Metode rješavanja transportnih problema - izrada programskog rješenja

Florijan, Aleks

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:384394

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Aleks Florijan

Matični broj: 0016154404

Studij: Informacijski i poslovni sustavi

METODE RJEŠAVANJA TRANSPORTNIH PROBLEMA - IZRADA PROGRAMSKOG RJEŠENJA

ZAVRŠNI RAD

Mentor:

Dr. sc. Nenad Perši

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

Aleks Florijan

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Ovaj završni rad posvećen je analizi i rješavanju transportnih problema kroz opis i primjenu različitih metoda optimizacije. Glavni fokus rada je na metodama poput metode sjeverozapadnog kuta, metode minimalnih troškova, Vogel-ove metode i MODI metode, koje su objašnjene i primijenjene na oglednim primjerima. Osim teorijskog pregleda, rad uključuje i razvoj programske aplikacije u C# jeziku korištenjem Windows Forms, koja omogućuje praktičnu primjenu metoda. Pored toga, istražena je i pojava degeneracije u transportnim problemima te su predstavljene strategije za njeno rješavanje.

Ključne riječi: transportni problemi, metode optimizacije, sjeverozapadni kut, minimalni troškovi, Vogel-ova metoda, MODI metoda, C# razvoj, Windows Forms, degeneracija

Sadržaj

1. Uvod

Transportni problemi predstavljaju jednu od ključnih tema u području operacijskih istraživanja i optimizacije, s širokom primjenom u logistici, distribuciji i upravljanju lancem opskrbe. Cilj transportnog problema je pronaći najoptimalniji način raspodjele proizvoda iz više izvora prema više odredišta uz minimalne troškove prijevoza, poštujući pritom kapacitete izvora i potrebe odredišta.

U okviru ovog rada istražit će se različite metode rješavanja transportnih problema, među kojima su metoda sjeverozapadnog kuta, metoda minimalnih troškova, Vogel-ova metoda i MODI metoda. Svaka od ovih metoda ima specifičan pristup u određivanju početnih rješenja i optimizaciji transportnog plana, te će biti objašnjena i analizirana, uz pažnju na pojavu degeneracije koja može otežati proces optimizacije.

Osim teorijskog pregleda, rad uključuje razvoj programskog rješenja u programskom jeziku C# koristeći Windows Forms, koji omogućuje primjenu ovih metoda na konkretnim primjerima. Aplikacija je osmišljena kao alat koji korisnicima omogućuje rješavanje transportnih problema, pružajući uvid u korake procesa rješavanja.

U konačnici, rad ima za cilj osigurati sveobuhvatno razumijevanje transportnih problema, od njihovih osnovnih načela do implementacije u obliku softverske aplikacije, što je neophodno za uspješno rješavanje izazova u ovom području, uključujući i specifične slučajeve poput degeneracije.

2. Metode i tehnike rada

Razvoj programskog rješenja za rješavanje transportnih problema proveden je koristeći najnovije alate i tehnologije kako bi se postigao kvalitetan i efikasan rezultat. Projekt je realiziran u Visual Studio 2022 razvojnom okruženju, koje omogućuje napredno upravljanje projektima, brzu kompilaciju i jednostavno otklanjanje pogrešaka, što je bilo ključno za uspješan razvoj.

Aplikacija je izrađena u C# programskom jeziku, odabranom zbog njegove robusnosti i jednostavnosti korištenja, uz podršku .NET Framework-a. Grafičko korisničko sučelje razvijeno je korištenjem Windows Forms tehnologije, koja omogućuje intuitivnu interakciju s aplikacijom.

U radu su implementirani algoritmi za rješavanje transportnih problema, uključujući Metodu sjeverozapadnog kuta, Metodu minimalnih troškova i Vogel-ovu metodu. Evaluacija rješenja uključivala je provjeru ispravnosti rezultata i analizu korisničkog iskustva, čime se osigurava da aplikacija bude jednostavna za korištenje.

3. Transportni problem linearnog programiranja

Transportni problem predstavlja model optimizacije kojim se nastoji minimizirati ukupni trošak transporta dobara između izvorišta i odredišta. Ovaj problem pronalazi optimalnu raspodjelu robe iz više izvorišta prema više odredišta, pri čemu su zadani kapaciteti izvorišta i zahtjevi odredišta. [1]

Odredište / **Izvorište**	B_1	B_2	B_n	**Ponuda** a_i
A_1	c_{11} x_{11}	c_{12} x_{12}	c_{1n} x_{1n}	a_1
A_2	c_{21} x_{21}	c_{22} x_{22}	c_{2n} x_{2n}	a_2
A_m	c_{m1} x_{m1}	c_{m2} x_{m2}	c_{mn} x_{mn}	a_m
Potražnja b_j	b_1	b_2	b_n	

Tablica 1: Matrica transporta

U formulaciji transportnog problema, imamo m izvorišta i n odredišta. Svako izvorište i (gdje je $i = 1, \dots, m$) raspolaže s količinom robe a_i, koja se mora distribuirati prema određenim odredištima. Svako odredište j (gdje je $j = 1, \dots, n$) zahtijeva količinu robe b_j, koju mora primiti iz jednog ili više izvorišta.

Ključna varijabla u ovom modelu je x_{ij}, koja predstavlja količinu robe koja se prenosi iz izvorišta A_i do odredišta B_j. Trošak transporta jedne jedinice robe između izvorišta A_i i odredišta B_j označava se kao c_{ij}. Cilj optimizacije je minimizirati ukupne troškove transporta, što se matematički izražava kao zbroj svih pojedinačnih troškova $c_{ij} \times x_{ij}$ za sve kombinacije izvorišta i odredišta.

Model transportnog problema osigurava da se svi kapaciteti izvorišta i zahtjevi odredišta u potpunosti zadovolje, a da se pritom minimiziraju ukupni troškovi

transporta. Ovakva optimizacija igra ključnu ulogu u učinkovitoj distribuciji resursa, smanjenju operativnih troškova i povećanju učinkovitosti unutar lanca opskrbe.

Funkcija cilja transportnog problema definirana je kao:

$$T = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}\ x_{ij}\ \rightarrow \min$$

U ovoj formuli, c_{ij} predstavlja cijenu transporta jedinice robe od izvorišta A_i do odredišta B_j, dok x_{ij} označava količinu robe koja se prenosi iz A_i u B_j. Cilj je minimizirati ukupni trošak T , koji se sastoji od zbroja svih pojedinačnih troškova $c_{ij} \times x_{ij}$ na svim relacijama između izvorišta i odredišta.

Funkcija cilja, označena sa Z , predstavlja ukupne troškove transporta koje nastojimo minimizirati kako bismo postigli optimalno rješenje. Time osiguravamo učinkovito korištenje resursa, smanjenje troškova i potpuno zadovoljenje zahtjeva odredišta. [1]

3.1 Zatvoreni model transporta

Zatvoreni model transporta je specifična verzija transportnog problema gdje su ukupna ponuda i ukupna potražnja jednaki, što znači da se sva raspoloživa roba mora distribuirati bez ostataka. Ovaj model optimizira transportne troškove pod uvjetom da se zadovolje svi zahtjevi odredišta i da se iskoriste svi kapaciteti izvorišta. [2]

Za zatvoreni model transporta, ukupna ponuda mora biti jednaka ukupnoj potražnji:

$$\sum_{i=1}^{m} a_i\ = \sum_{j=1}^{n} b_j$$

gdje:

- a_i predstavlja ponudu na izvorištu A_i (za $i = 1,2, \dots, m$),
- b_j predstavlja potražnju na odredištu B_j (za $j = 1,2, \dots, n$).

Funkcija cilja ovog problema, koju želimo minimizirati, je ukupni transportni trošak:

$$Z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} \times x_{ij}$$

gdje:

- c_{ij} označava trošak transporta jedne jedinice robe s izvorišta A_i do odredišta B_j
- x_{ij} je količina robe koja se transportira s A_i do B_j

3.1.1 Ograničenja

Zatvoreni model mora zadovoljiti sljedeća ograničenja:

1. **Ograničenja ponude (izvorišta):**

Svako izvorište ima fiksnu količinu robe koja mora biti distribuirana prema odredištima:

$$\sum_{j=1}^{n} x_{ij} = b_j\,, \quad za\ svaki\ j = 1,2,\dots,n$$

Ovo ograničenje osigurava da količina robe koja odlazi iz izvorišta A_i ne prelazi njegovu ponudu a_i. [1]

2. **Ograničenja potražnje (odredišta)**

Svako odredište zahtijeva određenu količinu robe koja mora biti u potpunosti zadovoljena:

$$\sum_{i=1}^{m} x_{ij} = b_j\,, \quad za\ svaki\ j = 1,2,\dots,n$$

Ovo ograničenje osigurava da količina robe koja odlazi iz izvorišta B_j ne prelazi njegovu ponudu b_j. [1]

3. **Ograničenja ne-negativnosti:**

Količine robe koje se transportiraju moraju biti ne-negativne:

$$x_{ij} \geq 0, \quad za\ svaki\ i = 1,2,\dots,m \quad i \quad j = 1,2,\dots,n$$

3.2 Otvoreni transportni problem

Otvoreni transportni problem nastaje kada ukupni kapacitet izvorišta nije jednak ukupnom kapacitetu odredišta. Ovaj problem možemo izraziti kao neravnotežu između sume proizvoda na izvorištima i sume potreba na odredištima:

$$\sum_{i=1}^{m} a_i \neq \sum_{j=1}^{n} b_j$$

Višak se može pojaviti na strani izvorišta ili na strani odredišta, ovisno o situaciji, te prema tome razlikujemo dvije vrste otvorenog transportnog problema [2]:

a) Otvoreni transportni problem s viškom u ponudi

b) Otvoreni transportni problem s viškom u potražnji

3.2.1 Otvoreni transportni problem s viškom u ponudi

Kod otvorenog transportnog problema s viškom u ponudi, ukupni kapacitet izvorišta je veći od ukupnog kapaciteta odredišta:

$$\sum_{i=1}^{m} a_i > \sum_{j=1}^{n} b_j$$

Kako bismo izjednačili ove količine, dodajemo fiktivno odredište koje preuzima višak ponude. Ovaj višak u sustavu bilježimo kao dodatnu potražnju koja omogućava balansiranje problema. Na taj način, originalno neuravnoteženi sustav postaje zatvoren i spreman za rješavanje. [3]

Nakon dodavanja fiktivnog odredišta, funkcija cilja se proširuje na sljedeći način:

$$Z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} \times x_{ij} + \sum_{i=1}^{m} c_{if} \times x_{if} \rightarrow min$$

gdje je x_{if} količina tereta koja ide prema fiktivnom odredištu, a trošak c_{if} je jednak nuli jer to odredište fizički ne postoji.

3.2.2 Otvoreni transportni problem s viškom u potražnji

S druge strane, kada se višak pojavljuje na strani odredišta, tj. kada ukupna potražnja prelazi ukupnu ponudu, situacija se rješava dodavanjem fiktivnog izvorišta:

$$\sum_{i=1}^{m} a_i \; < \; \sum_{j=1}^{n} b_j$$

U ovom slučaju, dodajemo fiktivno izvorište s dovoljno proizvoda kako bi se zadovoljila preostala potražnja. Funkcija cilja se opet proširuje, ali sada uključuje dodatne varijable koje predstavljaju količine proizvoda s fiktivnog izvorišta:

$$Z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} \times x_{ij} + \sum_{j=1}^{n} c_{fj} \times x_{fj} \; \rightarrow \; min$$

gdje x_{fj} predstavlja količinu proizvoda koja dolazi s fiktivnog izvorišta, a trošak c_{fj} je kao i prije jednak nuli. [3]

Nakon dodavanja fiktivnog izvorišta, možemo se fokusirati na pronalaženje optimalnog rasporeda tereta, čime se početno otvoreni problem pretvara u zatvoreni transportni problem spreman za rješavanje.

3.3 Dual transportnog problema

Dual transportnog problema predstavlja problem maksimizacije, suprotno originalnom transportnom problemu koji minimizira ukupne troškove. U dualnom problemu cilj je maksimizirati iskoristivost kapaciteta izvorišta i zadovoljavanje potražnje odredišta, uz poštivanje ograničenja koja postavlja originalni problem. [1]

3.3.1 Dualne varijable

Za rješavanje dualnog problema uvode se dualne varijable povezane s izvorištima i odredištima:

- Dualne varijable za izvorišta označavaju se s u_i (za $i = 1,2, \dots, m$),
- Dualne varijable za odredišta označavaju se s v_j (za $j = 1,2, \dots, n$).

Ove varijable često se nazivaju "shadow prices" (cijene u sjeni) i imaju sljedeću interpretaciju:

- Varijabla u_i predstavlja porast vrijednosti funkcije cilja kod povećanja količine proizvoda na izvorištu A_i za jednu jedinicu,
- Varijabla v_i predstavlja smanjenje vrijednosti funkcije cilja kod smanjenja potražnje na odredištu B_j za jednu jedinicu.

3.3.2 Dualna funkcija cilja

Dualna funkcija cilja se računa kao suma umnožaka proizvoda na izvorištima i pripadnih dualnih varijabli izvorišta, te još umnožaka potražnje odredišta i pripadnih dualnih varijabli odredišta (maksimiziraju se vrijednosti Z_d pod uvjetom da se zadovolje sva ograničenja postavljena za u_i i v_j):

$$Z_d = \sum_{i=1}^{m} a_i \times u_i + \sum_{j=1}^{n} b_j \times v_j \rightarrow max$$

3.3.3 Ograničenja dualnog problema

Ograničenja dualnog problema vezana su za pojedinačne realne troškove prijevoza između izvorišta i odredišta. Svako od ovih ograničenja zahtijeva da zbroj dualnih varijabli izvorišta i odredišta na određenoj relaciji bude manji ili jednak stvarnom trošku transporta za tu relaciju. To se izražava na sljedeći način:

$$u_i + v_j \leq c_{ij}, \quad za\ svaki\ i = 1,2,\dots,m \ \text{ i } \ j = 1,2,\dots,n$$

Ova ograničenja osiguravaju da se dualne varijable u potpunosti usklade s realnim troškovima transporta u izvornom problemu.

3.3.4 Dualni problem i uvjeti ne negativnosti

Za razliku od originalnog transportnog problema, dualni problem nema definirane uvjete ne negativnosti za varijable u_i i v_j . Ovo znači da dualne varijable mogu poprimiti i negativne vrijednosti, ovisno o situaciji. Na ovaj način, dualni problem pruža dodatnu fleksibilnost u analizi kako se promjene u kapacitetu izvorišta ili potražnji odredišta odražavaju na ukupne troškove. [1]

4. Metode početnog raspoređivanja tereta

Kod rješavanja transportnih problema, jedan od ključnih koraka je inicijalno određivanje rasporeda tereta između različitih dobavljača i potrošača. Ovaj početni raspored ne samo da služi kao osnova za daljnje optimizacije, već može značajno utjecati na konačnu učinkovitost rješenja. Pravilno postavljen početni raspored može smanjiti broj iteracija potrebnih za pronalaženje optimalnog rješenja, a time i ukupne troškove transporta. Iz tog razloga, odabir odgovarajuće metode za početno raspoređivanje tereta je jako važan. [1]

Različite metode koje se koriste za izračun početnog rješenja imaju svoje specifičnosti i temelje se na različitim pristupima. Svaka metoda koristi određen skup pravila koja određuju kako će se teret alocirati između dobavljača i potrošača. Ove metode mogu biti jednostavnije, poput metode sjeverozapadnog kuta, ili složenije, poput Vogel-ove aproksimativne metode, koja pokušava uzeti u obzir što više faktora već u inicijalnoj fazi raspodjele. Bez obzira na kompleksnost, svaka metoda ima za cilj postići što bolje početno rješenje koje će olakšati daljnju optimizaciju.

Tri najčešće korištene metode za početno raspoređivanje tereta su metoda sjeverozapadnog kuta, metoda minimalnih troškova i Vogel-ova aproksimativna metoda. Metoda sjeverozapadnog kuta oslanja se na jednostavan i sistematičan pristup bez obzira na troškove, dok metoda minimalnih troškova odmah uzima u obzir ekonomski aspekt raspodjele. Vogel-ova aproksimativna metoda kombinira pristupe i pokušava postići ravnotežu između jednostavnosti i optimizacije troškova od samog početka. Svaka od ovih metoda donosi različite prednosti i nedostatke, što ih čini pogodnima za različite vrste transportnih problema.

4.1 Metoda sjeverozapadnog kuta

U ovom poglavlju prikazat ćemo primjenu metode sjeverozapadnog kuta (SZ metoda) na primjeru transportnog problema. Tablica 2 prikazuje početne troškove transporta između dobavljača A_i (ishodišta) i odredišta B_j, kao i dostupne količine za svaki dobavljač i potražnje svakog odredišta.

Početna tablica sljedećeg primjera izgleda ovako:

P. Odredište / **Izvorište**	B_1	B_2	B_3	B_4	**Ponuda** a_i
A_1	3	1	7	4	300
A_2	2	6	5	9	400
A_3	8	3	3	2	500
Potražnja b_j	250	350	400	200	

Tablica 2: SZ - Početna tablica

Metoda sjeverozapadnog kuta započinje s alokacijom resursa iz gornjeg lijevog (sjeverozapadnog) kuta tablice. Cilj metode je postaviti početno bazično ne degenerirano rješenje tako da se u svaki korak dodjeljuje maksimalno moguća količina robe x_{ij} koja se može alocirati u trenutno polje tablice, uzimajući u obzir raspoloživu ponudu i preostalu potražnju. [2] [4]

1. Odredište / Izvorište	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3 **250**	1	7	4	50
A_2	2	6	5	9	400
A_3	8	3	3	2	500
Potražnja	0	350	400	200	

Tablica 3: SZ – Prva alokacija

S obzirom da je maksimalna količina koja može biti transportirana iz A_1 prema B_1 250 (jer je to potražnja odredišta B_1), ta količina se alocira. Kapacitet A_1 sada iznosi 50, dok je B_1 u potpunosti zadovoljen.

2. Odredište / Izvorište	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3 **250**	1 **50**	7	4	0
A_2	2	6	5	9	400
A_3	8	3	3	2	500
Potražnja	0	300	400	200	

Tablica 4: SZ – Druga alokacija

Preostali kapacitet A_1 (50 jedinica) sada se alocira prema B_2 , smanjujući njegovu potražnju na 300. Kapacitet A_1 je sada u potpunosti iskorišten.

3. Odredište / **Izvorište**	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3 **250**	1 **50**	7	4	0
A_2	2	6 **300**	5	9	100
A_3	8	3	3	2	500
Potražnja	0	0	400	200	

Tablica 5: SZ – Treća alokacija

Nakon što je A_1 u potpunosti iskorišten, prelazimo na A_2 . Preostala potražnja B_2 je 300, stoga se tih 300 jedinica alocira iz A_2 prema B_2. Time je B_2 u potpunosti zadovoljen, a kapacitet A_2 smanjen na 100.

4. Odredište / **Izvorište**	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3 **250**	1 **50**	7	4	0
A_2	2	6 **300**	5 **100**	9	0
A_3	8	3	3	2	200
Potražnja	0	0	300	200	

Tablica 6: SZ – Četvrta alokacija

Preostali kapacitet A_2 koristi se za djelomično zadovoljenje . Sada potražnja B_3 iznosi 300, dok je A_2 u potpunosti iskorišten.

5. Odredište / **Izvorište**	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3 **250**	1 **50**	7	4	0
A_2	2	6 **300**	5 **100**	9	0
A_3	8	3	3 **300**	2	0
Potražnja	0	0	0	200	

Tablica 7: SZ – Peta alokacija

Dobavljač A_3 sada zadovoljava preostalu potražnju B_3, te se njegov kapacitet smanjuje na 200.

6. Odredište / **Izvorište**	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3 **250**	1 **50**	7	4	0
A_2	2	6 **300**	5 **100**	9	0
A_3	8	3	3 **300**	2 **200**	0
Potražnja	0	0	0	0	

Tablica 8: SZ – Šesta alokacija

Konačno, preostali kapacitet A_3 se koristi za zadovoljenje potražnje B_4.

Ukupni trošak transporta prema ovom rješenju iznosi **4400** jedinica, što je rezultat umnožaka količina x_{ij} i pripadajućih troškova c_{ij} za svaku od alokacija. $Z = 3 \cdot 250 + 50 + 6 \cdot 300 + 5 \cdot 100 + 3 \cdot 300 + 2 \cdot 200 = 4400$

4.2 Metoda minimalnih troškova

U ovom poglavlju prikazat ćemo primjenu metode minimalnog troška na primjeru transportnog problema. Kao i u prethodnoj metodi, tablica 9 prikazuje početne troškove transporta između dobavljača A_i (ishodišta) i odredišta B_j, kao i dostupne količine za svaki dobavljač i potražnje svakog odredišta. [2][4]

Početna tablica sljedećeg primjera izgleda ovako:

P. Odredište / Izvorište	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3	1	7	4	300
A_2	2	6	5	9	400
A_3	8	3	3	2	500
Potražnja	250	350	400	200	

Tablica 9: MMT - Početna tablica

Metoda minimalnog troška započinje s alokacijom iz polja tablice s najmanjim troškom c_{ij}. Nakon svake alokacije, prilagođava se raspoloživa ponuda i potražnja te se ponavlja postupak dok se ne zadovolje sve potrebe.

1. Odredište **Izvorište**	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3	1 **300**	7	4	0
A_2	2	6	5	9	400
A_3	8	3	3	2	500
Potražnja	250	50	400	200	

Tablica 10: MMT – Prva alokacija

Prvo se alocira maksimalna moguća količina prema odredištu B_2 jer je trošak $c_{12} = 1$ najmanji. Nakon ove alokacije, A_1 više nema kapaciteta, a potreba B_2 smanjena je na 50.

2. Odredište **Izvorište**	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3	1 **300**	7	4	0
A_2	2	6	5	9	400
A_3	8	3	3	2 **200**	300
Potražnja	250	50	400	0	

Tablica 11: MMT – Druga alokacija

Sljedeća alokacija je prema B_4 jer je trošak $c_{34} = 2$ najmanji preostali trošak. B_4 je sada u potpunosti zadovoljen, dok je kapacitet A_3 smanjen na 300.

3. Odredište / **Izvorište**	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3	1 **300**	7	4	0
A_2	2 **250**	6	5	9	150
A_3	8	3	3	2 **200**	300
Potražnja	0	50	400	0	

Tablica 12: MMT – Treća alokacija

Nakon toga, alociramo $x_{21} = 250$ prema B_1 jer je trošak $c_{21} = 2$ minimalan među preostalim troškovima. B_1 je sada u potpunosti zadovoljen.

4. Odredište / **Izvorište**	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3	1 **300**	7	4	0
A_2	2 **250**	6	5	9	150
A_3	8	3 **50**	3	2 **200**	250
Potražnja	0	0	400	0	

Tablica 13: MMT – Četvrta alokacija

Preostala potreba B_2 od 50 jedinica alocira se iz A_3 prema B_2, s obzirom na trošak $c_{32} = 3$. Time je B_2 u potpunosti zadovoljen, a A_3 sada ima preostalih 250 jedinica.

5. Odredište / Izvorište	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3	1 **300**	7	4	0
A_2	2 **250**	6	5	9	150
A_3	8	3 **50**	3 **250**	2 **200**	0
Potražnja	0	0	150	0	

Tablica 14: MMT – Peta alokacija

Količina $x_{33} = 250$ sada se alocira iz A_3 prema B_3, s troškom $c33 = 3$.

6. Odredište / Izvorište	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3	1 **300**	7	4	0
A_2	2 **250**	6	5 **150**	9	0
A_3	8	3 **50**	3 **250**	2 **200**	0
Potražnja	0	0	0	0	

Tablica 15: MMT – Šesta alokacija

Nakon što su potrebe za B_1, B_2 i B_4 u potpunosti zadovoljene, ostaje preostalih 150 jedinica za B_3. Ove jedinice alociramo iz A_2, gdje je trošak $c_{23} = 5$. Time se u potpunosti zadovoljava B_3 i koristi preostali kapacitet A_2.

$$Z = 300 + 2 \cdot 250 + 3 \cdot 50 + 3 \cdot 250 + 2 \cdot 200 + 5 \cdot 150 = 2850$$

4.3 Vogel-ova metoda

Vogel-ova metoda, poznata i kao Vogel-ova aproksimacijska metoda (VAM - Vogel's Approximation Method), je jedan od najefikasnijih algoritama za pronalaženje početnog osnovnog rješenja u transportnim problemima. Ova metoda koristi koncept penalizacije kako bi odredila optimalne početne alokacije robe iz različitih izvora prema različitim odredištima, s ciljem minimiziranja ukupnih troškova transporta.

U transportnom problemu, svaki dobavljač ima određeni kapacitet, dok svako odredište ima određenu potrebu. Troškovi transporta između svakog dobavljača i odredišta variraju, pa je cilj pronaći kombinaciju dodjele koja će rezultirati najnižim ukupnim troškovima.

Koristi penalizaciju kako bi odabrala najpovoljniju alokaciju u svakom koraku. **Penalizacija** se izračunava kao razlika između najmanjeg i drugog najmanjeg troška u svakom retku i stupcu tablice troškova. Ova razlika predstavlja potencijalnu „kaznu“ koja se primjenjuje ako ne odaberemo najniži trošak u tom retku ili stupcu. [2][4]

Početna tablica sljedećeg primjera izgleda ovako:

P. Odredište / Izvorište	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3	1	7	4	300
A_2	2	6	5	9	400
A_3	8	3	3	2	500
Potražnja	250	350	400	200	

Tablica 16: VAM - Početna tablica

1. ***Korak***

Penalizacija za retke:

- $A_1: |1-3| = 2$
- $A_2: |2-5| = 3$
- $A_3: |2-3| = 1$

Penalizacija za stupce:

- $B_1: |2-3| = 1$
- $B_2: |1-3| = 2$
- $B_3: |3-5| = 2$
- $B_4: |2-4| = 2$

Najveća penalizacija je 3 u retku A_2. Trošak u tom retku je najmanji za stupac B_1 (trošak = 2). Alociramo minimalnu količinu $\min(250,400) = 250$.

1. Odredište / Izvorište	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3	1	7	4	300
A_2	2 **250**	6	5	9	150
A_3	8	3	3	2	500
Potražnja	0	350	400	200	

Tablica 17: VAM – Prva alokacija

2. ***Korak***

Penalizacija za retke:

- $A_1: |1-3| = 2$
- $A_2: |6-9| = 3$
- $A_3: |2-3| = 1$

Penalizacija za stupce:

- B_1: $već\ zadovoljena\ potreba$
- $B_2: |1-3| = 2$
- $B_3: |3-5| = 2$
- $B_4: |2-4| = 2$

Opet je najveća penalizacija 3 u retku A_2. Najmanji trošak u A_2 je u stupcu B_3 (trošak = 5). Alociramo $\min(150,400) = 150$.

2. Odredište / Izvorište	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3	1	7	4	300
A_2	2 **250**	6	5 **150**	9	0
A_3	8	3	3	2	500
Potražnja	0	350	250	200	

Tablica 18: VAM – Druga alokacija

3. ***Korak***

Penalizacija za retke:

- A_1: $|1-3| = 2$
- A_3: $|2-3| = 1$

Penalizacija za stupce:

- B_1: $već\ zadovoljena\ potreba$
- B_2: $|1-3| = 2$
- B_3: $|3-5| = 2$
- B_4: $|2-4| = 2$

Penalizacija je najveća (2) za retke A_1 i stupce B_2, B_3, B_4. Biramo najmanji trošak u A_1 za stupac B_2 (trošak = 1). Alociramo $\min(300,350) = 300$.

3. Odredište / Izvorište	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3	1 **300**	7	4	0
A_2	2 **250**	6	5 **150**	9	0
A_3	8	3	3	2	500
Potražnja	0	50	250	200	

Tablica 19: VAM – Treća alokacija

4. ***Korak***

Penalizacija za retke:

- A_3: $|2-3| = 1$

Penalizacija za stupce:

- B_1: $već\ zadovoljena\ potreba$
- B_2: $|3-6| = 3$
- B_3: $|3-5| = 2$
- B_4: $|2-4| = 2$

Najveća penalizacija je 3 za stupac B_2, no budući da u B_2 preostaje samo 50 jedinica, alociramo $\min(50,500) = 50$ iz A_3 prema B_2 (trošak = 3).

4. Odredište / Izvorište	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3	1 **300**	7	4	0
A_2	2 **250**	6	5 **150**	9	0
A_3	8	3 **50**	3	2	450
Potražnja	0	0	250	200	

Tablica 20: VAM – Četvrta alokacija

5. Korak

Penalizacija za retke:

- A_3: $|2-3|=1$

Penalizacija za stupce:

- B_1: *već zadovoljena potreba*
- B_2: *već zadovoljena potreba*
- B_3: $|3-5|=2$
- B_4: $|2-4|=2$

Penalizacije su sada jednake (2) za stupce B_3 i B_4. Biramo stupac s manjim troškom u A_3, B_4 (trošak = 2). Alociramo $\min(200,450)=200$.

5. Odredište / Izvorište	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3	1 **300**	7	4	0
A_2	2 **250**	6	5 **150**	9	0
A_3	8	3 **50**	3	2 **200**	250
Potražnja	0	0	250	0	

Tablica 21: VAM – Peta alokacija

6. Korak

Nakon alokacije prema B_4, jedino preostaje zadovoljenje B_3 s preostalih 250 jedinica iz A_3 (trošak = 3).

6. Odredište / Izvorište	B_1	B_2	B_3	B_4	**Ponuda**
A_1	3	1 **300**	7	4	0
A_2	2 **250**	6	**5** **150**	9	0
A_3	8	3 **50**	3 **250**	2 **200**	0
Potražnja	0	0	0	0	

Tablica 22: VAM – Šesta alokacija

Ovime je završena alokacija po Vogel-ovoj metodi. Korištenjem penalizacija, uspjeli smo minimizirati ukupni trošak transporta

$$Z = 300 + 2 \cdot 250 + 3 \cdot 50 + 3 \cdot 250 + 2 \cdot 200 + 5 \cdot 150 \; = \; 2850$$

5. MODI metoda za određivanje optimalnog rješenja

MODI metoda koristi se za optimizaciju inicijalnog rješenja transportnog problema. Nakon što se inicijalno rješenje pronađe pomoću metode sjeverozapadnog kuta, metode minimalnih troškova ili Vogel-ove metode, potrebno je odrediti je li to rješenje optimalno, odnosno minimizira li ukupne troškove transporta. Da bi se to postiglo, MODI metoda započinje izračunom potencijala U_i za izvore i V_j za odredišta. Ovi potencijali izračunavaju se za svaku rednu i stupčanu varijablu tako da zadovolje uvjet $U_i + V_j = c_{ij}$, gdje je c_{ij} trošak transporta iz izvora i do odredišta j . Obično se početna vrijednost jednog potencijala postavlja na nulu, primjerice $U_1 = 0$, nakon čega se koriste poznati troškovi bazičnih varijabli za izračun ostalih potencijala. [5]

Nakon izračuna potencijala, za svaku ne bazičnu varijablu x_{ij} potrebno je izračunati vrijednost neto troškova Δ_{ij} prema formuli $\Delta_{ij} = c_{ij} - (U_i + V_j)$. Ova vrijednost pokazuje koliko bi se ukupni troškovi promijenili ako bi se određena varijabla uključila u bazično rješenje. Ako su sve vrijednosti Δ_{ij} pozitivne ili jednake nuli, rješenje se smatra optimalnim jer nijedna dodatna promjena ne može smanjiti ukupne troškove. Međutim, ako postoji vrijednost Δ_{ij} manja od nule, to pokazuje na mogućnost smanjenja troškova uvođenjem te varijable u bazu. [5]

Sljedeći korak je odabir ne bazične varijable s najnižom Δ_{ij} vrijednošću, koja će ući u rješenje, čime se pokreće proces optimizacije. Nakon što je varijabla odabrana, formira se zatvorena petlja koja uključuje horizontalne i vertikalne linije kroz trenutne bazične varijable. Ova petlja služi za redistribuciju tereta u transportnoj tablici, pri čemu se izabranoj varijabli dodaje određena količina tereta, dok se s drugih varijabli oduzima kako bi se očuvala ravnoteža ponude i potražnje. Vrijednost najmanje bazične varijable u petlji određuje koliko tereta može biti premješteno. Redistribucija tereta nastavlja se dok sve Δ_{ij} vrijednosti ne postanu pozitivne ili jednake nuli, čime se postiže optimalno rješenje. [5]

5.1 Primjer MODI metode

Pogledat ćemo sljedeći transportni problem s tri izvora (A1, A2, A3) i tri odredišta (B1, B2, B3) s poznatim troškovima transporta prikazanim u tablici. Problem se sastoji od optimizacije raspodjele tereta između izvora i odredišta tako da ukupni transportni troškovi budu minimalni.

Početna tablica sljedećeg primjera izgleda ovako:

P. Odredište / Izvorište	B_1	B_2	B_3	**Ponuda**
A_1	2	3	1	30
A_2	5	4	2	40
A_3	3	2	4	20
Potražnja	20	40	30	

Tablica 23: MODI - Početna tablica

Početno rješenje možemo dobiti sjeverozapadnom metodom. Pretpostavimo da smo već dobili sljedeće početno rješenje:

R. Odredište / Izvorište	B_1	B_2	B_3	**Ponuda**
A_1	2 20	3 10	1	30
A_2	5	4 30	2 10	40
A_3	3	2	4 20	20
Potražnja	20	40	30	

Tablica 24: MODI - Početno rješenje

1. ***Korak:*** *Izračun potencijala* U_i *i* V_j *za sve bazične varijable*

Prvi korak MODI metode je izračun potencijala za redove (U_i) i stupce (V_j). Potencijali se računaju tako da za bazične varijable vrijedi $U_i + V_j = C_{ij}$, gdje je C_{ij} trošak transporta iz izvora i u odredište j .

Postavljamo $U_1 = 0$ za početak:

$$U_1 + V_1 = 2 \quad \Rightarrow \quad V_1 = 2$$

$$U_1 + V_2 = 3 \quad \Rightarrow \quad V_2 = 3$$

$$U_2 + V_2 = 4 \quad \Rightarrow \quad U_2 = 1$$

$$U_2 + V_3 = 2 \quad \Rightarrow \quad V_3 = 1$$

$$U_3 + V_3 = 4 \quad \Rightarrow \quad U_3 = 3$$

Dobivamo sljedeće potencijale:

$$U_1 = 0, \quad U_2 = 1, \quad U_3 = 3, \quad V_1 = 2, \quad V_2 = 3, \quad V_3 = 1$$

2. ***Korak:*** *Izračun vrijednosti Δ_{ij} za nezauzete ćelije*

Nakon izračuna potencijala, izračunavaju se vrijednosti Δ_{ij} za sve nezauzete ćelije prema formuli, one predstavljaju potencijal za smanjenje ukupnog troška:

$$\Delta_{ij} = c_{ij} - (U_i + V_j)$$

Tablica s izračunatim Δ_{ij} vrijednostima izgleda ovako:

	B_1	B_2	B_3
A_1	$\Delta_{11} = 2 - (0 + 2) = 0$	$\Delta_{12} = 3 - (0 + 3) = 0$	$\Delta_{13} = 1 - (0 + 1) = 0$
A_2	$\Delta_{21} = 5 - (1 + 2) = 2$	$\Delta_{22} = 4 - (1 + 3) = 0$	$\Delta_{23} = 2 - (1 + 1) = 0$
A_3	$\Delta_{31} = 3 - (3 + 2) = -2$	$\Delta_{32} = 2 - (3 + 3) = -4$	$\Delta_{33} = 4 - (3 + 1) = 0$

Tablica 25: MODI – Delta vrijednosti 1

1.	B_1	B_2	B_3	Ponuda	
A_1	2 20	3 10	1 0	30	$U_1 = 0$
A_2	5 2	4 30	2 10	40	$U_2 = 1$
A_3	3 -2	2 -4	4 20	20	$U_3 = 3$
	$V_1 = 2$	$V_2 = 3$	$V_3 = 1$		

Tablica 26: MODI - Prvi korak

3. ***Korak:*** *Analiza i poboljšanje rješenja*

Najmanja negativna vrijednost Δ_{ij} izračunata u prethodnom koraku je $\Delta_{32} = -4$, što pokazuje na to da postoji mogućnost poboljšanja rješenja smanjenjem ukupnog troška.

Započinjemo formiranje zatvorene petlje od ćelije $(3,2)$. Zatvorena petlja mora uključivati bazične varijable i prolaziti kroz naizmjenične $+$ i $-$ znakove:

Zatvorena petlja: $(3,2) \longrightarrow (3,3) \longrightarrow (2,3) \longrightarrow (2,2) \longrightarrow (3,2)$

4. ***Korak:*** *Redistribucija tereta preko zatvorene petlje*

Identificiramo najmanju količinu tereta u petlji, što je 20 jedinica (pozicija $(3,3)$). Smanjujemo 20 jedinica na svim negativnim pozicijama i dodajemo 20 jedinica na svim pozitivnim pozicijama u petlji.

2.	B_1	B_2	B_3	Ponuda	
A_1	2 20	3 10	1	30	$U_1 = 0$
A_2	5	4 30	2 10	40	$U_2 = 1$
A_3	3	2 20	4	20	$U_3 = 3$
	$V_1 = 2$	$V_2 = 3$	$V_3 = 1$		

Tablica 27: MODI – Redistribuirana tablica

5. ***Korak:*** *Ponovna provjera optimalnosti*

Ponovimo izračun potencijala U_i i V_j za novu bazu, kako bismo provjerili je li rješenje sada optimalno.

Postavljamo $V_2 = 0$ i izračunavamo:

$$U_1 + V_2 = c_{12} \quad \longrightarrow \quad U_1 + 0 = 3 \quad \longrightarrow \quad U_1 = 3$$
$$U_1 + V_1 = c_{11} \quad \longrightarrow \quad 3 + V_1 = 2 \quad \longrightarrow \quad V_1 = -1$$
$$U_2 + V_2 = c_{22} \quad \longrightarrow \quad U_2 + 0 = 4 \quad \longrightarrow \quad U_2 = 4$$
$$U_2 + V_3 = c_{23} \quad \longrightarrow \quad 4 + V_3 = 2 \quad \longrightarrow \quad V_3 = -2$$
$$U_3 + V_2 = c_{32} \quad \longrightarrow \quad U_3 + 0 = 2 \quad \longrightarrow \quad U_3 = 2$$

Novi potencijali:

$$U_1 = 3, \;\; U_2 = 4, \;\; U_3 = 2, \;\; V_1 = -1, \;\; V_2 = 0, \;\; V_3 = -2$$

6. ***Korak:*** *Izračun Δ_{ij} za novu bazu*

Ponovno izračunavamo Δ_{ij} za sve nezauzete ćelije:

Za (1,3):

$$\Delta_{13} = c_{13} - (U_1 + V_3) = 1 - \big(3 + (-2)\big) = 0$$

Za (2,1):

$$\Delta_{21} = c_{21} - (U_2 + V_1) = 5 - \big(4 + (-1)\big) = 2$$

Za (3,1):

$$\Delta_{31} = c_{31} - (U_3 + V_1) = 3 - \big(2 + (-1)\big) = 2$$

Za (3,3):

$$\Delta_{33} = c_{33} - (U_3 + V_3) = 4 - \big(2 + (-2)\big) = 4$$

Nakon provjere Δ_{ij} vrijednosti, svi rezultati su veći ili jednaki nuli, što znači da je postignuto optimalno rješenje.

$$Z = 2 \times 20 + 3 \times 10 + 4 \times 30 + 2 \times 10 + 4 \times 20 = 290$$

$$Z_{optimizirano} = 2 \times 20 + 3 \times 10 + 4 \times 10 + 2 \times 30 + 2 \times 20 = 210$$

6. Degeneracija

Degeneracija u transportnom problemu nastaje kada broj bazičnih varijabli u početnom rješenju ne zadovoljava uvjet $m + n - 1$, gdje je m broj redaka, a n broj stupaca u transportnoj tablici. Bazične varijable su one koje sudjeluju u optimizaciji i imaju pozitivne vrijednosti, dok ne bazične varijable imaju vrijednost nula.

U transportnom problemu, ako se u nekom trenutku pojavi manje od $m + n - 1$ pozitivnih varijabli x_{ij} , rješenje postaje degenerirano. Degeneracija se obično pojavljuje kada se kapaciteti ili potražnje iscrpe prije nego što je ispunjen dovoljan broj bazičnih varijabli. [6][7]

6.1 Razlozi i posljedice degeneracije

Ako broj pozitivnih bazičnih varijabli x_{ij} u početnom rješenju nije jednak $m + n - 1$, dolazi do degeneracije. Ovo može uzrokovati probleme u daljnjoj optimizaciji jer metode kao MODI metoda zahtijevaju točan broj bazičnih varijabli kako bi pravilno funkcionirale.

Degeneracija također može uzrokovati višestruka optimalna rješenja ili dovesti do cikličkih iteracija koje ne daju konačan rezultat.

6.2 Rješavanje degeneracije

Degeneracija se rješava na način da se nedostajućim bazičnim varijablama dodijeli vrijednost nula. Ova nula služi kao "fiktivna" varijabla koja omogućuje daljnje računanje bez utjecaja na ukupni trošak rješenja.

Za transportni problem, gdje su varijable x_{ij} količine dobara prenesene iz izvora A_i u odredište B_j, rješenje degeneracije se postiže dodavanjem nule na mjesta gdje nedostaju bazične varijable.

6.3 Primjer degeneracije

Imamo tri dobavljača (A_1, A_2, A_3) i četiri odredišta (B_1, B_2, B_3, B_4). Kapaciteti dobavljača su $a_1 = 50,\ a_2 = 90,\ a_3 = 60$, dok su potrebe odredišta $b_1 = 50,\ b_2 = 40,\ b_3 = 70,\ b_4 = 40$.

Troškovi transporta između svakog dobavljača i odredišta su:

- $c_{11} = 2,\ c_{12} = 5,\ c_{13} = 4,\ c_{14} = 5$
- $c_{21} = 1,\ c_{22} = 2,\ c_{23} = 1,\ c_{24} = 4$
- $c_{31} = 3,\ c_{32} = 1,\ c_{33} = 5,\ c_{34} = 2$

Koristeći metodu sjeverozapadnog kuta za pronalaženje početnog rješenja, dobivamo sljedeće alokacije:

- $x_{11} = 50$
- $x_{22} = 40$
- $x_{23} = 50$
- $x_{33} = 20$
- $x_{34} = 40$

Međutim, imamo samo 5 bazičnih varijabli $x_{11},\ x_{22},\ x_{23},\ x_{33},\ x_{34}$, dok prema pravilu $m + n - 1 = 3 + 4 - 1 = 6$, potrebno je imati 6 bazičnih varijabli. Rješenje je stoga degenerirano.

Kako bismo riješili degeneraciju, dodajemo fiktivnu varijablu s vrijednošću nula na poziciju x_{12}. Sada imamo:

- $x_{11} = 50$
- $x_{12} = 0\ (fiktivna\ nula)$
- $x_{22} = 40$
- $x_{23} = 50$
- $x_{33} = 20$
- $x_{34} = 40$

Odredište / **Izvorište**	B_1	B_2	B_3	B_4	**Ponuda**
A_1	2 **50**	5 **0**	4	5	50
A_2	1	2 **40**	1 **50**	4	90
A_3	3	1	5 **20**	2 **40**	60
Potražnja	50	40	70	40	

Tablica 28: Degeneracija

Time je degeneracija otklonjena, a rješenje sada sadrži potrebnih 6 bazičnih varijabli, omogućavajući daljnje izračune i optimizaciju troškova. Ukupni trošak za ovo rješenje iznosi $Z = 410$ jedinica.

7. Programsko rješenje

Cilj je bio izraditi softverski alat koji korisnicima omogućava brzo i efikasno rješavanje transportnih problema koristeći različite metode, kao što su metoda sjeverozapadnog kuta, metoda minimalnih troškova i Vogel-ova metoda. Aplikacija je razvijena u programskom jeziku C# i koristi Windows Forms za grafičko korisničko sučelje (GUI), omogućujući intuitivno upravljanje i praćenje koraka rješavanja problema.

GitHub repozitorij programskog rješenja:

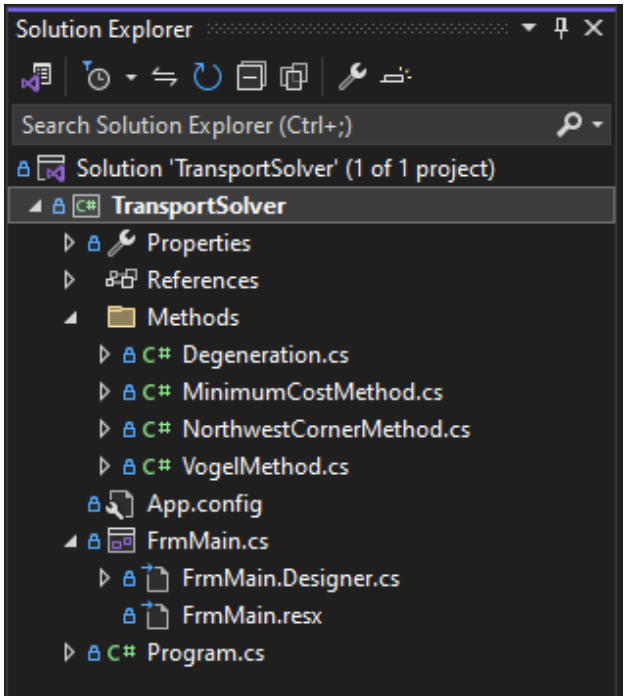
https://github.com/aflorijan21/TransportSolver

7.1 Arhitektura rješenja

Arhitektura programskog rješenja organizirana je u skladu s najboljim praksama razvoja softverskih aplikacija. Projekt je podijeljen na nekoliko ključnih komponenti:

- **Klase za metode rješavanja problema**: Svaka od implementiranih metoda rješavanja transportnih problema smještena je u zasebnu statičku klasu unutar mape `Methods`. Ovaj modularni pristup omogućuje jednostavno proširivanje funkcionalnosti aplikacije dodavanjem novih metoda ili modificiranjem postojećih. Na primjer, klasa `MinimumCostMethod` sadrži logiku za rješavanje transportnog problema metodom minimalnih troškova, dok klasa `NorthwestCornerMethod` implementira metodu sjeverozapadnog kuta. Sve klase koriste sličnu strukturu i sučelje, što omogućava jednostavnu integraciju u glavni obrazac aplikacije.
- **Glavni obrazac (FrmMain):** `FrmMain` predstavlja centralni dio korisničkog sučelja. Ovdje se nalazi logika za interakciju korisnika s aplikacijom, kao što su unos podataka (kapaciteti dobavljača, potrebe odredišta), odabir metode rješavanja, te prikaz rezultata i koraka rješavanja problema. Metode implementirane u zasebnim klasama pozivaju se iz ovog obrasca, ovisno o odabiru korisnika.

- **Dodatne komponente:** Aplikacija također sadrži korisničke kontrole, kao što su `TextBox` za unos podataka, `ComboBox` za odabir metode, te `DataGridView` za prikaz i unos vrijednosti u matricu transportnih troškova. Ove kontrole omogućuju korisniku da lako upravlja aplikacijom i prati korake rješavanja.



Slika 1: Arhitektura rješenja unutar Visual Studija

7.2 Korisničko sučelje

Korisničko sučelje programa dizajnirano je s naglaskom na jednostavnost korištenja, omogućujući korisnicima da lako upravljaju svim potrebnim funkcijama za rješavanje transportnih problema. Elementi sučelja su sljedeći:

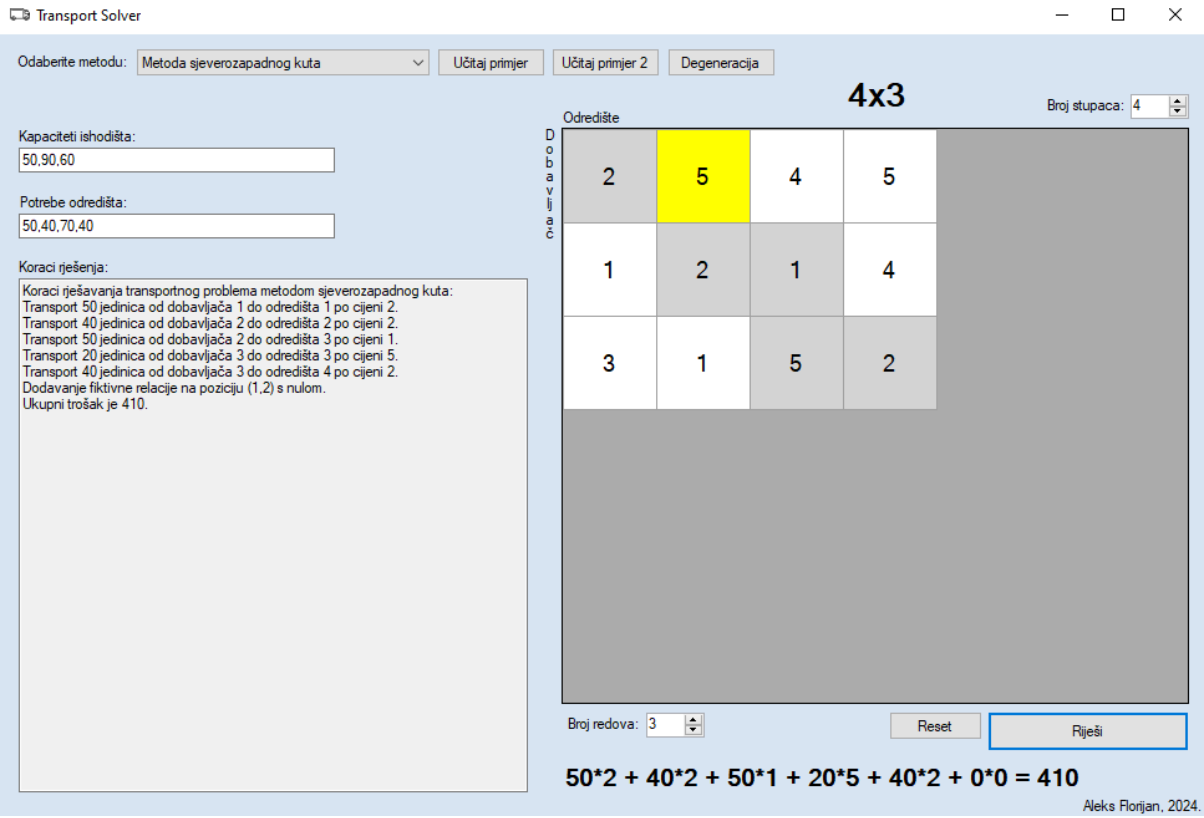
- **Veličina tablice**: Korisnici mogu mijenjati veličinu tablice prema svojim potrebama, što omogućava fleksibilnost u radu s različitim veličinama problema. Tablica se automatski prilagođava odabranim dimenzijama.
- **Odabir metode**: Unutar *ComboBox* izbornika, korisnici mogu birati između različitih metoda rješavanja transportnog problema. Ovaj izbor omogućuje

jednostavan pristup i eksperimentiranje s različitim algoritmima kako bi se našlo optimalno rješenje.

- **Unos kapaciteta i potreba**: Polja za unos kapaciteta ishodišta i potreba odredišta nalaze se unutar sučelja, omogućavajući korisnicima da precizno definiraju svoje transportne parametre.
- **Gumbi za učitavanje primjera i degeneraciju**: Sučelje sadrži gumbe *"Učitaj primjer 1"*, *"Učitaj primjer 2"* i *"Degeneracija"*, koji automatski postavljaju sve potrebne parametre programa prema unaprijed definiranim primjerima. Ovi gumbi omogućuju brzo demonstriranje funkcionalnosti programa bez potrebe za ručnim unosom svih podataka.
- **Gumb za rješavanje i resetiranje**: Gumb *"Riješi"* pokreće algoritam za rješavanje transportnog problema i prikazuje rješenje ispod gumba, kod oznake *"Z"*. Gumb *"Reset"* vraća sve postavke na početne vrijednosti, omogućujući korisniku da počne od nule.
- **Ispis koraka rješavanja**: Unutar velikog *TextBox* polja na lijevoj strani prikazuju se svi koraci koje algoritam prolazi tijekom rješavanja problema. Ova funkcionalnost omogućuje korisnicima da prate proces i razumiju kako se dolazi do konačnog rješenja.
- **Bojanje matrice rješenja**: Nakon što je problem riješen, matrica unutar *DataGridView* polja automatski se oboji kako bi se vizualno prikazalo optimalno rješenje. Ova vizualizacija pomaže korisnicima da brzo uoče ključne elemente rješenja.

Ovakav dizajn korisničkog sučelja omogućava korisnicima, bez obzira na njihovu razinu tehničkog znanja, da lako koriste program i eksperimentiraju s različitim scenarijima transportnih problema.

Korisničko sučelje programa je također responzivno i rastezljivo, što znači da se automatski prilagođava različitim veličinama ekrana i prozora. Bez obzira na rezoluciju ili veličinu prozora, svi elementi sučelja ostaju pregledni i funkcionalni.



Slika 2: Korisničko sučelje programa

7.3 Forma FrmMain.cs

Ovaj dio objašnjava implementaciju glavne forme `FrmMain.cs`, koja upravlja korisničkim sučeljem aplikacije za rješavanje transportnih problema. Program omogućuje korisniku odabir metode za rješavanje transportnog problema, unos podataka, prikaz koraka rješenja, te pregled rezultata. U nastavku su detaljno opisane glavne funkcionalnosti implementirane u `FrmMain.cs`.

Metoda `FrmMain_Load()` se poziva prilikom učitavanja forme. Ova metoda odgovorna je za postavljanje početnih vrijednosti:

```
private void FrmMain_Load(object sender, EventArgs e)
{
FillCmbMethod();
SetupDgvMatrix();
}
```

Prvo se poziva funkcija FillCmbMethod(), koja popunjava ComboBox s dostupnim metodama rješavanja transportnog problema:

```
private void FillCmbMethod()
{
    cmbMethod.Items.Add("Metoda sjeverozapadnog kuta");
    cmbMethod.Items.Add("Metoda minimalnih troškova");
    cmbMethod.Items.Add("Vogel-ova metoda");
    cmbMethod.Items.Add("MODI Metoda");

    cmbMethod.SelectedItem = "Metoda sjeverozapadnog kuta";
}
```

Ovdje se sve metode za rješavanje transportnog problema dodaju u ComboBox, a kao zadana postavljena je "Metoda sjeverozapadnog kuta".

Nakon toga, metoda SetupDgvMatrix() postavlja početnu veličinu matrice unutar DataGridView komponente:

```
private void SetupDgvMatrix()
{
    dgvMatrix.RowCount = 3;
    dgvMatrix.ColumnCount = 3;

    SetupDgvMatrixCellSize();
}
```

Ova metoda postavlja broj redaka i stupaca u DataGridView na tri. Nakon toga, poziva se metoda SetupDgvMatrixCellSize(), koja prilagođava veličinu ćelija matrice na zadane dimenzije:

```
private void SetupDgvMatrixCellSize()
{
    int desiredWidth = 75;
    int desiredHeight = 75;

    foreach (DataGridViewColumn column in dgvMatrix.Columns)
    {
        column.Width = desiredWidth;
    }

    foreach (DataGridViewRow row in dgvMatrix.Rows)
    {
        row.Height = desiredHeight;
    }
}
```

Veličina svake ćelije postavlja se na 75 piksela po širini i visini, što omogućuje jasnu vidljivost sadržaja unutar ćelija matrice.

U slučaju promjene broja redaka ili stupaca, metode nudRows_ValueChanged i nudColumns_ValueChanged prilagođavaju broj redaka i stupaca te ažuriraju veličinu ćelija i oznaku koja prikazuje dimenzije matrice:

```
private void nudRows_ValueChanged(object sender, EventArgs e)
{
    dgvMatrix.RowCount = (int)nudRows.Value;
    SetupDgvMatrixCellSize();
    UpdateMatrixSizeLabel(dgvMatrix.RowCount, dgvMatrix.ColumnCount);
    ResetResultLabel();
    ResetDgvMatrixCellColors();
    txtSolutionSteps.Clear();
}

private void nudColumns_ValueChanged(object sender, EventArgs e)
{
    dgvMatrix.ColumnCount = (int)nudColumns.Value;
    SetupDgvMatrixCellSize();
    UpdateMatrixSizeLabel(dgvMatrix.RowCount, dgvMatrix.ColumnCount);
    ResetResultLabel();
    ResetDgvMatrixCellColors();
    txtSolutionSteps.Clear();
}
```

Ove metode mijenjaju broj redaka i stupaca u DataGridView prema vrijednostima unesenim u NumericUpDown kontrolama, a zatim resetiraju rezultat i osvježavaju prikaz matrice.

Kada korisnik unese podatke i odabere metodu rješavanja, pritiskom na gumb "Riješi", poziva se metoda btnSolve_Click, koja pokreće proces rješavanja problema:

```
private void btnSolve_Click(object sender, EventArgs e)
{
    if (ValidateInputs())
    {
        string result = "Z = ?";
        int rows = dgvMatrix.Rows.Count;
        int cols = dgvMatrix.Columns.Count;

        object[,] matrix = new object[rows, cols];

        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                matrix[i, j] = dgvMatrix.Rows[i].Cells[j].Value;
            }
        }

        if (cmbMethod.SelectedItem.ToString() == "Metoda sjeverozapadnog
kuta")
        {
            result =
NorthwestCornerMethod.NorthWestCornerCalculator(matrix,
txtOutputCapacity.Text, txtDestinationNeeds.Text, txtSolutionSteps,
dgvMatrix);
```

```
        } else if (cmbMethod.SelectedItem.ToString() == "Metoda minimalnih
troškova")
        {
            result = MinimumCostMethod.MinimumCostCalculator(matrix,
txtOutputCapacity.Text, txtDestinationNeeds.Text, txtSolutionSteps,
dgvMatrix);
        } else if (cmbMethod.SelectedItem.ToString() == "Vogel-ova
metoda")
        {
            result = VogelMethod.VogelCalculator(matrix,
txtOutputCapacity.Text, txtDestinationNeeds.Text, txtSolutionSteps,
dgvMatrix);
        }

        lblResult.Text = result;
        dgvMatrix.ClearSelection();
    }
}
```

Ova metoda prvo provjerava jesu li svi potrebni podaci ispravno uneseni pozivom funkcije ValidateInputs. Ako su podaci valjani, kreira se dvodimenzionalni niz objekata matrix koji sadrži vrijednosti iz DataGridView. Zatim se na temelju odabrane metode rješavanja iz ComboBox-a, poziva odgovarajuća metoda za izračun rješenja. Nakon izračuna, rezultat se prikazuje u Label komponenti lblResult, dok se DataGridView osvježava kako bi prikazao sve korake rješenja.

Metoda ValidateInputs() provjerava ispravnost unosa korisnika:

```
private bool ValidateInputs()
{
    if (string.IsNullOrWhiteSpace(txtDestinationNeeds.Text) ||
string.IsNullOrWhiteSpace(txtOutputCapacity.Text))
    {
        MessageBox.Show("Polja za potrebe odredišta i kapacitet dobavljača
moraju biti popunjena.", "Pogreška unosa", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        return false;
    }

    string uzorak = @"^[0-9,\s]+$";
    if (!Regex.IsMatch(txtDestinationNeeds.Text, uzorak) ||
!Regex.IsMatch(txtOutputCapacity.Text, uzorak))
    {
        MessageBox.Show("Unosi moraju sadržavati samo brojeve i zareze.",
"Pogreška unosa", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return false;
    }

    int[] kapaciteti =
txtOutputCapacity.Text.Split(',').Select(int.Parse).ToArray();
    int[] potrebe =
txtDestinationNeeds.Text.Split(',').Select(int.Parse).ToArray();

    if (kapaciteti.Length != dgvMatrix.RowCount)
    {
```

```
            MessageBox.Show("Broj redaka u matrici mora odgovarati broju
        dobavljača (kapaciteta).", "Pogreška unosa", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
            return false;
        }

        if (potrebe.Length != dgvMatrix.ColumnCount)
        {
            MessageBox.Show("Broj stupaca u matrici mora odgovarati broju
        odredišta (potreba).", "Pogreška unosa", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
            return false;
        }

        for (int i = 0; i < dgvMatrix.RowCount; i++)
        {
            for (int j = 0; j < dgvMatrix.ColumnCount; j++)
            {
                if (dgvMatrix.Rows[i].Cells[j].Value == null ||
        !int.TryParse(dgvMatrix.Rows[i].Cells[j].Value.ToString(), out _))
                {
                    MessageBox.Show($"Ćelija u redu {i + 1} i stupcu {j + 1}
        nije ispravno popunjena.", "Pogreška unosa", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
                    return false;
                }
            }
        }

        return true;
    }
```

Ova funkcija provjerava jesu li polja za unos kapaciteta i potreba popunjena, jesu li uneseni isključivo brojevi i zarezi te jesu li dimenzije matrice u skladu s unosima kapaciteta i potreba. Ako bilo koji od ovih uvjeta nije ispunjen, korisniku se prikazuje odgovarajuća poruka o pogrešci.

Kada korisnik želi resetirati unos i postavke, koristi se metoda btnReset_Click:

```
    private void btnReset_Click(object sender, EventArgs e)
    {
        dgvMatrix.Rows.Clear();
        dgvMatrix.Columns.Clear();

        nudRows.Value = 0;
        nudColumns.Value = 0;

        txtOutputCapacity.Clear();
        txtDestinationNeeds.Clear();

        SetupDgvMatrixCellSize();
        ResetDgvMatrixCellColors();
    }
```

Neke metode nisu ključne za razumijevanje glavne logike programa, zbog čega nije potrebno detaljno prikazivati njihov kod. One služe prvenstveno za olakšavanje

interakcije korisnika s aplikacijom i osiguravaju pravilno funkcioniranje korisničkog sučelja.

7.4 Klasa NorthwestCornerMethod.cs

Ova metoda koristi jednostavan algoritam za dodjelu resursa i transporta od dobavljača do odredišta. Kroz ovu metodu dodjeljuju se resursi počevši od gornjeg lijevog (sjeverozapadnog) kuta matrice.

Ovdje započinje definicija statičke metode NorthWestMethodCalculator, koja se koristi za rješavanje transportnog problema koristeći Vogel-ovu aproksimativnu metodu. Metoda prima nekoliko parametara: matrica, izlazniKapacitet, potrebeOdredista, txtKoraciRjesavanja i dgvMatrix.

```
public static string NorthWestCornerCalculator(object[,] matrix, string
outputCapacity, string destinationNeeds, TextBox txtSolutionSteps,
DataGridView dgvMatrix)
```

Parametar matrica predstavlja troškovnu matricu koja opisuje troškove transporta između izvora i odredišta. Tip object[,] omogućava da svaki element matrice bude opći tip objekta, za fleksibilnost radom 2D polja. Parametar izlazniKapacitet je string koji sadrži kapacitete svih izvora, razdvojenih zarezima, dok potrebeOdredista predstavlja string koji sadrži potrebe svih odredišta, također razdvojene zarezima.

Parametar txtKoraciRjesavanja je TextBox kontrola u koju će se unositi tekstualni prikaz koraka rješavanja, omogućavajući korisniku uvid u postupak rješavanja problema. Parametar dgvMatrix je DataGridView kontrola koja služi za prikaz i ažuriranje podataka unutar matrice tijekom izvršavanja algoritma.

Ovakvu strukturu zadržava i svaka sljedeća metoda, radi lakše interakije sa glavnom formom, te pozivom tih metoda.

Na početku metode, podaci koji su zadani kao stringovi (outputCapacity i destinationNeeds) pretvaraju se u polja cijelih brojeva koja predstavljaju kapacitete dobavljača i potrebe odredišta. Također, inicijaliziraju se indeksi (i,j) koji predstavljaju trenutnu poziciju u matrici.

```
int[] kapaciteti = Array.ConvertAll(outputCapacity.Split(','), int.Parse);
```

```
int[] potrebe = Array.ConvertAll(destinationNeeds.Split(','), int.Parse);

int i = 0, j = 0;
```

Dalje, Provjerava se jesu li ukupni kapaciteti jednaki ukupnim potrebama. Ako nisu, rješenje nije moguće, pa se vraća poruka o grešci.

```
if (kapaciteti.Sum() != potrebe.Sum())
{
    return "Suma kapaciteta mora biti jednaka sumi potreba.";
}
```

Koriste se varijable za pohranu ukupnog troška (ukupniTrosak) i za pohranu rezultata rješavanja (result). Također, varijabla zauzetaPolja prati broj zauzetih polja u matrici, što je važno za kasniju provjeru degeneracije.

```
StringBuilder result = new StringBuilder();
int ukupniTrosak = 0;

int rang = kapaciteti.Length + potrebe.Length - 1; // r = m + n - 1
int zauzetaPolja = 0;
```

Metoda počinje od gornjeg lijevog kuta matrice (matrix[0, 0]) i postepeno dodjeljuje resurse i transport dok se ne ispune sve potrebe i kapaciteti.

- **Količina koja se transportira (quantity)**: Određuje se kao minimum između trenutnog kapaciteta i potrebe, jer ne možemo transportirati više nego što imamo kapaciteta ili potrebe.
- **Trošak (cost):** Dohvaća se iz matrice troškova.
- **Ažuriranje matrice i varijabli**: Nakon što se transportira određena količina, kapacitet dobavljača i potreba odredišta se umanjuju za tu količinu. Također, ukupni trošak se povećava za iznos koji predstavlja trošak za tu količinu.

```
while (i < kapaciteti.Length && j < potrebe.Length)
{
    int quantity = Math.Min(kapaciteti[i], potrebe[j]); // Količina koja se
može transportirati.
    int cost = Convert.ToInt32(matrix[i, j]); // Trošak za tu količinu.

    result.Append($"{quantity}*{cost} + "); // Dodaje korak u rezultat.

    txtSolutionSteps.AppendText($"Transport {quantity} jedinica od dobavljača
{i + 1} do odredišta {j + 1} po cijeni {cost}.\n");

    ukupniTrosak += quantity * cost; // Ažurira ukupni trošak.

    kapaciteti[i] -= quantity; // Smanjuje kapacitet za transportiranu
količinu.
```

```
        potrebe[j] -= quantity; // Smanjuje potrebu za transportiranu količinu.

        dgvMatrix.Rows[i].Cells[j].Style.BackColor = Color.LightGray; // Označava
    ćeliju kao obrađenu.

        zauzetaPolja++;

        // Ažurira indekse i ili j, ovisno o tome je li iscrpljen kapacitet ili
    potreba.
        if (kapaciteti[i] == 0) i++;
        if (potrebe[j] == 0) j++;
    }
```

Nakon što se popune sva polja, algoritam provjerava postoji li degeneracija, tj. je li broj zauzetih polja manji od ranga. Ako je to slučaj, poziva se metoda SolveDegeneration koja dodaje fiktivne alokacije. Ta klasa će biti objašnjena kasnije.

```
Degeneration.SolveDegeneration(dgvMatrix, ref zauzetaPolja, rang,
txtSolutionSteps, result);
```

Na kraju, uklanja se zadnji nepotrebni " + " iz rezultata i prikazuje se ukupni trošak.

```
if (result.Length > 3)
{
    result.Length -= 3;
}
result.Append($" = {ukupniTrosak}");

txtSolutionSteps.AppendText($"Ukupni trošak je {ukupniTrosak}.\n");

return result.ToString();
```

To je jednostavan, ali ne nužno optimalan algoritam, jer ne uzima u obzir minimizaciju troškova kod svakog koraka. Algoritam je dobar za početno rješenje, koje se kasnije može poboljšati drugim metodama.

7.5 Klasa MinimalCostMethod.cs

Ovaj algoritam nalazi početno rješenje tako da se minimalizira trošak prijevoza na svakom koraku. Za razliku od metode Sjeverozapadnog kuta, metoda minimalnih troškova aktivno traži najjeftinije dostupne rute između dobavljača i odredišta kako bi smanjila ukupni trošak transporta.

Na početku metode, ulazni stringovi outputCapacity i destinationNeeds se pretvaraju u cijelobrojna polja koja predstavljaju kapacitete dobavljača i potrebe odredišta.

Dodatno, dohvaćaju se broj redaka i stupaca iz matrice troškova (matrix), te provjera sume kao i kod prijašnje metode.

```
int[] kapaciteti = Array.ConvertAll(outputCapacity.Split(','), int.Parse);
int[] potrebe = Array.ConvertAll(destinationNeeds.Split(','), int.Parse);

int brojRedaka = matrix.GetLength(0);
int brojStupaca = matrix.GetLength(1);

if (kapaciteti.Sum() != potrebe.Sum())
{
    return "Suma kapaciteta mora biti jednaka sumi potreba.";
}
```

Inicijalizira se rješenje (rjesenje) kao matrica koja pohranjuje količine koje će biti transportirane. Također se inicijaliziraju kopije kapaciteta i potreba (preostaliKapaciteti i preostalePotrebe), kao i varijabla ukupniTrosak koja pohranjuje ukupni trošak transporta.

```
int[,] rjesenje = new int[brojRedaka, brojStupaca];
int[] preostaliKapaciteti = (int[])kapaciteti.Clone();
int[] preostalePotrebe = (int[])potrebe.Clone();
int ukupniTrosak = 0;
```

Algoritam sada prolazi kroz petlju gdje traži polje u matrici s minimalnim troškom koje ima još dostupnog kapaciteta i potreba. Kada se pronađe takvo polje, određuje se količina koja će biti transportirana (najmanja vrijednost između preostalog kapaciteta i potrebe) i ažuriraju se matrica rješenja, preostali kapaciteti, i preostale potrebe.

Sljedeći dijelovi koda se nalaze unutar while petlje:

```
while (Array.Exists(preostaliKapaciteti, c => c > 0) &&
Array.Exists(preostalePotrebe, d => d > 0))
```

Za svaki redak i stupac u matrici troškova, algoritam provjerava je li to polje ima manji trošak od trenutnog minimalnog i ima li dovoljno preostalog kapaciteta i potrebe.

```
int minTrosak = int.MaxValue;
int minRedak = -1;
int minStupac = -1;

for (int i = 0; i < brojRedaka; i++)
{
    for (int j = 0; j < brojStupaca; j++)
    {
        int trosak = Convert.ToInt32(matrix[i, j]);
        if (trosak < minTrosak && preostaliKapaciteti[i] > 0 &&
preostalePotrebe[j] > 0)
        {
            minTrosak = trosak;
            minRedak = i;
```

```
                minStupac = j;
            }
        }
    }
```

Količina koja se može transportirati određuje se kao minimum između preostalog kapaciteta u dobavljaču i preostale potrebe u odredištu. Zatim se ažuriraju rješenje i preostale količine.

```
int kolicina = Math.Min(preostaliKapaciteti[minRedak],
preostalePotrebe[minStupac]);

rjesenje[minRedak, minStupac] = kolicina;
preostaliKapaciteti[minRedak] -= kolicina;
preostalePotrebe[minStupac] -= kolicina;
ukupniTrosak += kolicina * minTrosak;
```

Dodaje se opis koraka u rješavanju problema u txtSolutionSteps, označava se odgovarajuće polje u DataGridView (tablici) kao obrađeno i povećava se broj zauzetih polja.

```
result.Append($"{kolicina}*{minTrosak} + ");
txtSolutionSteps.AppendText($"Prevezeno {kolicina} jedinica iz {minRedak +
1}. ishodišta u {minStupac + 1}. odredište po trošku {minTrosak}.\n");
txtSolutionSteps.AppendText(Environment.NewLine);

dgvMatrix.Rows[minRedak].Cells[minStupac].Style.BackColor =
System.Drawing.Color.LightGray;
zauzetaPolja++;
```

Ako postoji degeneracija (broj zauzetih polja je manji od ranga), poziva se metoda SolveDegeneration da se dodaju fiktivne alokacije.

```
Degeneration.SolveDegeneration(dgvMatrix, ref zauzetaPolja, rang,
txtSolutionSteps, result);
```

Na kraju se briše nepotreban " + " iz niza 'result' i dodaje se ukupni trošak. Rješenje se zatim vraća kao string.

```
if (result.Length > 3)
{
    result.Length -= 3;
}
result.Append($" = {ukupniTrosak}");
txtSolutionSteps.AppendText($"Ukupni trošak je {ukupniTrosak}.\n");

return result.ToString();
```

7.6 Klasa VogelMethod.cs

Cilj ove metode je minimizirati ukupni trošak transporta kod svakog koraka alociranjem količina na temelju tzv. "penala", s nižim troškovima u usporedbi s jednostavnijim metodama poput Sjeverozapadnog kuta ili metode minimalnih troškova.

Prvo, ulazni stringovi izlazniKapacitet i potrebeOdredista se pomoću Array.ConvertAll metode konvertiraju u integer nizove ‘kapaciteti' i 'potrebe' koji sadrže numeričke vrijednosti kapaciteta izvora i potreba odredišta.

```
int[] kapaciteti = Array.ConvertAll(izlazniKapacitet.Split(','),
int.Parse);
int[] potrebe = Array.ConvertAll(potrebeOdredista.Split(','), int.Parse);
```

Zatim se utvrđuje broj redaka (brojRedaka) i broj stupaca (brojStupaca) u matrici, pomoću GetLength metode. Ove varijable predstavljaju broj izvora i broj odredišta u transportnom problemu.

```
int brojRedaka = matrica.GetLength(0);
int brojStupaca = matrica.GetLength(1);
```

Nakon toga, provjerava se da li su ukupni kapaciteti jednaki ukupnim potrebama. Ta provjera osigurava da je problem uravnotežen, što je preduvjet za primjenu Vogelove metode. Ukoliko ukupni kapaciteti i potrebe nisu jednaki, metoda vraća poruku o grešci.

```
if (kapaciteti.Sum() != potrebe.Sum())
{
    return "Suma kapaciteta mora biti jednaka sumi potreba.";
}
```

Ako su kapaciteti i potrebe uravnoteženi, metoda nastavlja s inicijalizacijom varijabli za praćenje ukupnog troška (ukupniTrosak) i za pohranu teksta koji opisuje korake rješavanja (rezultat). TextBox kontrola txtKoraciRjesavanja se čisti kako bi se pripremila za unos novih koraka.

```
int ukupniTrosak = 0;
StringBuilder rezultat = new StringBuilder();
txtKoraciRjesavanja.Clear();
txtKoraciRjesavanja.AppendText("Koraci rješavanja transportnog problema
Vogelovom metodom:\r\n");
```

Metoda zatim određuje rang problema, koji se definira kao zbroj broja izvora i broja odredišta umanjen za jedan. Također, inicijalizira se broj zauzetih polja na 0, što će biti važno u kasnijem dijelu metode:

```
int rang = kapaciteti.Length + potrebe.Length - 1; // r = m + n - 1
int zauzetaPolja = 0;
```

Nakon toga slijedi glavni dio metode, gdje se u petlji while iterira dok god postoje neispunjeni kapaciteti i potrebe:

```
while (kapaciteti.Sum() > 0 && potrebe.Sum() > 0)
```

Unutar petlje, najprije se računa penal za svaki redak. Penali predstavljaju razliku između najmanjeg i drugog najmanjeg troška unutar aktivnih stupaca za red. To se postiže iteracijom kroz svaki redak i izračunom troška za aktivne stupce (tj. stupce s neispunjenim potrebama):

```
int[] redovniPenali = new int[brojRedaka];
for (int i = 0; i < brojRedaka; i++)
{
    var aktivneKolone = Enumerable.Range(0, brojStupaca).Where(j => potrebe[j] >
0).ToArray();
    if (aktivneKolone.Length >= 2)
    {
        int min1 = int.MaxValue, min2 = int.MaxValue;
        foreach (var j in aktivneKolone)
        {
            int trenutniTrosak = Convert.ToInt32(matrica[i, j]);
            if (trenutniTrosak < min1)
            {
                min2 = min1;
                min1 = trenutniTrosak;
            } else if (trenutniTrosak < min2)
            {
                min2 = trenutniTrosak;
            }
        }
        redovniPenali[i] = min2 - min1;
    }
}
```

Još k tome, izračunava se penal i za svaki stupac, računajući razliku između najmanjeg i drugog najmanjeg troška unutar aktivnih redaka (tj. redaka s neispunjenim kapacitetima):

```
int[] stupcaniPenali = new int[brojStupaca];
for (int j = 0; j < brojStupaca; j++)
{
    var aktivniRedci = Enumerable.Range(0, brojRedaka).Where(i =>
kapaciteti[i] > 0).ToArray();
    if (aktivniRedci.Length >= 2)
    {
        int min1 = int.MaxValue, min2 = int.MaxValue;
        foreach (var i in aktivniRedci)
        {
            int trenutniTrosak = Convert.ToInt32(matrica[i, j]);
            if (trenutniTrosak < min1)
            {
                min2 = min1;
```

```
                min1 = trenutniTrosak;
            } else if (trenutniTrosak < min2)
            {
                min2 = trenutniTrosak;
            }
        }
        stupcaniPenali[j] = min2 - min1;
    }
}
```

Nakon što su penali izračunati, pronalazi se maksimalni penal između svih redaka i stupaca. Ovo osigurava da se resursi prvo alociraju tamo gdje je najveća razlika između dva najjeftinija troška, čime se minimizira ukupni trošak:

```
int maxPenal = -1;
int odabraniRedak = -1, odabraniStupac = -1;

for (int i = 0; i < brojRedaka; i++)
{
    if (kapaciteti[i] > 0 && redovniPenali[i] > maxPenal)
    {
        maxPenal = redovniPenali[i];
        odabraniRedak = i;
        odabraniStupac = -1;
    }
}

for (int j = 0; j < brojStupaca; j++)
{
    if (potrebe[j] > 0 && stupcaniPenali[j] > maxPenal)
    {
        maxPenal = stupcaniPenali[j];
        odabraniRedak = -1;
        odabraniStupac = j;
    }
}
```

Ako niti jedan redak ni stupac nije odabran (što je signalizirano vrijednostima -1), petlja se prekida jer su svi kapaciteti i potrebe zadovoljeni:

```
if (odabraniRedak == -1 && odabraniStupac == -1)

{
    break;
}
```

Ako je odabran redak, pronalazi se stupac s najmanjim troškom unutar tog retka, dok se u suprotnom traži redak s najmanjim troškom unutar odabranog stupca:

```
if (odabraniRedak != -1)
{
    odabraniStupac = Enumerable.Range(0, brojStupaca).Where(j =>
potrebe[j] > 0).OrderBy(j => Convert.ToInt32(matrica[odabraniRedak,
j])).First();
```

```
} else
{
    odabraniRedak = Enumerable.Range(0, brojRedaka).Where(i =>
kapaciteti[i] > 0).OrderBy(i => Convert.ToInt32(matrica[i,
odabraniStupac])).First();
}
```

Nakon toga, alocira se minimalna količina između kapaciteta odabranog retka i potrebe odabranog stupca. Ova količina se zatim oduzima iz kapaciteta i potreba, dok se odgovarajući trošak dodaje na ukupni trošak:

```
int kolicina = Math.Min(kapaciteti[odabraniRedak],
potrebe[odabraniStupac]);
int trosak = Convert.ToInt32(matrica[odabraniRedak, odabraniStupac]);

kapaciteti[odabraniRedak] -= kolicina;
potrebe[odabraniStupac] -= kolicina;
ukupniTrosak += kolicina * trosak;
```

Informacije o alokaciji se upisuju u varijablu rezultat i prikazuju u txtKoraciRjesavanja, dok se odgovarajuća ćelija u dgvMatrix oboji kako bi se vizualno označilo koje su kombinacije izvor-odredište korištene u rješenju. Ova boja služi kao pokazatelj zauzetih polja u matrci:

```
rezultat.Append($"{kolicina}*{trosak} + ");
txtKoraciRjesavanja.AppendText($"Transport {kolicina} jedinica od
dobavljača {odabraniRedak + 1} do odredišta {odabraniStupac + 1} po cijeni
{trosak}.\n");
txtKoraciRjesavanja.AppendText(Environment.NewLine);

dgvMatrix.Rows[odabraniRedak].Cells[odabraniStupac].Style.BackColor =
System.Drawing.Color.LightGray;
zauzetaPolja++;
```

Ova petlja while nastavlja se sve dok postoje kapaciteti i potrebe koje treba zadovoljiti. Svaka iteracija petlje osigurava da se resursi alociraju na način koji minimizira ukupni trošak transporta. Nakon toga je provjera degeneracije kao i u prijašnjim metodama.

```
Degeneration.SolveDegeneration(dgvMatrix, ref zauzetaPolja, rang,
txtKoraciRjesavanja, rezultat);
```

Te na kraju brisanje zadnjeg znaka + i vraćanje prikaza rješenja.

7.6 Klasa Degeneration.cs

Definicija metode SolveDegeneration započinje sljedećom deklaracijom:

```
public static void SolveDegeneration(DataGridView dgvMatrix, ref int
zauzetaPolja, int rang, TextBox txtSolutionSteps, StringBuilder result)
```

Ova metoda prima sljedeće parametre:

- **dgvMatrix**: DataGridView koji prikazuje transportnu matricu
- **zauzetaPolja**: referenca na broj zauzetih polja u matrici
- **rang**: rang transportnog problema, izračunat kao m + n - 1 (broj redaka plus broj stupaca minus jedan)
- **txtSolutionSteps**: TextBox u kojem se prikazuju koraci rješavanja
- **result**: StringBuilder objekt koji sadrži tekstualni prikaz rješenja.

Metoda SolveDegeneration implementira petlju while koja se izvršava sve dok broj zauzetih polja nije jednak rangu problema:

```
while (zauzetaPolja < rang)
```

Unutar ove petlje, metoda pokušava dodati fiktivnu relaciju u matricu kako bi povećala broj zauzetih polja. Za svaku iteraciju koristi se varijabla dodanaFiktivnaRelacija koja prati je li fiktivna relacija uspješno dodana:

```
bool dodanaFiktivnaRelacija = false;
```

Metoda koristi dvije ugniježđene petlje 'for' koje iteriraju kroz sve ćelije matrice (dgvMatrix), provjeravajući svaku ćeliju kako bi našle prvu slobodnu (tj. nezauzetu) ćeliju:

```
for (int x = 0; x < dgvMatrix.RowCount; x++)
{
    for (int y = 0; y < dgvMatrix.ColumnCount; y++)
    {
        if (dgvMatrix.Rows[x].Cells[y].Style.BackColor != Color.LightGray)
```

Ako se pronađe nezauzeta ćelija (tj. ona koja nije označena bojom LightGray), ta se ćelija označava bojom Yellow, čime se pokazuje da je u nju dodana fiktivna relacija:

```
{
    dgvMatrix.Rows[x].Cells[y].Style.BackColor = Color.Yellow;
    zauzetaPolja++;
    dodanaFiktivnaRelacija = true;
    result.Append("0*0 + ");
    txtSolutionSteps.AppendText($"Dodavanje fiktivne relacije na poziciju
({x + 1},{y + 1}) s nulom.\n");
    txtSolutionSteps.AppendText(Environment.NewLine);
    break;
}
```

Dodavanjem fiktivne relacije u ćeliju, ažurira se broj zauzetih polja (zauzetaPolja++) te se u varijabli dodanaFiktivnaRelacija označava da je fiktivna relacija dodana. Tekstualni

prikaz ovog koraka ažurira se dodavanjem nule u StringBuilder objekt 'result', dok se u TextBox txtSolutionSteps bilježi detaljan opis koraka rješavanja.

Nakon dodavanja fiktivne relacije, metoda provjerava vrijednost varijable dodanaFiktivnaRelacija kako bi odlučila hoće li nastaviti pretraživati matricu za daljnje dodavanje relacija ili izaći iz petlje:

```
if (dodanaFiktivnaRelacija)
{
    break;
}
```

Ova se procedura ponavlja sve dok broj zauzetih polja ne postane jednak rangu problema.

Na kraju, metoda SolveDegeneration ne vraća nikakvu vrijednost (void), već svoje rezultate prikazuje kroz ažurirane parametre dgvMatrix, txtSolutionSteps, i result. Ovaj postupak omogućuje stabilno rješavanje transportnih problema, čak i u situacijama kada se pojavi degeneracija.

8. Zaključak

Provedena analiza metoda rješavanja transportnih problema pokazali su da svaka metoda ima svoje prednosti i ograničenja ovisno o specifičnostima problema i ciljevima optimizacije. Metoda sjeverozapadnog kuta i metoda minimalnih troškova pružaju brz i jednostavan način za dobivanje početnog rješenja, dok Vogel-ova aproksimacijska metoda i MODI metoda omogućuju dodatnu optimizaciju i smanjenje ukupnih troškova transporta.

Razvijeni softver u okviru ovog rada, temeljen na programskom jeziku C# i Windows Forms tehnologiji, predstavlja konkretan alat koji integrira neke od ovih metoda i omogućuje korisnicima interaktivno i intuitivno rješavanje transportnih problema. Kroz praktičnu primjenu softverskog rješenja, korisnici mogu brzo analizirati različite scenarije i prilagoditi transportne planove prema promjenjivim uvjetima tržišta i operativnim potrebama.

Jedna od ključnih prednosti razvijenog rješenja je i njegova fleksibilnost kod degeneriranih slučajeva. Degeneracija, koja može dovesti do nepreciznih rezultata ili čak nemogućnosti dobivanja rješenja, uspješno je riješena kroz algoritme koji su integrirani u aplikaciju.

Popis literature

[1] Fakultet organizacije i informatike, »PREDAVANJE - Formulacija transportnog problema i metode početnog rasporeda tereta. | Akademska godina 2022./2023.« [Na internetu]. Dostupno: https://elfarchive2223.foi.hr/mod/resource/view.php?id=76386 [Pristupano: 1.8.2024.].

[2] Fakultet organizacije i informatike, »SEMINAR - Transportni problem - Metode početnog raspoređivanja tereta | Akademska godina 2022./2023.« [Na internetu]. Dostupno: https://elfarchive2223.foi.hr/mod/resource/view.php?id=76392 [Pristupano: 1.8.2024.].

[3] Carlin Alvin, »Metode rješavanja transportnog problema« 2021-09-27. [Na internetu]. Dostupno: https://dabar.srce.hr/islandora/object/foi%3A6794 [Pristupano: 3.8.2024.].

[4] Fakultet organizacije i informatike, »SEMINAR - Webinar - Transportni problem - Metode početnog raspoređivanja tereta | Akademska godina 2022./2023.« [Na internetu]. Dostupno: https://elfarchive2223.foi.hr/mod/resource/view.php?id=76389 [Pristupano: 8.8.2024.].

[5] Fakultet organizacije i informatike, »SEMINAR - Transportni problem - Metode za traženje optimalnog rješenja_MODI metoda | Akademska godina 2022./2023.« [Na internetu]. Dostupno: https://elfarchive2223.foi.hr/mod/resource/view.php?id=109741 [Pristupano: 8.10.2024.].

[6] Fakultet organizacije i informatike, »PREDAVANJA - Transportni problem – Degeneracija | Akademska godina 2022./2023.« [Na internetu]. Dostupno: https://elfarchive2223.foi.hr/mod/resource/view.php?id=76388 [Pristupano: 8.12.2024.].

[7] Fakultet organizacije i informatike, »SEMINAR - Webinar - Transportni problem – Degeneracija | Akademska godina 2022./2023.« [Na internetu]. Dostupno: https://elfarchive2223.foi.hr/mod/resource/view.php?id=76391 [Pristupano: 8.12.2024.].

[8] OpenAI, »ChatGPT: AI jezični model – Korišteno za pomoć u otklanjanju grešaka u programskom kodu i razjašnjavanje nejasnih dijelova teme rada,« 2024. [Pristupano: 1.8.2024. - 23.8.2024.].

Popis tablica i slika