^{II} \leftarrow \mathbf{S}^{(n)}$ 
9:     return  $\mathbf{S}^{II}$ 
10:   end if
11:   determine  $\mathbf{S}^{(n)}$ :

```

$$\mathbf{s}_l^{(n)} = \arg \max_{\mathbf{s}_l \in \mathbf{D}_l^n} \sum_{r \in \widehat{\mathcal{R}}_n} a_{lr} b_r u_r(s_{lr})$$

where:

$$D_l^n = \left\{ \mathbf{s}_l : \sum_{r=1}^R a_{lr} s_{lr} \leq c_l, \forall_r 0 \leq s_{lr} \leq a_{lr} s_{lr}, \forall_{r \in \bigcup_{i=1}^{n-1} \widehat{\mathcal{R}}_{\phi(i)}} s_{lr} = \min_{k: a_{kr}=1} s_{kr}^{(n-1)} \right\}$$

```

12:    $n \leftarrow n + 1$ 
13: end while
14:  $\mathbf{S}^{II} \leftarrow \mathbf{S}^{(n)}$ 
15: return  $\mathbf{S}^{II}$ 

```

Algorytm - wstępne strategie

```
def compute_initial_strategies_for_single_link(link_index, routing_matrix, capacity, weights, path_weights, gamma):
    flows = routing_matrix.shape[1]
    strategies = np.zeros(flows)

    denominator = sum(
        routing_matrix[link_index][j] * (path_weights[j] * weights[j])** (1/gamma)
        for j in range(flows)
    )

    for r in range(flows):
        if routing_matrix[link_index][r] == 1:
            numerator = capacity * (path_weights[r] * weights[r])** (1/gamma)
            strategies[r] = numerator / denominator

    return strategies
```

$$s_{lr} = a_{lr} c_l \frac{(b_r w_r)^{1/\gamma}}{\sum_{j=1}^R a_{lj} (b_j w_j)^{1/\gamma}}$$

```
1:  $\mathbf{S}^{(1)} \leftarrow \mathbf{S}'$ 
2:  $\widehat{\mathcal{R}}_1 \leftarrow \{1, \dots, R\}$ 
3:  $\mathcal{L}_1 \leftarrow \{l : \sum_{r=1}^R a_{lr} (\min_{k: a_{kr}=1} s_{kr}^{(1)}) = c_l\}$ 
4:  $n \leftarrow 2$ 
5: while  $n \leq L$  do
6:    $\widehat{\mathcal{R}}_n \leftarrow \widehat{\mathcal{R}}_{n-1} \setminus \mathcal{R}_{\phi(n-1)}$ ,
   where:
    $\phi(i) := \min \mathcal{L}_i$ ,
    $\mathcal{L}_i = \left\{ l : \sum_{r=1}^R a_{lr} \left( \min_{k: a_{kr}=1} s_{kr}^{(i)} \right) = c_l \right\}$ 
7:   if  $\widehat{\mathcal{R}}_n = \emptyset$  then
8:      $\mathbf{S}^{(n)} \leftarrow \mathbf{S}^{(n)}$ 
9:     return  $\mathbf{S}^{(n)}$ 
10:   end if
11:   determine  $\mathbf{S}^{(n)}$ :
```

$$s_l^{(n)} = \arg \max_{s_l \in D_l^n} \sum_{r \in \widehat{\mathcal{R}}_n} a_{lr} b_r u_r(s_{lr})$$

where:

$$D_l^n = \left\{ s_l : \sum_{r=1}^R a_{lr} s_{lr} \leq c_l, \forall_r 0 \leq s_{lr} \leq a_{lr} s_{lr}, \forall_{r \in \bigcup_{i=1}^{n-1} \widehat{\mathcal{R}}_{\phi(i)}} s_{lr} = \min_{k: a_{kr}=1} s_{kr}^{(n-1)} \right\}$$

Algorytm - strategie w iteracji

```
def compute_strategy_for_single_link_in_iteration(link_index, routing_matrix, capacity, weights, path_weights, gamma, non_saturated_flows, previous_strategies):
    flows = routing_matrix.shape[1]
    strategies = np.zeros(flows)
    saturated_flows = set(range(flows)) - non_saturated_flows

    for r in saturated_flows:
        if routing_matrix[link_index][r] == 1:
            flow_capacity = min([previous_strategies[i][r] for i in range(routing_matrix.shape[0]) if routing_matrix[i][r] == 1])
            strategies[r] = flow_capacity
            capacity -= flow_capacity

    denominator = sum(
        routing_matrix[link_index][j] * (path_weights[j] * weights[j])** (1/gamma)
        for j in non_saturated_flows
    )

    for r in non_saturated_flows:
        if routing_matrix[link_index][r] == 1:
            numerator = capacity * (path_weights[r] * weights[r])** (1/gamma)
            strategies[r] = numerator / denominator

    return strategies
```

```
7: if  $\widehat{\mathcal{R}}_n = \emptyset$  then
8:    $S^{\text{ll}} \leftarrow S^{(n)}$ 
9:   return  $S^{\text{ll}}$ 
10: end if
11: determine  $S^{(n)}$ :
```

$$s_l^{(n)} = \arg \max_{s_l \in \mathcal{B}_l^n} \sum_{r \in \mathcal{R}_n} a_{lr} b_r u_r(s_{lr})$$

where:

$$D_l^n = \left\{ s_l : \sum_{r=1}^R a_{lr} s_{lr} \leq c_l, \forall_r 0 \leq s_{lr} \leq a_{lr} s_{lr}, \forall_{r \in \bigcup_{i=1}^{n-1} \widehat{\mathcal{R}}_{\neq(i)}} s_{lr} = \min_{k: a_{kr} > 0} s_{kr}^{(n-1)} \right\}$$

```
12:  $n \leftarrow n + 1$ 
13: end while
14:  $S^{\text{ll}} \leftarrow S^{(n)}$ 
15: return  $S^{\text{ll}}$ 
```


Algorytm

```
def capacity_allocation_game(links, capacities, routing_matrix, weights, gamma=1):
    num_flows = len(weights)

    strategies = [compute_initial_strategies_for_single_link(l, routing_matrix, capacities[l], weights, weights, gamma) for l in range(links)]
    non_saturated_flows = set(range(num_flows))
    links_that_run_out_of_capacity = [
        l for l in range(links) if sum(
            min([strategies[i][r] for i in range(links) if routing_matrix[i][r] == 1]) for r in range(num_flows) if routing_matrix[l][r] == 1
        ) == capacities[l]
    ]

    n = 2
    while n <= links:
        iteration_saturated_flows = set([r for r in range(num_flows) if any(routing_matrix[l][r] == 1 for l in links_that_run_out_of_capacity)])
        non_saturated_flows = non_saturated_flows - iteration_saturated_flows

        # Adjust strategies based on the iteration_saturated_flows
        new_strategies = [compute_strategy_for_single_link_in_iteration(l, routing_matrix, capacities[l], weights, weights, gamma, non_saturated_flows, strategies) for l in range(links)]
        strategies = new_strategies
        links_that_run_out_of_capacity = [
            l for l in range(links) if sum(
                min([strategies[i][r] for i in range(links) if routing_matrix[i][r] == 1]) for r in range(num_flows) if routing_matrix[l][r] == 1
            ) == capacities[l]
        ]

        n += 1
        if len(non_saturated_flows) == 0:
            break

    return strategies
```

4: $n \leftarrow 2$
5: while $n \leq L$ do

12: $n \leftarrow n + 1$

7: if $\widehat{\mathcal{R}}_n = \emptyset$ then
8: $\mathbf{S}^{II} \leftarrow \mathbf{S}^{(n)}$
9: return \mathbf{S}^{II}
10: end if

15: return \mathbf{S}^{II}

1: $\mathbf{S}^{(1)} \leftarrow \mathbf{S}^I$
2: $\widehat{\mathcal{R}}_1 \leftarrow \{1, \dots, R\}$
3: $\mathcal{L}_1 \leftarrow \left\{ l : \sum_{r=1}^R a_{lr} \left(\min_{k: a_{kr}=1} s_{kr}^{(1)} \right) = c_l \right\}$

6: $\widehat{\mathcal{R}}_n \leftarrow \widehat{\mathcal{R}}_{n-1} \setminus \mathcal{R}_{\phi(n-1)}$,
where:
 $\phi(i) := \min \mathcal{L}_i$, $\mathcal{L}_i = \left\{ l : \sum_{r=1}^R a_{lr} \left(\min_{k: a_{kr}=1} s_{kr}^{(i)} \right) = c_l \right\}$

11: determine $\mathbf{S}^{(n)}$:

Instancja problemu z artykułu

```
links = 2
capacities = [10, 100]
routing_matrix = np.array([
    [1, 1, 0],
    [1, 0, 1],
])
weights = [1, 1, 1]
gamma = 1

final_allocations = capacity_allocation_game(links, capacities, routing_matrix, weights, gamma)

allocations_df = pd.DataFrame(final_allocations, columns=[f"Przepływ {i+1}" for i in range(len(weights))],
                               index=[f"Połączenie {i+1}" for i in range(links)])

allocations_df
```

	Przepływ 1	Przepływ 2	Przepływ 3
Połączenie 1	5.0	5.0	0.0
Połączenie 2	5.0	0.0	95.0



Temat 5


Sieci samoorganizujące się

Jakub Radzik 260366
Kamil Herbertko 260389
Tomasz Płóciennik 260404

Specjalistyczne technologie
w sieciach informatycznych
nowej generacji

dr inż. Maciej Hojda

Funkcja optymalizacyjna



```
def optimize_with_gurobi(capacities, routing_matrix, weights, gamma):
    num_links = len(capacities)
    num_flows = len(weights)

    model = gp.Model("Flow Allocation")

    x = model.addVars(num_links, num_flows, lb=0, name="x")

    if gamma == 1:
        ...
    else:
        ...

    model.optimize()

    if model.status == GRB.OPTIMAL:
        allocations = np.zeros((num_links, num_flows))
        for l in range(num_links):
            for r in range(num_flows):
                if routing_matrix[l, r] == 1:
                    allocations[l, r] = x[l, r].X
        return allocations
    else:
        print("Optimization was not successful.")
        return None
```

Gamma równe 1

```
if gamma == 1:
    aux_vars = model.addVars(num_flows, lb=0, name="aux")
    aux_min_vars = model.addVars(num_flows, lb=0, name="aux_min")
    aux_log_vars = model.addVars(num_flows, lb=0, name="aux_log")
    aux_vars_prod = model.addVars(num_flows, lb=0, name="aux_prod")

    for r in range(num_flows):
        model.addConstr(aux_min_vars[r] == gp.min_([x[l, r] for l in range(num_links) if routing_matrix[l, r] == 1]),
                        name=f"min_alloc_{r}")

    for r in range(num_flows):
        model.addConstr(aux_log_vars[r] == aux_min_vars[r] + 1e-6,
                        name=f"log_alloc_{r}")

    for r in range(num_flows):
        model.addGenConstrLog(aux_log_vars[r], aux_vars[r], name=f"aux_constraint_{r}")

    for r in range(num_flows):
        model.addConstr(aux_vars_prod[r] == aux_vars[r] * weights[r],
                        name=f"prod_alloc_{r}")

    objective = gp.quicksum(aux_vars_prod[r]
                            for r in range(num_flows))
    model.setObjective(objective, GRB.MAXIMIZE)
```

Gamma různé od 1

```
if gamma != 1:
    aux_vars = model.addVars(num_flows, lb=0, name="aux")
    aux_min_vars = model.addVars(num_flows, lb=0, name="aux_min")

    for r in range(num_flows):
        model.addConstr(aux_min_vars[r] == gp.min_([x[l, r] for l in range(num_links) if routing_matrix[l, r] == 1]),
                        name=f"min_alloc_{r}")

    for r in range(num_flows):
        model.addConstr(aux_vars[r] == (aux_min_vars[r] + 1e-6)**(1-gamma) , name=f"aux_constraint_{r}")
    objective = gp.quicksum(weights[r] / (1 - gamma) * aux_vars[r]
                             for r in range(num_flows))
    model.setObjective(objective, GRB.MAXIMIZE)

for l in range(num_links):
    model.addConstr(gp.quicksum(x[l, r] for r in range(num_flows) if routing_matrix[l, r] == 1) <= capacities[l],
                    name=f"capacity_{l}")

for r in range(num_flows):
    model.addConstr(gp.quicksum(x[l, r] for l in range(num_links) if routing_matrix[l, r] == 1) >= 0,
                    name=f"non_negativity_{r}")
```

Przykładowe uruchomienie

```
links = 5
capacities = [30, 100, 50, 40, 100]
routing_matrix = np.array([
    [1, 1, 0, 0, 0],
    [1, 0, 1, 1, 0],
    [1, 0, 1, 0, 1],
    [1, 0, 1, 1, 1],
    [1, 1, 1, 0, 0],
])
weights = [1,1,1,1,1]
gamma = 1

allocations = optimize_with_gurobi(capacities, routing_matrix, weights, gamma)
if allocations is not None:
    print("Optimal Allocations:")
    print(allocations)
```

Wynik



Best objective 1.227655832807e+01, best bound 1.227765360545e+01, gap 0.0089%

Optimal Allocations:

```
[[ 7.35088903 22.64911093 0.          0.          0.          ]
 [ 7.35088903 0.          10.88303698 10.88303698 0.          ]
 [ 7.35088903 0.          10.88303698 0.          10.88303698]
 [ 7.35088903 0.          10.88303698 10.88303698 10.88303698]
 [ 7.35088903 22.64911093 10.88303698 0.          0.          ]]
```


Parametry



W ramach testów zostanie sprawdzony wpływ tych parametrów na wyniki:

1. Gamma
2. Rozrzut wagi
3. Ilość połączeń
4. Ilość przepływów
5. Rozrzut pojemności

Dla każdego parametru badamy 3 różne wartości z 3 powtórzeniami

Parametry c.d.



Z rozkładu normalnego losujemy:

- wagi,
- pojemności.

Wagi po wylosowaniu zostaną przeskalowane, aby sumowały się do 1.

Hipotezy



1. Algorytm powinien być szybszy od solvera
2. Solver powinien dostarczać wyniki o wyższej wartości użyteczności niż algorytm

Do testów hipotezy zastosujemy test t-studenta dla prób niezależnych



Temat 5

Sieci samoorganizujące się

Jakub Radzik 260366
Kamil Herbertko 260389
Tomasz Płóciennik 260404

Specjalistyczne technologie
w sieciach informatycznych
nowej generacji

dr inż. Maciej Hojda

Wstęp i instancje problemu

Bazowe parametry instancji problemu:

- gamma - 0.5
- rozrzut wag - 0.1
- ilość połączeń - 3
- ilość przepływów - 3
- rozrzut pojemności - 1,

Zakresy zmienności parametrów;

- gamma: [0.3, 0.5, 0.7, 1, 2],
- rozrzut wag: [0, 0.1, 1, 3, 5],
- ilość połączeń: [2, 3, 4, 6, 10],
- ilość przepływów: [2, 3, 4, 6, 10],
- rozrzut pojemności: [0.05, 0.1, 0.2, 0.5, 1]

Przykładowa wylosowana instancja:

pojemności: [9.4, 9.54, 11.21]

macierz przepływów:

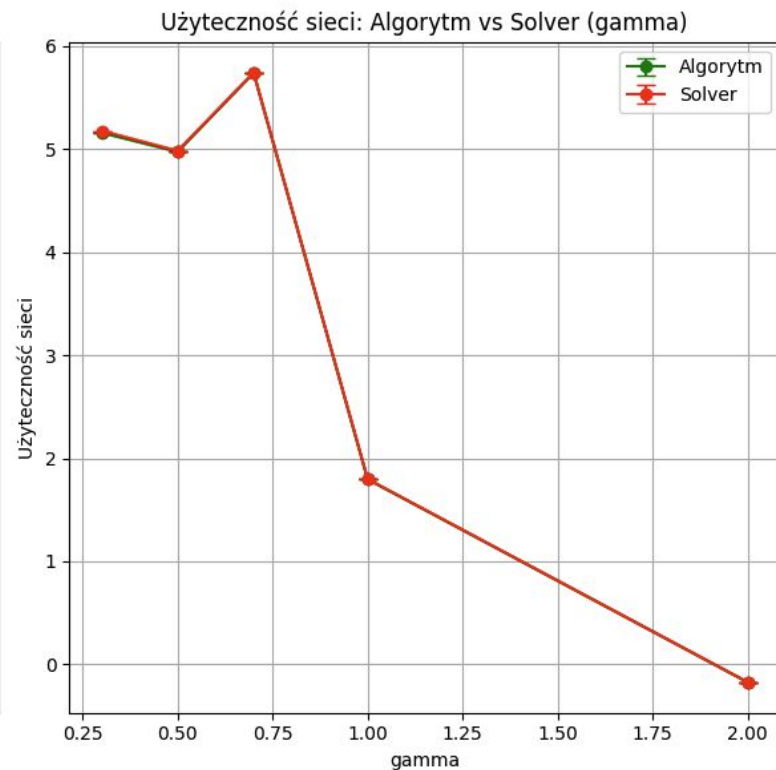
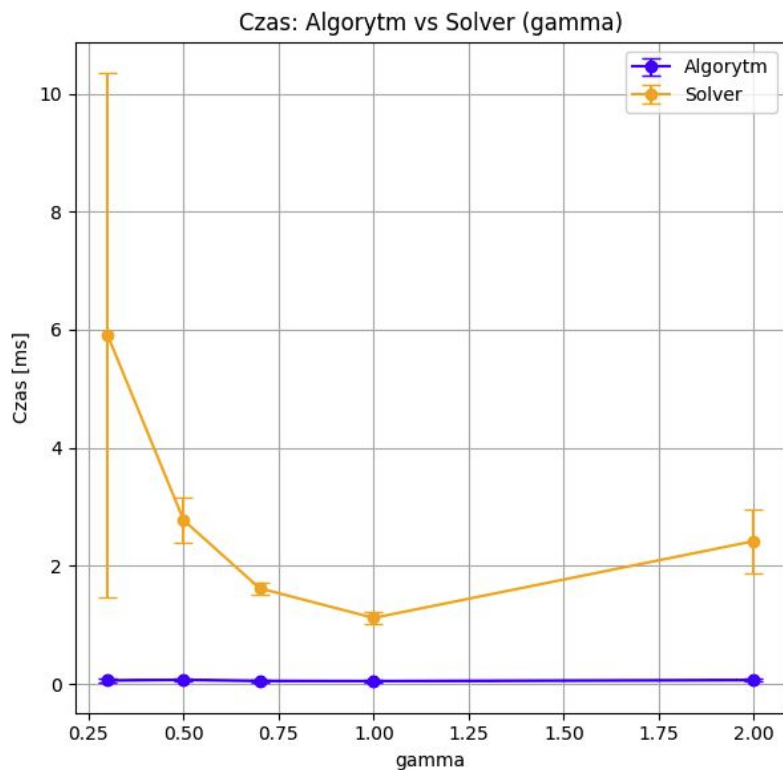
[[0, 1, 0],

[1, 0, 1],

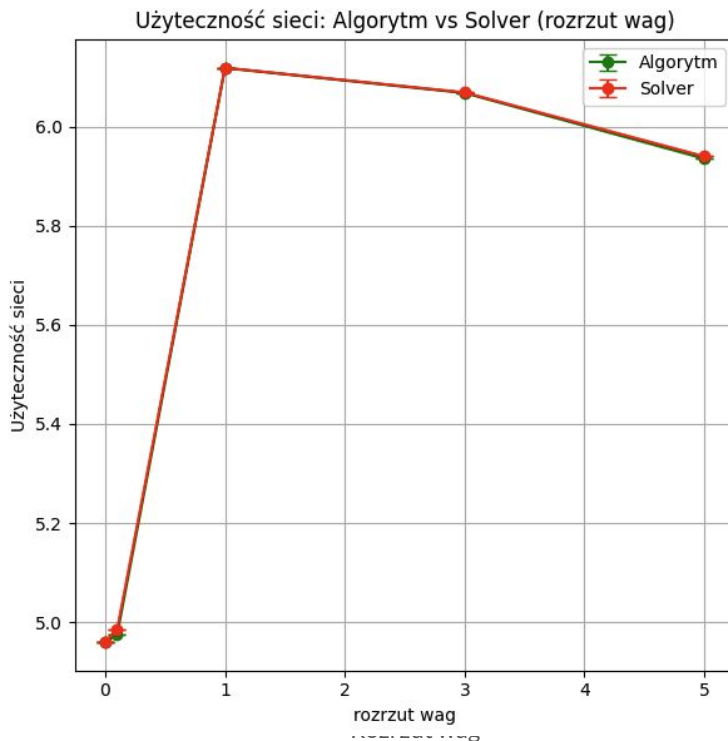
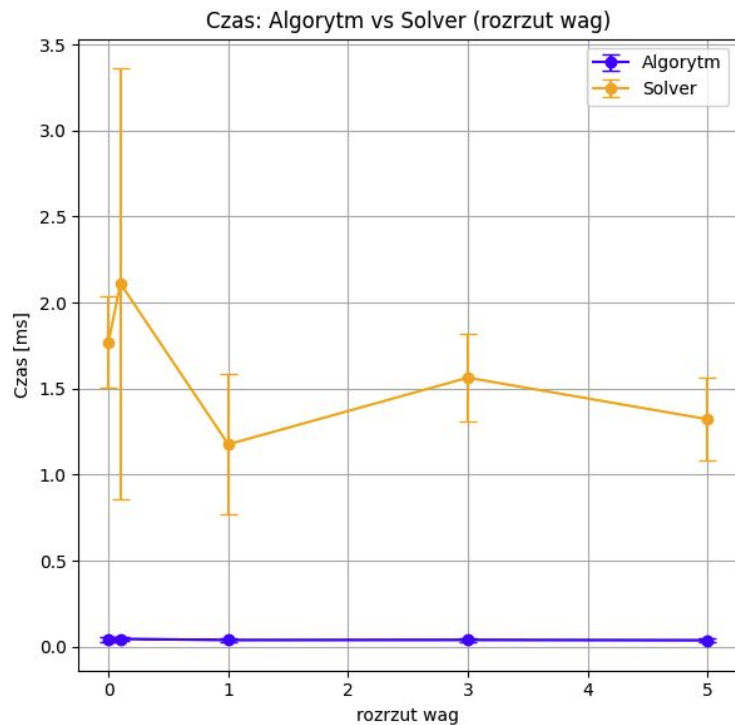
[0, 0, 1]]

wagi: [0.35 0.34, 0.30]

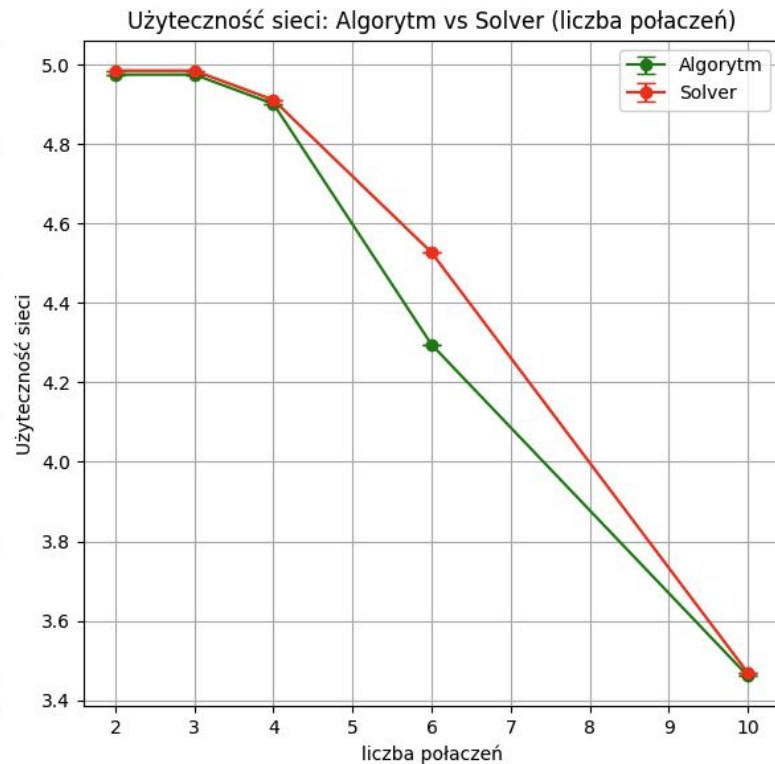
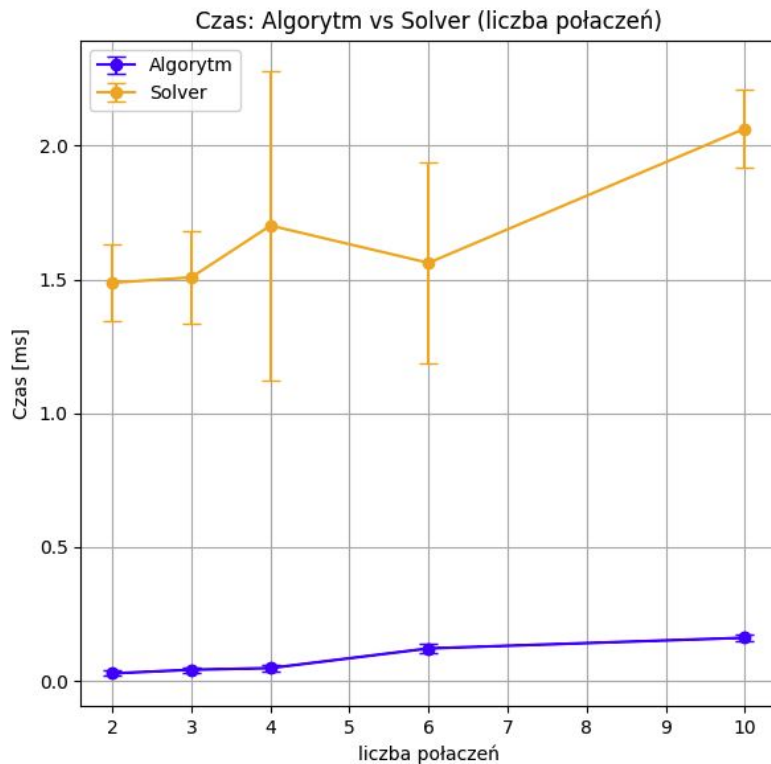
Czas i użyteczność sieci dla zmiennej gammy



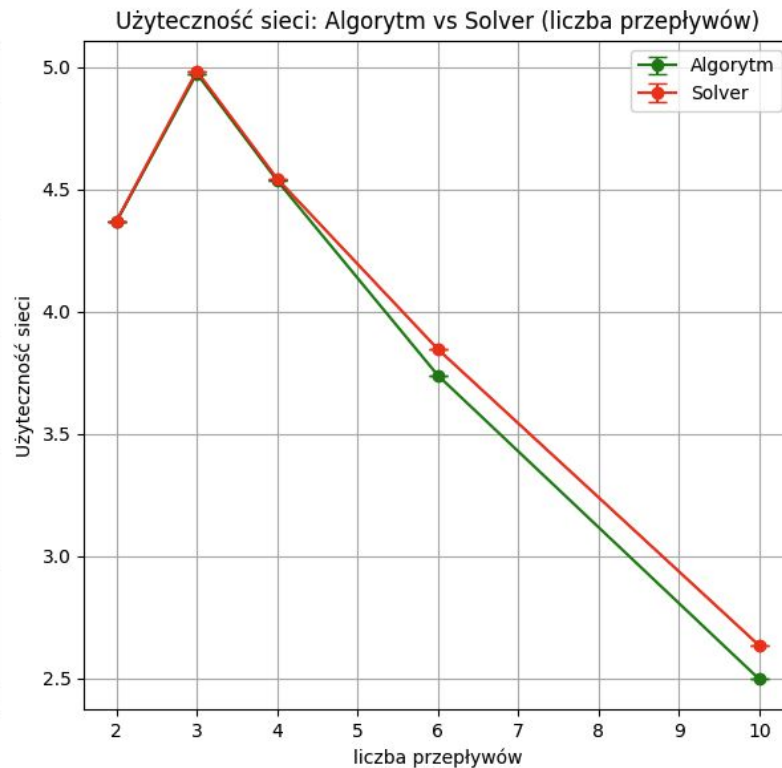
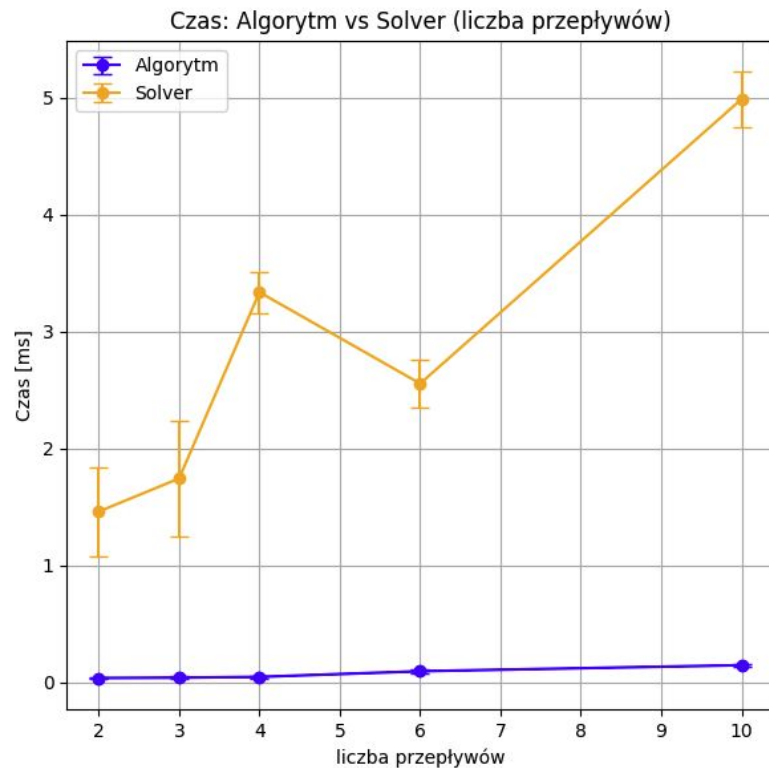
Czas i użyteczność sieci dla zmiennego rozrzutu wag



Czas i użyteczność sieci dla zmiennej liczby połączeń



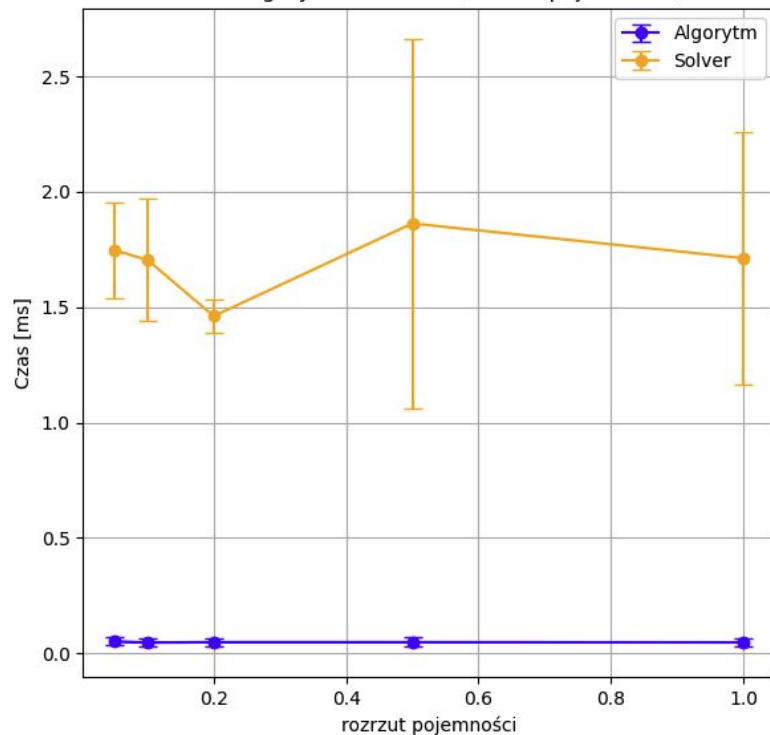
Czas i użyteczność sieci dla zmiennej liczby przepływów



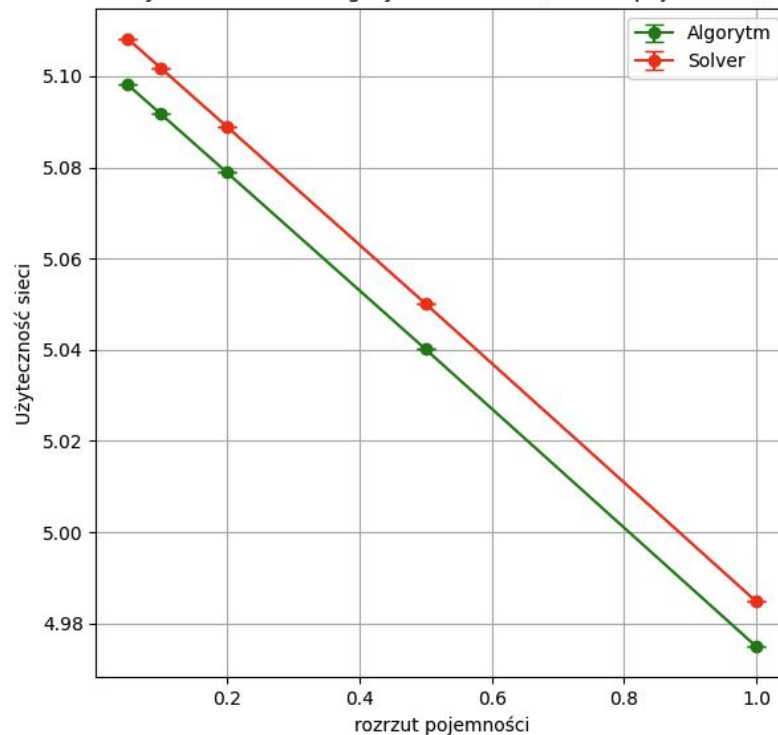
Czas i użyteczność sieci dla zmiennego rozrzutu pojemności



Czas: Algorytm vs Solver (rozrzut pojemności)



Użyteczność sieci: Algorytm vs Solver (rozrzut pojemności)



Porównanie czasu wykonania algorytmu i solvera




Hipoteza zerowa H_0 - Średni czas wykonania algorytmu jest równy średniemu czasowi wykonania solvera.

Hipoteza alternatywna H_1 - Średni czas wykonania algorytmu jest mniejszy niż średni czas wykonania solvera.


Zakładany poziom istotności $\alpha = 1\%$.

Porównanie czasu wykonania algorytmu i solvera c.d.



	Średni czas wykonania algorytmu	Średni czas wykonania solvera
	5.9680673682	590.7297134
	6.8944325903	277.3920695
	5.2083662013	162.0690028
	4.8597333565	111.8262609
	6.4916326664	241.8677012
	4.2305337653	177.0973206
	4.5221997425	211.0640208
	3.8513661517	117.6039378
	3.9180323559	156.3707987
	3.7500004206	132.1951548
	2.8166677415	148.7255096
	4.1735659276	150.7679621
	4.7985997905	170.1037089
	12.1402990771	156.1721166
	16.1305671403	206.3194911
	3.4361000871	145.7691193
	3.7486005264	174.0932465
	4.4166663429	333.3648046
	9.2540993743	255.2350362
	14.4111332096	498.5332489
	5.1402661484	174.665451
	4.7111000943	170.5090205
	4.8416327142	146.1426417
	4.8430326084	186.308225
P-value	1.68891E-09	4.7722666447
		171.2322235

Porównanie czasu wykonania algorytmu i solvera - wnioski



Jako, że p-value jest mniejsze niż zakładany poziom istotności, to odrzucamy H_0 na rzecz H_1 .

Porównanie użyteczności sieci z algorytmu i solvera




Hipoteza zerowa H_0 - Średnia użyteczność sieci z rozwiązania algorytmu jest równy średniej użyteczności z rozwiązania solvera.

Hipoteza alternatywna H_1 - Średnia użyteczność sieci z rozwiązania algorytmu jest różna od średniej użyteczności z rozwiązania solvera.

Zakładany poziom istotności $\alpha = 1\%$.

Porównanie użyteczności sieci z algorytmu i solvera c.d.



		Średnia użyteczność sieci rozwiązania algorytmu	Średnia użyteczność sieci rozwiązania solvera
		51.59377781	51.79194215
		49.74836252	49.84700313
		57.39547729	57.44879846
		17.95306121	17.97681611
		-1.742337587	-1.739822506
		49.60522399	49.60522811
		49.74836252	49.84700313
		61.18175462	61.18460982
		60.67751057	60.68732521
		59.3510975	59.39839693
		49.74836252	49.84700313
		49.74836252	49.84700313
		49.01074612	49.10938677
		42.95189339	45.26872841
		34.61044167	34.68841644
		43.68852645	43.70064591
		49.74836252	49.84700313
		45.36678086	45.44657507
		37.39897564	38.47023004
		24.96941953	26.343621
		50.98129694	51.08215411
		50.91715844	51.01790017
		50.78863574	50.88914618
		50.40107887	50.50089223
P-value	0.948831497	49.74836252	49.84700313

Porównanie użyteczności sieci z algorytmu i solvera - wnioski



Jako, że p-value jest większe niż zakładany poziom istotności, to nie mamy podstaw do odrzucenia H_0 .

Wnioski



Z powyższych wykresów oraz testów statystycznych możemy wywnioskować, że rozwiązania oferowane przez algorytm z artykułu są porównywalne jakościowo do rozwiązań oferowanych przez solver, natomiast algorytm znajduje rozwiązania w znacząco krótszym czasie.