Paulina Luksa Jakub Nowak Kacper Papuga	Badania Operacyjne i Logistyka	Informatyka Techniczna, 3 rok - semestr 6	grupa ITE 2 zestaw 3

Sprawozdanie z projektu *„Pośrednik”*

1. Zagadnienie pośrednika

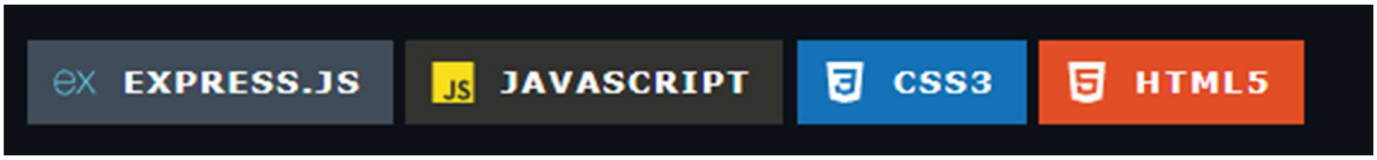
Zagadnienie transportowe - służy do obliczania najkorzystniejszego rozplanowania wielkości dostaw homogenicznego towaru pomiędzy **M** dostawcami a **N** odbiorcami. Zazwyczaj oblicza się minimalny całkowity koszt transportu. Zadanie nazywane jest ***zbilansowanym*** jeżeli całkowite możliwości dostawcze równe są całkowitemu popytowi. W przeciwnym razie zadanie jest ***niezbilansowane*** (algorytm opracowany na potrzeby projektu realizuje zarówno zagadnienie zbilansowane jak i niezbilansowane).

Zagadnienie pośrednika - zagadnienie pośrednika jest odwróconym zagadnieniem transportowym (celem jest maksymalizacja zysków pośrednika). Algorytm rozwiązania składa się z następujących etapów:

1. Wyznaczenie jednostkowego zysku z na poszczególnych trasach od dostawców do odbiorców na podstawie **cen sprzedaży, kosztów zakupu** oraz **kosztów jednostkowych transportu.**
2. Wprowadzenie do tablicy transportowej **fikcyjnego dostawcy** (o podaży równej całkowitemu popytowi) oraz **fikcyjnego odbiorcy** (o popycie równym całkowitej podaży) o jednostkowych zyskach z równych „0”.
3. Wyznaczenie pierwszego przybliżenia z wykorzystaniem metody ***„maksymalnego elementu macierzy”*** lub ***„wierzchołka północno-zachodniego”*.**
4. Wyznaczenie zmiennych dualnych na podstawie tras **bazowych**. Wyznaczenie zmiennych kryterialnych dla tras **niebazowych**.
5. W przypadku jeżeli któraś ze zmiennych kryterialnych będzie miała wartość dodatnią należy wybrać pętlę zmian oraz dokonać na jej podstawie nowego obsadzenia tras i powrócić do **punktu 5**.

2. Temat projektu oraz opis użytych narzędzi informatycznych

Wykorzystane technologie:



Aplikacja realizuje **niezbilansowane** zagadnienie pośrednika dla dowolnej liczby dostawców i odbiorców. Posiada również graficzny interfejs użytkownika, który umożliwia wpisanie:

- podaży,
- popytu,
- cen zakupu i sprzedaży,
- kosztów transportu

oraz prezentuje :

- tabela zysków jednostkowych,
- tabela optymalnych przewozów
- koszt całkowity, przychód całkowity, zysk pośrednika

3. Przedstawienie przykładowego rozwiązania i fragmenty kodu odpowiadające za ważniejsze funkcjonalności algorytmu:

Repozytorium projektu ze kodem źródłowym oraz szczegółowym opisem:
https://github.com/Qba02/Broker-problem

Algorytm rozwiązywania zadania pośrednika został ujęty w głównej funkcji modułu **middleman-calc-module, w pliku core.js.** Funkcja ***solveIntermediaryProblem*** odpowiedzialna jest za zawarcie całej logiki takiego problemu na, który składa się:

- Walidacja wejścia algorytmu,
- Sprawdzenie bilansu problemu (obliczenie podaży i popytu),
- Odpowiednie utworzenie dostawcy i odbiorcy fikcyjnego dla zadania niezbilansowanego,
- Obliczenie zysków jednostkowych,
- Znalezienie podstawowego rozwiązania (rozkładu tras) za pomocą jednej z metod: wierzchołka północno-zachodniego, największego elementu macierzy lub zmodyfikowanej wersji z warunkiem nieujemności trasy dla maksymalnego elementu macierzy,
- Zastosowanie algorytmu e-perturbacji w przypadku pojawienia się jako początkowego rozkładu tras, rozkładu niezdegenerowanego (liczba tras < n+m-1)
- Rozwiązanie pętli optymalizacyjnej, na którą składają się:
    - Obliczenie potencjałów rozwiązania
    - Obliczenie wskaźników optymalności
    - Wyznaczenie cyklu zmian
    - Wyznaczenie nowego rozwiązania
    - Zapisanie kroków optymalizacji (algorytm umożliwia sprawdzenie każdego kroku optymalizacji)

- Wybranie kroku (jednego z rozwiązań z zapisanych w pętli optymalizacyjnej + wstępnego rozwiązania) który zwraca największą wartość zysku pośrednika (maksymalizuje funkcję celu)
- Zwrócenie odpowiednich wartości

Fragment głównej logiki

```
function solveIntermediaryProblem(suppliers, consumers, supply, demand,
purchaseCosts, sellingCosts, transportationCosts) {

    try {
        validateInputs(suppliers, consumers, supply, demand, purchaseCosts,
sellingCosts, transportationCosts);
    } catch (error) {
        return { error: error.message };
    }

    const totalSupply = supply.reduce((acc, val) => acc + val, 0);
    const totalDemand = demand.reduce((acc, val) => acc + val, 0);

    let unitProfits = calculateUnitProfits(sellingCosts, purchaseCosts,
transportationCosts);

    let isBalanced = totalSupply === totalDemand;
    if (isBalanced) {…} else {
        …
    }


    allocationTable = getInitialFeasibleSolutionMaxMatrixElementMethod(supply,
demand, unitProfits, allocationTable, isBalanced)

    if (actualBaseRoutes < requiredBaseRoutes) {
        allocationTable = applyEPerturbation(allocationTable,
requiredBaseRoutes);
    }

    let optimized = false;
    while (!optimized) {
        const { deltas, deltaTable } = calculateDeltas(allocationTable,
unitProfits);


        if (deltas.every(delta => delta <= 0)) {
            optimized = true;
        } else {
            let newAllocationTable = optimizeAllocation(allocationTable,
deltaTable);
            …
```

```
            if (steps.some(step => JSON.stringify(step.postOptimizationTable)
=== JSON.stringify(newAllocationTable))) {
                break;
            }        }

        let intermediaryProfit = calculateTotalProfit(postOptimizationTable,
unitProfits);


        steps.push({
            preOptimizationTable,
            postOptimizationTable,
            deltas,
            deltaTable,
            intermediaryProfit
        });
    }

    let bestStep = steps.reduce((max, step) => step.intermediaryProfit >
max.intermediaryProfit ? step : max, steps[0]);
    allocationTable = bestStep.postOptimizationTable;

    const totalCost = calculateTotalCost(allocationTable, transportationCosts,
purchaseCosts, sellingCosts);
    const totalRevenue = calculateTotalRevenue(allocationTable, sellingCosts,
purchaseCosts);
    const intermediaryProfit = calculateTotalProfit(allocationTable,
unitProfits);


    return { allocationTable, allocationTableRealRoutes, unitProfits, steps,
totalCost, totalRevenue, intermediaryProfit };
}
```

Powyższa funkcja do realizacji zadania korzysta z wielu funkcji zdefiniowanych w pliku **helpers.js.** Najważniejszymi z tych funkcji są funkcje odpowiedzialne za obliczanie początkowego rozwiązania ***getInitialFeasibleSolutionMaxMatrixElementMethod,*** oraz za optymalizację i znajdowanie cyklów optymalizacyjnych: ***optimizeAllocation*** oraz ***findSteppingStonePath:***

```
function getInitialFeasibleSolutionMaxMatrixElementMethod(supply, demand,
unitProfits, allocationTable, isBalanced) {
    let remainingSupply = [...supply];
    let remainingDemand = [...demand];

    while (remainingSupply.some(s => s > 0) && remainingDemand.some(d => d >
0)) {
        let maxProfit = -Infinity;
```

```
        let maxRow = -1;
        let maxCol = -1;

        /***
         Number(!isBalanced) excludes fictitious supplier and consumer if
exists
         ***/
        for (let i = 0; i < unitProfits.length - Number(!isBalanced); i++) {
            for (let j = 0; j < unitProfits[i].length - Number(!isBalanced);
j++) {
                if (remainingSupply[i] > 0 && remainingDemand[j] > 0 &&
unitProfits[i][j] > maxProfit) {
                    maxProfit = unitProfits[i][j];
                    maxRow = i;
                    maxCol = j;
                }
            }
        }

        if (maxRow === -1 || maxCol === -1) break;

        let allocation = Math.min(remainingSupply[maxRow],
remainingDemand[maxCol]);
        allocationTable[maxRow][maxCol] = allocation;
        remainingSupply[maxRow] -= allocation;
        remainingDemand[maxCol] -= allocation;
    }

    while (remainingSupply.some(s => s > 0) && remainingDemand.some(d => d >
0)) {
        let maxProfit = -Infinity;
        let maxRow = -1;
        let maxCol = -1;

        for (let i = 0; i < unitProfits.length; i++) {
            for (let j = 0; j < unitProfits[i].length; j++) {
                if (remainingSupply[i] > 0 && remainingDemand[j] > 0 &&
unitProfits[i][j] > maxProfit) {
                    maxProfit = unitProfits[i][j];
                    maxRow = i;
                    maxCol = j;
                }
            }
        }

        if (maxRow === -1 || maxCol === -1) break;

        let allocation = Math.min(remainingSupply[maxRow],
remainingDemand[maxCol]);
```

```
        allocationTable[maxRow][maxCol] = allocation;
        remainingSupply[maxRow] -= allocation;
        remainingDemand[maxCol] -= allocation;
    }

    return allocationTable;
}
```

```
function optimizeAllocation(allocationTable, deltaTable) {
    let maxDelta = -Infinity;
    let pos = { i: -1, j: -1 };

    for (let i = 0; i < deltaTable.length; i++) {
        for (let j = 0; j < deltaTable[i].length; j++) {
            if (deltaTable[i][j] !== null && deltaTable[i][j] > maxDelta) {
                maxDelta = deltaTable[i][j];
                pos = { i, j };
            }
        }
    }

    if (maxDelta <= 0) {
        return allocationTable;
    }

    const { cycle, valid } = findSteppingStonePath(allocationTable, pos);

    if (!valid) {
        console.error("Invalid cycle found!");
        return allocationTable;
    }

    let minAllocation = Infinity;
    for (let k = 1; k < cycle.length; k += 2) {
        const { i, j } = cycle[k];
        minAllocation = Math.min(minAllocation, allocationTable[i][j]);
    }

    for (let k = 0; k < cycle.length; k++) {
        const { i, j } = cycle[k];
        if (k % 2 === 0) {
            allocationTable[i][j] += minAllocation;
        } else {
            allocationTable[i][j] -= minAllocation;
        }
    }
```

```
    return allocationTable;
}

function findSteppingStonePath(allocationTable, pos) {
    const cycle = [];
    const rows = allocationTable.length;
    const cols = allocationTable[0].length;

    function isBasicVariable(i, j) {
        return allocationTable[i][j] > 0;
    }

    cycle.push({ i: pos.i, j: pos.j, type: 'positive' });

    const tempValue = allocationTable[pos.i][pos.j];
    allocationTable[pos.i][pos.j] = 1;

    function searchNextElement(current, direction, excluded) {
        const { i, j } = current;
        if (direction === 'horizontal') {
            for (let jj = 0; jj < cols; jj++) {
                if (jj !== j && isBasicVariable(i, jj) && !(excluded &&
excluded.i === i && excluded.j === jj)) {
                    return { i, j: jj };
                }
            }
        } else if (direction === 'vertical') {
            for (let ii = 0; ii < rows; ii++) {
                if (ii !== i && isBasicVariable(ii, j) && !(excluded &&
excluded.i === ii && excluded.j === j)) {
                    return { i: ii, j };
                }
            }
        }
        return null;
    }

    let current = { i: pos.i, j: pos.j };
    let direction = 'horizontal';
    let type = 'negative';
    let excluded = null;

    while (true) {
        let nextElement = searchNextElement(current, direction, excluded);
        if (!nextElement) {
            /***
             Backtrack: remove the last element and try a different path
             We are looking for next elements step by step, that is why
```

```
            we can find the element that does not have its next element,
            and also is not the ending point (starting point), so we need
            to remove it and search for another option excluding this one
            element in current iteration, so a new one can be found
            ***/
            excluded = cycle.pop();
            if (cycle.length === 0) break;
            current = cycle[cycle.length - 1];
            direction = direction === 'horizontal' ? 'vertical' :
'horizontal';
            type = cycle.length % 2 === 0 ? 'positive' : 'negative';
            continue;
        }

        if (nextElement.i === pos.i && nextElement.j === pos.j) {
            break;
        }

        excluded = null;
        nextElement.type = type;
        cycle.push(nextElement);

        current = nextElement;
        direction = direction === 'horizontal' ? 'vertical' : 'horizontal';
        type = cycle.length % 2 === 0 ? 'positive' : 'negative';
    }

    allocationTable[pos.i][pos.j] = tempValue;

    if (validateCycle(cycle,allocationTable) &&
validateSupplyDemand(allocationTable,cycle)) {
        return { cycle, valid: true };
    } else {
        return { cycle, valid: false };
    }
}
```

Funkcjonalność modułu do rozwiązywania problemu została udostępniona poprzez API. Aplikacja backendowa działa przy wykorzystaniu **node.js** jako środowiska runtimowego, oraz biblioteki **express** do stowrzenia prostego **endpointu /calculate.** Definicja tego serwera znajduje się w pliku service.js i wygląda następująco:

```
app.post('/calculate', (req, res) => {
    const { suppliers, consumers, supply, demand, purchaseCosts, sellingCosts,
transportationCosts } = req.body;
    const result = solveIntermediaryProblem(suppliers, consumers, supply,
demand, purchaseCosts, sellingCosts, transportationCosts);
```

```
    if (result.error) {
        res.status(400).json({ error: result.error });
    } else {
        res.json(result);
    }
});

app.listen(3000, () => {
    console.log('Server is running on port 3000');
});
```

Aplikacja backendowa w przypadku poprawnego rozwiązania zadania zwraca kod 200, a w przypadku błędu walidacji lub błędu algorytmu kod 400.

Ponadto działanie algorytmu jest testowane za pomocą frameworku do testowania kodu w JavaScript – Jest. Testy jednostkowe obejmują 11 przykładów zdefiniowanych w pliku core.test.js:

```
/***
 While adding new test cases, test only intermediaryProfit,
 as this is the goal function that we are maximizing, thus
 different algorithm may find different allocationTables with same profit

 Note: expect(result.allocationTable).toEqual commented below are these which
were found by algorithm implemented in core.js
 ***/

describe('solveIntermediaryProblem Tests', () => {
    test('Test Case 1', () => {
        const result = solveIntermediaryProblem(["s1","s2"], ["c1","c2","c3"],
[20,30], [10,28,27], [10,12], [30,25,30], [[8,14,17],[12,9,19]]);
        //
expect(result.allocationTable).toEqual([[10,0,10,0],[0,28,0,2],[0,0,17,48]]);
        expect(result.intermediaryProfit).toBe(262);
    });
…
}
```

Wysłanie zapytania w postaci (w środowisku Postman):

Zwraca wynik:

{

"allocationTable": [

[16, 0, 4, 0],

[0, 0, 20, 20],

[0, 12, 0, 40]

],

"allocationTableRealRoutes": [

[16, 0, 4],

[0, 0, 20]

],

"unitProfits": [

[7, 2, 6, 0],

[2, -2, 3, 0],

[0, 0, 0, 0]

],

"steps": [

{

"preOptimizationTable": [

[16, 0, 4, 0],

[0, 12, 20, 8],

[0, 0, 0, 52]

],

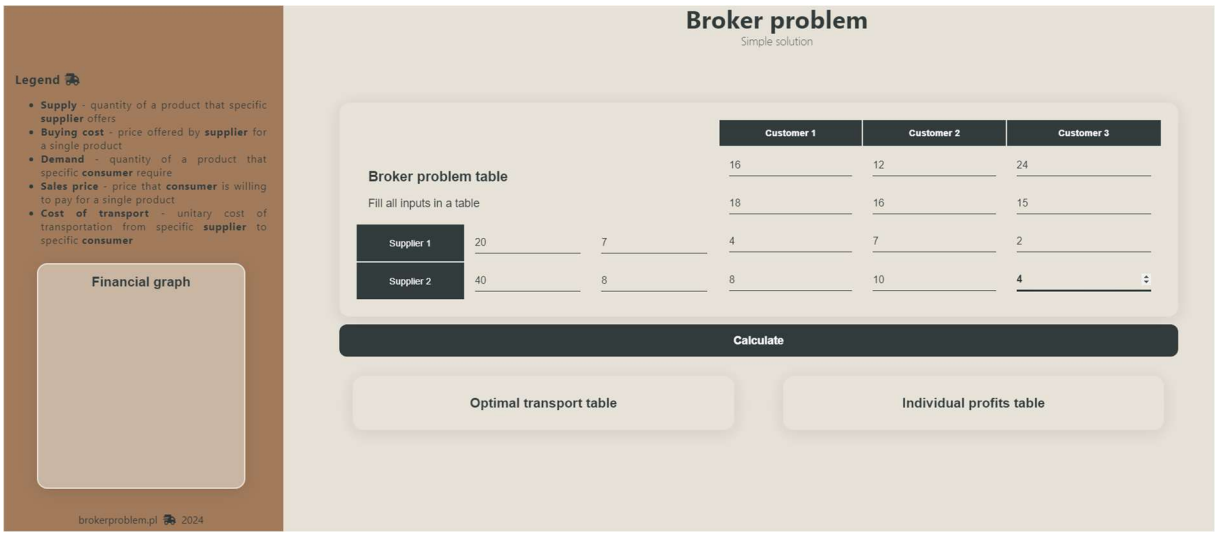
```
"postOptimizationTable": [
[16, 0, 4, 0],
[0, 0, 20, 20],
[0, 12, 0, 40]
],
"deltas": [
1, -3, -2, -4, 2, -3
],
"deltaTable": [
[null, 1, null, -3],
[-2, null, null, null],
[-4, 2, -3, null]
]
},
{
"preOptimizationTable": [
[16, 0, 4, 0],
[0, 0, 20, 20],
[0, 12, 0, 40]
],
"postOptimizationTable": [
[16, 0, 4, 0],
[0, 0, 20, 20],
[0, 12, 0, 40]
],
"deltas": [
-1, -3, -2, -2, -4, -3
],
"deltaTable": [
[null, -1, null, -3],
[-2, -2, null, null],
[-4, null, -3, null]
]
}
],
```

```
"totalCost": 452,
"totalRevenue": 648,
"intermediaryProfit": 196
}
```

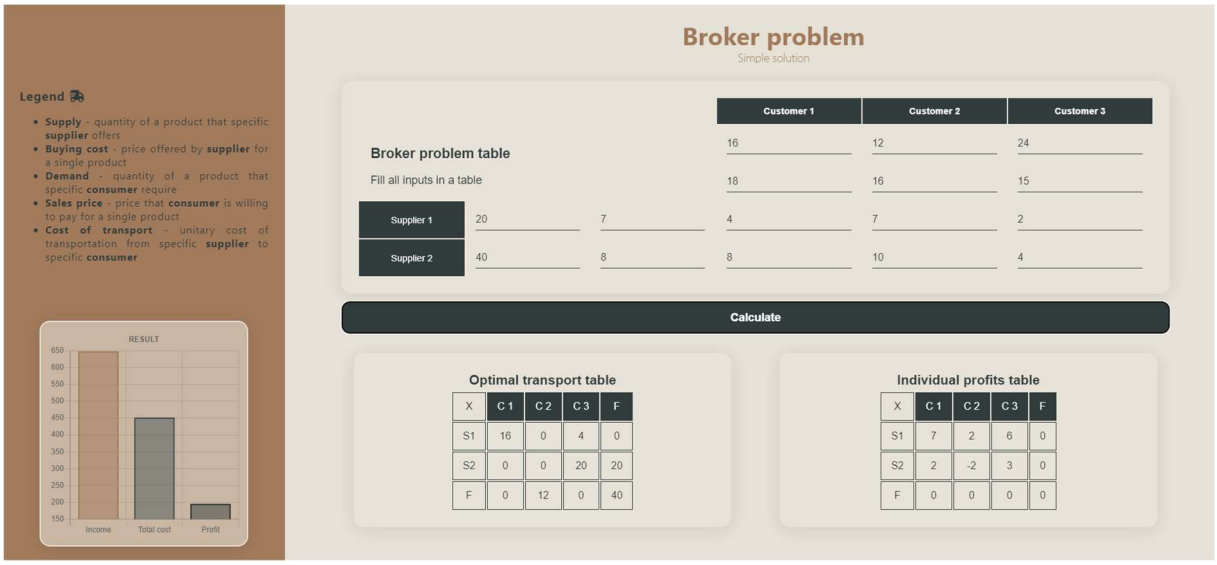

Oznaczenia zwracanych wartości:

```
Description of returned values:
- `allocationTable` - table of routes after end of an algorithm, includes fictitious actors (fictitious supplier and consumer),
- `allocationTableRealRoutes` - as above but only with real actors,
- `unitProfits` - profits on each route calculates as selling_price - (transportation_cost + purchase_cost),
- `steps` - steps of new optimized routes:
  - `preOptimizationTable` - routes before optimization in this step
  - `postOptimizationTable` - routes after
  - `deltas` - optimization indicators,
  - `deltaTable` - deltas in tables
- `totalCost`
- `totalRevenue`
- `intermediaryProfit`
```

Wysłanie analogicznego zapytania poprzez aplikację frontendową:



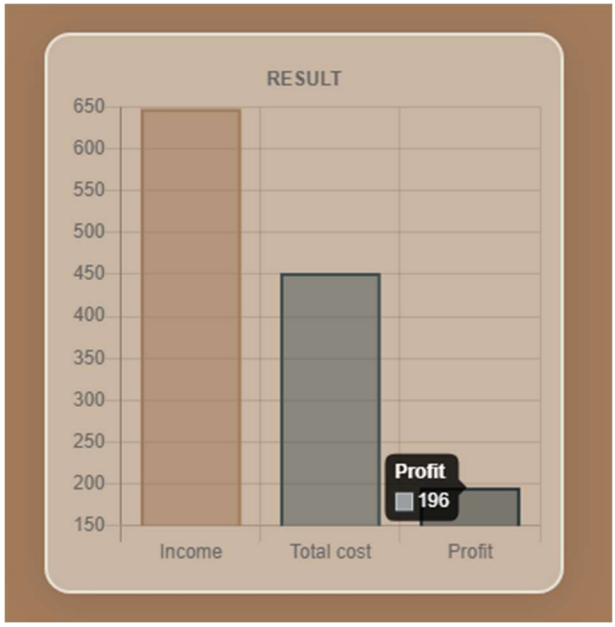
Wynik:



Wynik przestawia:

➔ Rozkład optymalnych tras oraz zysków indywidualnych (dla zadania z fikcyjnymi aktorami zwraca także ich wartości):

➔ Rozkład kosztów, zysków brutto i zysków netto:



Harmonogram projektu „Pośrednik”

dla naszej grupy

13 maj 2024

Projekt	Stan	Powiązane pliki	Uwagi (podział obowiązków)
Opracowanie algorytmu	Uruchomiono	Plik	Kacper Kuba Paulina
Implementacja algorytmu	Uruchomiono	Plik	Kacper
Utworzenie tabeli do wprowadzania danych wejściowych	Uruchomiono	Plik	Paulina
Czytelna wizualizacja danych wyjściowych	Uruchomiono	Plik	Kuba
Dodanie wielu odbiorców/dostawców	Uruchomiono	Plik	
Obsługa błędów	Uruchomiono	Plik	Paulina
Sprawozdanie	Uruchomiono	Plik	Kuba

- **priorytet wyższy**
- **priorytet niższy**