

Praktické aplikace optimalizačních metod: příklady a řešení

Nikita Yalynnyi

Bakalářská práce
2025



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	Nikita Yalynnyi
Osobní číslo:	A22423
Studijní program:	B0613A140020 Softwarové inženýrství
Forma studia:	Prezenční
Téma práce:	Praktické aplikace optimalizačních metod: příklady a řešení
Téma práce anglicky:	Practical Applications of Optimization Methods: Examples and Solutions

Zásady pro vypracování

- Definujte použité optimalizační či grafové problémy.
- Co nejpodrobněji popište jejich využití v praxi.
- Vytvořte sadu netriviálních řešených příkladů.
- Navrhněte alespoň jeden relativně obtížný a praktický příklad pro daný optimalizační problém.
- Praktický příklad vyřešte pomocí dostupných metod, popřípadě pomocí vlastního programu, a výsledky, v závislosti na čase výpočtu, porovnejte.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. VANDERBEI, Robert J. Linear programming: foundations and extensions. 4th ed. International series in operations research & management science. New York, N.Y.: Springer, c2014. ISBN 978-1-4614-7629-0.
2. HILLIER, Frederick S. a LIEBERMAN, Gerald J. Introduction to operations research. 9. ed. New York, NY: McGraw-Hill, 2010. ISBN 007-337629-9.
3. RARDIN, Ronald L. Optimization in operations research. Upper Saddle River: Prentice Hall, 2000. ISBN 0-02-398415-5.
4. WEST, Douglas Brent. Introduction to graph theory. Second edition. Pearson modern classic. [New York]: Pearson, [2018]. ISBN 978-0-13-143737-1.
5. HLINĚNÝ, Petr. Základy teorie grafů. Online. 1. vyd. Brno: Masarykova univerzita, 2010. Elportál. ISSN 1802-128X. Dostupné z: <http://is.muni.cz/elportal/?id=878389>.
6. STEINBACH, Bernd a POSTHOFF, Christian. Logic Functions and Equations: Examples and Exercises. Dordrecht: Springer Netherlands, 2009. Dostupné z: <https://doi.org/978-1-4020-9595-5>.
7. BAZARAA, M. S.; JARVIS, John J. a SHERALI, Hanif D. Linear programming and network flows. 4th ed. Hoboken, N.J.: John Wiley, c2010. ISBN 978-0-470-46272-0.

Vedoucí bakalářské práce: **Mgr. Zbyněk Cerman, Ph.D.**
Ústav matematiky

Datum zadání bakalářské práce: **27. října 2024**
Termín odevzdání bakalářské práce: **2. června 2025**



L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA BAKALÁŘSKÉ PRÁCE

Beru na vědomí, že

- odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení bakalářské práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – bakalářskou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na bakalářské práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.
- při psaní práce jsem využil(a) nástroj umělé inteligence (ChatGPT od OpenAI, verze GPT-4o) k formálním úpravám textu, ověřování jazykové správnosti a inspiraci při formulaci některých částí. Obsah práce však vychází z mé vlastní odborné činnosti a odpovídá mému porozumění tématu.

Ve Zlíně, dne

.....

podpis autora

ABSTRAKT

Cílem této bakalářské práce je vysvětlení praktického použití optimalizačních metod. Teoretická část definuje klíčové optimalizační problémy a popisuje jejich teoretické základy. Dále je analyzováno jejich využití v různých oblastech, jako jsou logistika, doprava, řízení zdrojů a informační technologie.

Praktická část se věnuje ukázce řešení netriviálních optimalizačních příkladů, přičemž její součástí je návrh a řešení složitějšího praktického příkladu, který je vyřešen pomocí dostupných metod a vlastního algoritmu. Výsledky jsou porovnány z hlediska náročnosti a efektivity.

Klíčová slova: optimalizační metody, grafové algoritmy, lineární programování, teorie grafů, analýza složitosti

ABSTRACT

The aim of this bachelor thesis is to explain the practical use of optimization methods and graph algorithms. The theoretical part defines the key optimization problems and describes their theoretical foundations. Furthermore, their application in different areas such as logistics, transportation, resource management and information technology is analyzed.

The practical part is devoted to the demonstration of solving non-trivial optimization examples, and includes the design and solution of a more complex practical example, which is solved using available methods and a custom algorithm. The results are compared in terms of difficulty and efficiency.

Keywords: optimization methods, graph algorithms, linear programming, graph theory, complexity analysis

Rád bych poděkoval Mgr. Zbyňkovi Cermanovi, Ph.D., za jeho cenné rady a odborné vedení, které mi poskytl během zpracování mé bakalářské práce.

Dále bych chtěl poděkovat své přítelkyni a rodičům za jejich neustálou podporu, víru ve mne a motivaci, jež mi dodávaly sílu pokračovat.

OBSAH

ÚVOD.....	11
I TEORETICKÁ ČÁST	12
1 POUŽITÉ OPTIMALIZAČNÍ PROBLEMY	13
1.1 PŘÍRAZOVACÍ ÚLOHA	13
1.1.1 Matematická definice	13
1.1.2 Algoritmy řešení.....	14
1.1.3 Mackův algoritmus.....	14
1.1.4 Maďarská metoda.....	15
1.2 DOPRAVNÍ ÚLOHA	16
1.2.1 Matematická definice	17
1.2.2 Typy dopravní úlohy	18
1.2.3 Algoritmy řešení.....	18
1.2.4 MODI metoda	20
2 PRAKTICKÉ VYUŽITÍ OPTIMALIZAČNÍCH METOD	22
2.1 PŘÍRAZOVACÍ ÚLOHA	22
2.2 DOPRAVNÍ ÚLOHA	23
3 ŘEŠENÍ NETRIVIÁLNÍCH OPTIMALIZAČNÍCH ÚLOH.....	25
3.1 PŘÍRAZOVACÍ ÚLOHA	25
3.1.1 Maďarská metoda.....	25
3.1.2 Mackův algoritmus.....	31
3.2 DOPRAVNÍ PROBLÉM	37
II PRAKTICKÁ ČÁST.....	51
4 CÍL A PŘEHLED PRAKTICKÉ ČÁSTI.....	52
4.1.1 Cíl praktické části.....	52
5 NÁVRH OPTIMALIZAČNÍCH PROBLÉMŮ.....	53
5.1 PŘÍRAZOVACÍ ÚLOHA	53
5.2 DOPRAVNÍ PROBLÉM	54
6.1 PYTHON.....	56
6.1.1 Knihovny v Python	56
6.1.2 Použité knihovny.....	56
6.2 EXCEL	56
6.2.1 Excel Solver	57
6.2.2 Jak zapnout Excel Solver	57
7 IMPLEMENTACE ALGORITMŮ V PYTHON.....	59
7.1 PŘÍRAZOVACÍ ÚLOHA	59
7.1.1 Použití knihovny pro řešení přiřazovací úlohy	64

7.2	DOPRAVNÍ PROBLÉM	64
7.2.1	Použití knihovny pro řešení dopravního problému	69
8	ŘEŠENÍ PROBLÉMU POMOCÍ EXCEL.....	71
8.1	PŘÍRAZOVACÍ ÚLOHA	71
8.1.1	Příprava vstupních dat	71
8.1.2	Tvorba modelu pro Solver.....	71
8.1.3	Řešení pomocí makra	72
8.2	DOPRAVNÍ PROBLÉM	74
8.2.1	Příprava vstupních dat	74
8.2.2	Tvorba modelu pro Solver.....	74
8.2.3	Řešení pomocí makra	77
9	HODNOCENÍ VÝSLEDKU.....	81
9.1	PŘÍRAZOVACÍ ÚLOHA	81
9.2	DOPRAVNÍ PROBLÉM	82
	ZÁVĚR.....	84
	SEZNAM POUŽITÉ LITERATURY	85
	SEZNAM OBRÁZKŮ	89
	SEZNAM TABULEK.....	90
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	92
	SEZNAM PŘÍLOH.....	93

ÚVOD

Cílem této práce je demonstrace praktického využití metod optimalizace, jejich význam a všudypřítomnost. Optimalizace je klíčovým nástrojem pro řešení různých úloh ve vědě, technice a dokonce i v přírodě.

V oblasti softwarového inženýrství se s ní setkáváme prakticky na každém kroku - například při vývoji webových nebo mobilních aplikací je často cílem minimalizovat dobu odezvy systému, zrychlit vyhledávání nebo zefektivnit rozvržení dat. Dalším z výrazných příkladů optimalizačních procesů je skládání bílkovin v organismu. V tomto procesu protein nachází takovou prostorovou konfiguraci, která minimalizuje jeho volnou energii, což je zásadní pro jeho správnou funkci a interakci s jinými molekulami. Optimalizace tedy není pouze matematickou úlohou, ale také základním principem, který určuje zákony přírody. Toto uvědomění se pro mě stalo zdrojem motivace k napsání této práce, protože samotný život je skvělým příkladem optimalizace.

V první kapitole se věnujeme základním pojmům z oblasti optimalizace, které jsou klíčové pro porozumění celé problematice. Následně definujeme použité optimalizační problémy, které jsou předmětem této práce, a vysvětlíme jejich teoretické základy.

Ve druhé kapitole analyzujeme praktické využití definovaných problémů v různých oblastech.

Ve třetí kapitole sestavíme sadu netriviálních problémů a ukažeme jejich řešení. Každý z těchto problémů podrobně popíšeme a vysvětlíme jeho význam.

Poté, v praktické části, zaměříme se na návrh a řešení relativně obtížných praktických příkladů pro použité optimalizační problémy. Nejprve je definujeme a následně je vyřešíme pomocí dostupných metod i vlastního algoritmu. Nakonec porovnáme dosažené výsledky z hlediska času výpočtu.

I TEORETICKÁ ČÁST

1 POUŽITÉ OPTIMALIZAČNÍ PROBLEMY

V první kapitole jsou představeny používané optimalizační problémy, především přiřazovací úloha, která je speciálním případem dopravního problému.

1.1 Přiřazovací úloha

Přiřazovací úloha představuje důležitý nástroj operačního výzkumu, který slouží k optimálnímu rozdělení dostupných zdrojů na konkrétní činnosti. Její využití se používá v mnoha praktických oblastech, a proto je pochopení tohoto typu optimalizačního problému zásadní pro kvalitní rozhodování a řízení v praxi. [1]

Představme si, že máme x úkolů, které je třeba vykonat na y strojích. Každé přiřazení úkolu ke stroji má určitou hodnotu. Naším cílem je přiřadit jednotlivé úkoly různým strojům tak, aby každý stroj dostal právě jeden úkol a současně celkové náklady byly minimální. [2]

1.1.1 Matematická definice

Matematická definice přiřazovací úlohy je následující:

Cílem je minimalizovat:

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

Kde:

c_{ij} – náklady nebo hodnota spojená s přiřazením stroje i k úkolu j

x_{ij} – binární proměnná, která nabývá hodnoty:

$$x_{ij} = \begin{cases} 1, & \text{pokud stroj } i \text{ je přiřazen k úkolu } j \\ 0, & \text{pokud stroj } i \text{ není přiřazen k úkolu } j \end{cases}$$

Za podmínek:

a)

$$\sum_{i=1}^n x_{ij} = 1, j = 1, 2, \dots, n$$

To znamená, že každý úkol je vykonáván strojem j , $i = 1, 2, \dots, n$

b)

$$\sum_{j=1}^n x_{ij} = 1, i = 1, 2, \dots, n$$

To znamená, že právě jeden úkol je přiřazen stroji j , $j = 1, 2, \dots, n$ [2]

1.1.2 Algoritmy řešení

Algoritmy řešení přiřazovací úlohy mohou být rozděleny do tří skupin:

1. Algoritmy na základě nalezení maximálního toku

Tyto metody vycházejí z primal-dualního přístupu, kde se postupně buduje přípustné duální řešení, řeší se omezená primární úloha a iterativně se upravuje duální řešení. Příkladem je metoda Papadimitriou a Steiglitz (1982).

2. Algoritmy na základě nejkratších cest

Lineární přiřazení je zde interpretováno jako problém minimálních nákladů toku. Hlavní myšlenkou je iterativní vylepšování řešení hledáním augmentačních cest. Mezi klíčové metody patří algoritmy Tomizawy (1971) nebo metoda Hung a Rom (1980) využívající Dijkstrův algoritmus.

3. Algoritmy na základě simplexové metody

Tyto algoritmy představují úpravy klasické simplexové metody. Jsou náchylné na nulové pivoty (zero-cost steps), což vede k vysoké výpočetní složitosti a degeneraci řešení. Přesto existují efektivní implementace – například metoda Barr, Glover a Klingman (1977). [3]

Vybrané algoritmy řešení dané úlohy

Mezi širokou škálou metod se v praxi osvědčily dvě metody: Maďarský algoritmus a Mackův algoritmus. Tyto algoritmy budou dále podrobněji rozebrány, včetně jejich principů a praktického využití.

1.1.3 Mackův algoritmus

Algoritmus je založen na jednoduchých a přehledných principech. Nejprve se upraví nákladová matice tak, aby se nezměnilo její optimální řešení. Poté se hledá takové uspořádání, kdy minima v jednotlivých řádcích této upravené matice leží v různých sloupcích. Pokud tomu tak není, upravují se hodnoty v matici tak, aby bylo možné tato minima rozložit do více sloupců, čímž se usnadní nalezení optimálního řešení celého problému. [4]

Postup řešení:

1. Najděte v každém řádku nákladové matice minimální prvky a podtrhněte je. Pokud se přitom v každém sloupci matice nachází právě jeden podtržený prvek, je řešení nalezeno (tvoří ho podtržené prvky matice), jinak přejděte ke kroku 2.
2. Vyberte ze sloupců, které nejsou označeny znakem „+“ (na začátku řešení takových sloupců není), sloupec, který obsahuje více než jeden podtržený prvek, a označte ho znakem „+“.
3. Nechť prvek ve sloupci označeném znakem „+“ v řádku i je b_i a minimální prvek ve sloupci, který není označen znakem „+“ v tomto řádku, je a_i . Nechť platí

$$\min(a_i - b_i) = a'_r - b'_r$$

kde i probíhá přes všechny příslušné indexy.

4. Zvětšete všechny prvky ve sloupcích označených znakem „+“ o hodnotu $a'_r - b'_r$.
5. Označte prvek a'_r znakem „*“.
6. Prohlédněte sloupec, který obsahuje hodnotu a'_r . Pokud v něm již je podtržený prvek, označte tento sloupec znakem „+“ a přejděte ke kroku 3, jinak přejděte ke kroku 7.
7. Podtrhněte prvek a'_r a odstraňte z něj označení „*“.
8. Najděte podtržený prvek v řádku, který obsahuje a'_r . Odstraňte podtržení tohoto prvku.
9. Prohlédněte sloupec, ze kterého právě bylo odstraněno podtržení. Pokud obsahuje podtržené prvky, přejděte ke kroku 10, jinak najděte v tomto sloupci prvek označený znakem „*“, označte ho jako a'_r a přejděte ke kroku 7.
10. Pokud existují sloupce bez podtržených prvků, odstraňte všechna označení „+“ a vraťte se ke kroku 2. [4]

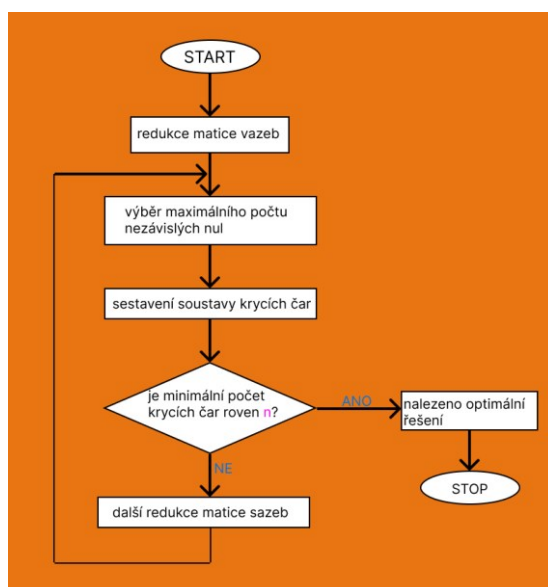
1.1.4 Maďarská metoda

Hlavní myšlenkou maďarské metody je řešit úlohu s redukovanou maticí, ve které všechny sazby zůstanou nezáporné a každý řádek a sloupec obsahuje alespoň jednu nulovou sazbu. Optimální řešení pak odpovídá přiřazení, kde kladným složkám v matici odpovídají nulové sazby.

Algoritmus:

1. Odečtete nejmenší prvek v každém řádku od všech prvků tohoto řádku.
2. Odečtete nejmenší prvek v každém sloupci od všech prvků tohoto sloupce.
3. Nakreslete čáry přes vhodné řádky a sloupce tak, aby byly pokryty všechny nulové prvky v nákladové matici a bylo použito minimální možné množství takových čar.
4. Test optimálnosti:
 - 1) Pokud je minimální počet krycích čar roven rozměru nákladové matice (např. 3 krycí čáry pro matici 3x3), je možné optimální přiřazení nul a algoritmus končí.
 - 2) Pokud je minimální počet krycích čar menší než nn , optimální přiřazení nul zatím není možné. V tom případě pokračujte na krok 5.
5. Určete nejmenší prvek, který není pokryt žádnou čarou. Odečtete tento prvek od každého nepokrytého řádku a přičtete jej ke každému pokrytému sloupci. Vraťte se ke kroku 3. [5]

Obrázek 1 Schématické zobrazení Maďarské metody



(zdroj: vlastní)

1.2 Dopravní úloha

Dopravní problém se zaměřuje na efektivní rozvoz určitého zboží z několika výchozích míst, označovaných jako zdroje, do několika cílových míst, označovaných jako odběratele. Cílem je nalézt takové rozdělení přepravních toků mezi zdroji a odběrateli, které povede k minimálním celkovým nákladům na dopravu. [6]

Vyjadření dopravního problému pomocí tabulky:

Obrázek 2 Vyajdření dopravní úlohy pomocí tabulky

Dodavatelé (zdroje)	Odběratelé (cílová místa)				Kapacity dodavatelů (zdrojů)
	Q_1	Q_2	...	Q_n	
D_1	c_{11} x_{11}	c_{12} x_{12}	...	c_{1n} x_{1n}	a_1
D_2	c_{21} x_{21}	c_{22} x_{22}	...	c_{2n} x_{2n}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
D_m	c_{m1} x_{m1}	c_{m2} x_{m2}	...	c_{mn} x_{mn}	a_m
Požadavky odběratelů (cílových míst)	b_1	b_2	...	b_n	$\sum a_i$ $\sum b_j$

(zdroj: [7])

1.2.1 Matematická definice

Minimalizovat

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

Kde:

 x_{ij} – množství zboží od dodavatele i k odběrateli j c_{ij} – přepravní náklady na jednotku zboží mezi dodavatelem i a odběratelem j

Za podmínek:

$$\sum_{j=1}^n x_{ij} = a_i, i = 1, 2, \dots, m$$

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

$$x_{ij} \geq 0, i = 1, 2, \dots, m, j = 1, 2, \dots, n \quad [8]$$

1.2.2 Typy dopravní úlohy

Jsou tři typy dopravních úloh:

- Vyrovnaný dopravní problém

Součet všech zdrojů je roven součtu všech požadavků, tedy $\sum_{i=1}^m a_i = \sum_{j=1}^n b_j$.

Tento typ DP má vždy přípustné i optimální řešení.

- Nevyrovnaný dopravní problém s přebytkem zdrojů

Součet kapacit zdrojů je větší, než součet kapacit požadavků, tedy

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

- Nevyrovnaný dopravní problém s nedostatkem zdrojů

Součet kapacit požadavků je větší, než součet kapacit zdrojů, tedy

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

Každý nevyrovnaný DP lze vždy převést na vyrovnaný DP přidáním fiktivního dodavatele nebo přidáním fiktivního odběratele. [9]

1.2.3 Algoritmy řešení

Řešení DP obvykle zahrnuje dva hlavní kroky:

1. Nalezení výchozího základního přípustného řešení
2. Zlepšování řešení až do okamžiku, kdy je nalezeno optimální řešení

Typickými metodami nalezení výchozího základního přípustného řešení jsou:

- Metoda severozápadního rohu
- Indexní metoda (metoda maticového minima)
- Vogelová aproximační metoda [10]

Metoda severozápadního rohu patří mezi nejjednodušší způsoby nalezení přípustného řešení dopravní úlohy.

1. Výpočet začíná výběrem buňky severozápadního rohu (tj. levý horní roh) tabulky a přiřazení maximálního možného počtu jednotek mezi dodávkami a poptávkami. Při přiřazování se zcela ignorují přepravní náklady (neberou se v úvahu).

2. Odstraňte řádek nebo sloupec, který již neobsahuje žádné hodnoty (zcela vyčerpán) pro dodávky nebo poptávku.
3. Nyní, s novou zkrácenou tabulkou, opět vyberte buňku severozápadního rohu a přiřaďte dostupné hodnoty.
4. Opakujte kroky 2 a 3, dokud všechny hodnoty dodávek a poptávky nejsou nulové.
5. Získejte počáteční základní přípustné řešení. [11]

Indexní metoda obvykle poskytuje řešení s nižšími přepravními náklady než předchozí metoda. Jedná se o tzv. hladový algoritmus, kdy postupně vybíráme trasy s nejnižšími náklady.

1. Vyberte buňku s nejnižšími přepravními náklady v celé tabulce a přiřaďte dodávky a poptávky.
2. Odstraňte řádek/sloupec, který byl zcela vyčerpán. Tento vymazaný řádek/sloupec již není zohledněn při dalším přiřazování.
3. Opět vyberte buňku s nejnižšími náklady v aktuální tabulce a přiřaďte hodnoty. (Poznámka: Pokud existuje více buněk s nejnižšími náklady, vyberte ty, kde je možné provést maximální přiřazení.)
4. Získejte počáteční základní přípustné řešení. [12]

Vogelova aproximační metoda je třetí heuristickou metodou a její výhoda oproti indexní metodě spočívá v tom, že v každém kroku zohledňuje "alternativní náklady" při výběru nejlevnější trasy. Pracuje s diferencemi, tedy rozdíly mezi druhou nejlepší a nejlepší variantou v každém řádku a sloupci. Volí se vždy nejlevnější trasa v řádku nebo sloupci s největší diferencí.

Kroky:

1. Vypočítejte penalizace pro každý řádek a sloupec tak, že zjistíte rozdíl mezi nejnižšími náklady a následujícími nejvyššími náklady v daném řádku/sloupci. Pokud existují dvě stejné nejnižší náklady, penalizace je nula.
2. Vyberte řádek/sloupec s nejvyšší penalizací a proveďte přiřazení v buňce s nejnižšími náklady v vybraném řádku/sloupci. Pokud existují dvě nebo více stejné penalizace, vyberte řádek/sloupec s nejnižšími náklady na jednotku. Pokud je rovnost i nadále, vyberte ten, kde je možné provést maximální přiřazení.

3. Odstraňte řádek/sloupec, který uspokojil požadavky na dodávky a poptávku.
4. Opakujte kroky 1 a 2, dokud nebudou všechny dodávky a poptávky uspokojeny.
5. Získejte počáteční základní přípustné řešení. [13]

1.2.4 MODI metoda

Metoda potenciálů, známá také jako MODI (z angl. Modified Distribution Method), představuje druhý krok při řešení dopravní úlohy, který následuje po nalezení základního přípustného řešení. Tato metoda slouží ke zlepšování základního přípustného řešení dopravní úlohy a umožňuje nalézt řešení optimální. Je založena na principu řešení duální úlohy a využívá tzv. potenciály.

Postup:

1. Nechť u_i a v_j jsou pomocné proměnné (potenciály) přiřazené ke řádkům a sloupcům, které slouží k výpočtu nákladů a určení optimálnosti řešení. Pro začátek je některé u_i nebo v_j přiřazeno hodnotě nula. Doporučuje se přiřadit nulu tomu u_i nebo v_j , kde je v řádku nebo sloupci nejvíce přiřazení, což výrazně sníží výpočetní náročnost. Poté je nutné dokončit výpočet ostatních u_i a v_j pomocí vztahu pro všechny obsazené buňky:

$$c_{ij} = u_i + v_j$$

2. Pro neobsazené buňky vypočítejte nepřímé sazby pomocí vztahu:

$$d_{ij} = c_{ij} - (u_i + v_j)$$

3. Zkontrolujte znaménka každého d_{ij} . Znaménka ukazují, zda bylo dosaženo optimálního řešení. Význam znamének je následující:
 - Pokud $d_{ij} > 0$, aktuální základní řešení je optimální.
 - $d_{ij} = 0$ znamená, že existuje alternativní řešení se stejnou celkovou hodnotou. Pokud $d_{ij} = 0$, aktuální řešení zůstává beze změny, ale existuje alternativa
 - Pokud $d_{ij} < 0$, lze získat vylepšené řešení.
4. Vytvořte smyčku pro neobsazenou buňku s největší zápornou hodnotou nepřímé sazby. Začněte touto neobsazenou buňkou a označte ji znaménkem plus (+). Sledujte cestu podél řádků (nebo sloupců) k obsazené buňce, označte roh

znaménkem minus (-), pokračujte střídavě plus a minus, dokud smyčka nezavře zpět k vybrané neobsazené buňce.

5. Vyberte nejmenší množství z buněk označených znaménkem minus na rozích uzavřené smyčky. Toto množství přiřďte vybrané neobsazené buňce, přičtěte ho k ostatním obsazeným buňkám označeným znaménkem plus a odečtěte od obsazených buněk označených znaménkem minus.
6. Získejte nové vylepšené řešení podle kroku 5 a vypočítejte nové celkové náklady na přepravu.
7. Ověřte nové řešení z hlediska optimality. Postup ukončete, pokud jsou všechna $d_{ij} \geq 0$ pro neobsazené buňky. [14]

Podmínka použitelnosti:

Pro správné použití metody musí výchozí řešení obsahovat alespoň $m + n - 1$ obsazených polí. Pokud tato podmínka není splněna, jedná se o degenerované řešení, které vyžaduje dodatečné úpravy.

2 PRAKTICKÉ VYUŽITÍ OPTIMALIZAČNÍCH METOD

V této kapitole se zaměřím na praktické využití metod optimalizace v různých oblastech, jako je průmysl, doprava, ekonomika a informační technologie. Podrobně prozkoumám, jak každá z použitých metod přispívá ke snižování nákladů, optimalizaci složitých procesů a efektivnějšímu řízení zdrojů.

2.1 Přiřazovací úloha

Nejprve obecně popíšeme oblasti využití problému přiřazení v reálném životě, následně budou představeny tři konkrétní příklady jeho aplikace.

Tento problém je aktuální v následujících oblastech:

- Optimalizace doručovacích tras:

Optimalizace tras pomáhá snížit náklady na palivo a údržbu vozidel, zkracuje dobu doručení a zvyšuje spokojenost zákazníků i produktivitu řidičů.

- Rozvrhování pracovních sil:

Při plánování směn nemocničních sester se zohledňují jejich dostupnost, dovednosti a preference, aby byla zajištěna efektivní práce a pokrytí.

- Zadání projektu:

Projektoví manažeři optimalizují přiřazení úkolů členům týmu na základě jejich schopností a dostupnosti, čímž maximalizují efektivitu projektu.

- Shodovací algoritmy:

Algoritmy přiřazování pomáhají párovat lidi nebo věci podle vlastností a preferencí, například v online seznamkách nebo při hledání práce.

- Plánování sportů:

Úloha přiřazení se využívá k optimálnímu rozdělení týmů na zápasy s ohledem na dostupnost míst, cestovní vzdálenosti a pořadí. [15]

Nyní si podrobněji představíme tři konkrétní případy, které ilustrují praktické využití problému přiřazení.

1. V roce 2021 byla publikována studie zabývající se problémem přiřazení studentů do tříd na Gunma University v Japonsku. Autoři řešili situaci, kdy každý student má seznam preferovaných tříd seřazený podle svých priorit. V rámci této práce

navrhli nové přístupy k optimalizaci přiřazení, včetně tzv. minimax-rank omezených maximálních užitkových přiřazení a kompromisu mezi maximálním užitkem a spravedlností přiřazení. Výsledky studie ukazují, že použitím navržených metod bylo možné významně snížit počet studentů, kteří nebyli přiřazení k třídám z jejich prvních tří preferencí, téměř na nulu. [16]

2. Dalším příkladem praktického využití optimalizační úlohy přiřazení je studie publikovaná v roce 1998, která se zabývá rozvrhem zkoušek na Právnické fakultě Bělehradské univerzity. Autoři řešili problém přiřazení studentů ke zkouškovým dnům s ohledem na specifická pravidla. Jelikož stávající modely nebyly dostatečné, navrhli nový přístup založený na teorii grafů, kde bylo přiřazení převedeno na hledání maximálních klik. K nalezení řešení byla použita speciální heuristika v kombinaci s metodou Tabu search. Výsledky ukázaly, že navržený přístup byl úspěšně aplikován při několika reálných zkouškových obdobích a poskytl kvalitní a prakticky využitelná řešení. [17]
3. Třetí zajímavý příklad využití úlohy přiřazení představuje studie, která se zabývá výběrem vhodného notebooku podle typu uživatele. Autoři rozdělili uživatele do čtyř kategorií podle jejich požadavků a potřeb a přiřazovali jim odpovídající notebooky z nabídky 21 modelů. Pro řešení této úlohy, která má nevyvážený charakter, byl použit ROA algoritmus v kombinaci s tzv. „Ghadle and Muley“ pravidlem. Výpočet optimálního řešení byl následně ověřen pomocí programu v MATLABu, který poskytl výsledky v řádu milisekund. Studie tak ukazuje, jak lze pomocí modelu přiřazení efektivně řešit i spotřebitelská rozhodnutí v reálném životě. [18]

2.2 Dopravní úloha

1. V roce 2024 bylo publikováno výzkumné článkové studie, které se zabývalo reálným dopravním problémem ve společnosti Hyundai, kde bylo třeba optimalizovat rozvoz automobilů z několika továren do distribučních center s cílem minimalizovat celkové přepravní náklady. Autoři modelovali situaci jako klasický transportní problém, ve kterém byly výrobní kapacity a poptávka po vozech rovnoměrně rozloženy mezi zdroji a cíli. V případě nerovnováhy mezi nabídkou a poptávkou využili fiktivní továrny nebo distribuční centra, což umožnilo model udržet vyvážený a najít optimální řešení. Pro výpočet byla

využita metoda lineárního programování a software TORA. Výsledky ukázaly, že daný přístup je efektivní při řešení rozsáhlých reálných logistických problémů v automobilovém průmyslu. [19]

2. V oblasti farmaceutického průmyslu se dopravní úloha často využívá k optimalizaci logistických procesů, zejména při distribuci výrobků mezi výrobními závody a distribučními centry. V roce 2015 byla publikována studie, která se zabývá případem společnosti BioPharma vyrábějící chemické produkty pro farmaceutický průmysl. V práci je použita modelová úloha přepravy založená na lineárním programování s cílem minimalizovat celkové náklady na dopravu mezi šesti závody a šesti odběratelskými místy. Optimalizace umožnila vypracovat nový plán dodávek, který snižuje náklady společnosti přibližně o 12 %, což představuje zhruba 3,97 milionu USD měsíčně. [20]
3. V oblasti petrochemického průmyslu, kde je klíčová efektivní distribuce produktů na mezinárodní trhy, se často využívá transportní problém pro optimalizaci přepravních nákladů. V roce 2020 byla publikována studie, která se zabývala optimalizací přepravy polymeru polyvinylchloridu vyráběného v Malajsii. Studie představila případ malajské petrochemické společnosti Petronas a její obchodní dceřiné firmy Mitco Labuan, která zajišťuje distribuci z výroben do čtyř hlavních exportních destinací: Číny, Blízkého východu, Evropy a jihovýchodní Asie. Pro řešení logistického problému byl použit klasický transportní model s cílem minimalizovat celkové náklady na dopravu. Byly aplikovány metody lineárního programování VAM a MODI, které umožnily nalézt optimální distribuční plán. Výsledkem bylo nalezení řešení, které minimalizuje přepravní náklady na 120 000 milionů malajsijských ringgitů. Studie potvrdila, že lineární programování představuje efektivní nástroj pro plánování distribuce a maximalizaci zisků v petrochemickém průmyslu. [21]

3 ŘEŠENÍ NETRIVIÁLNÍCH OPTIMALIZAČNÍCH ÚLOH

V rámci této části ukážeme postup řešení pěti různých netriviálních optimalizačních problémů. Pro řešení každé úlohy bude použitý jiný přístup.

3.1 Přiřazovací úloha

3.1.1 Maďarská metoda

Zadání úlohy:

V logistickém centru pracuje 5 dispečerů: Jan, Petr, Karel, Lucie a Tereza. Každý z nich musí být přiřazen k jedné směně během pracovního týdne (pondělí až pátek).

Každý dispečer má jinou efektivitu v závislosti na konkrétním dni, kdy pracuje. Čím nižší číslo, tím efektivněji daný dispečer danou směnu zvládne (například s nižšími časovými nebo provozními náklady). Najděte takové přiřazení dispečerů ke směnám, aby celkové náklady byly minimální a žádná směna nebyla obsazena více než jednou osobou.

Následující matice udává náklady na přiřazení jednotlivých dispečerů ke směnám:

Tabulka 1 Nákladová matice

Pracovníky	Dny v týdnu				
	Po	Út	St	Čt	Pá
Jan	8	5	9	4	9
Petr	3	4	3	6	1
Karel	5	1	2	5	2
Lucie	5	7	1	1	3
Tereza	9	4	9	1	6

(zdroj: vlastní)

Řešení:

Řešení úlohy přiřazení pomocí Maďarské metody začíná redukcí řádků a sloupců matice nákladů. Za tímto účelem nejprve nalezneme minimální hodnotu v každém řádku matice.

Tabulka 2 Nalezení minimalního prvku v řádku

Pracovníky	Dny v týdnu				
	Po	Út	St	Čt	Pá
Jan	8	5	9	4	9
Petr	3	4	3	6	1
Karel	5	1	2	5	2
Lucie	5	7	1	1	3
Tereza	9	4	9	1	6

(zdroj: vlastní)

Odečtením těchto hodnot od všech prvků příslušných řádků získáme matici, která je zobrazena v tab. 3.

Tabulka 3 Matice po odečtení min řádku

Pracovníky	Dny v týdnu				
	Po	Út	St	Čt	Pá
Jan	4	1	5	0	5
Petr	2	3	2	5	0
Karel	4	0	1	4	1
Lucie	4	6	0	0	2
Tereza	8	3	8	0	5

(zdroj: vlastní)

Redukce sloupců se provádí stejným způsobem jako redukce řádků matice, ale všechny operace se provádějí pouze se sloupci. Po nalezení minimální hodnoty v každém sloupci matice (viz tab. 4) odečteme tyto hodnoty od všech prvků příslušných sloupců.

Tabulka 4 Minimalní hodnoty ve sloupcích

Pracovníky	Dny v týdnu				
	Po	Út	St	Čt	Pá
Jan	4	1	5	0	5
Petr	2	3	2	5	0
Karel	4	0	1	4	1
Lucie	4	6	0	0	2
Tereza	8	3	8	0	5

(zdroj: vlastní)

Výsledkem těchto operací získáme redukovanou matici, která je zobrazena v tab. 5.

Tabulka 5 Zredukováná matice

Pracovníky	Dny v týdnu				
	Po	Út	St	Čt	Pá
Jan	2	1	5	0	5
Petr	0	3	2	5	0
Karel	2	0	1	4	1
Lucie	2	6	0	0	2
Tereza	6	3	8	0	5

(zdroj: vlastní)

V dalším kroku nakreslíme čáry přes nulové prvky v matici tak, aby všechny nuly byly pokryty a zároveň bylo použito minimální možné množství těchto čar.

Tabulka 6 Pokrytí nul čarami

Pracovníky	Dny v týdnu				
	Po	Út	St	Čt	Pá
Jan	2	1	5	0	5
Petr	0	3	2	5	0
Karel	2	0	1	4	1
Lucie	2	6	0	0	2
Tereza	6	3	8	0	5

(zdroj: vlastní)

Následně je nutné ověřit, zda se počet pokrývacích čar rovná rozměru čtvercové nákladové matice.

V tomto případě se počet pokrývacích čar nerovná rozměru matice – je jejich 4, proto je nutné pokračovat úpravou matice následujícím způsobem:

- Určíme nejmenší prvek, který není pokryt žádnou čarou.
- Odečtete tento prvek od každého nepokrytého prvku v řádcích a zároveň jej přičtete ke každému prvku, který leží na průsečíku pokrytých řádků a sloupců.

- Následně se vrátíme ke kroku 3 a proces opakujte, dokud počet čar nebude odpovídat rozměru matice.

Tabulka 7 Nejmenší nepokrytý čarou prvek

Pracovníky	Dny v týdnu				
	Po	Út	St	Čt	Pá
Jan	2	1	5	0	5
Petr	0	3	2	5	0
Karel	2	0	1	4	1
Lucie	2	6	0	0	2
Tereza	6	3	8	0	5

(zdroj: vlastní)

Po upravě matice vypadá takto:

Tabulka 8 Matice po odečtení a přičtení minimálního prvku

Pracovníky	Dny v týdnu				
	Po	Út	St	Čt	Pá
Jan	1	0	4	1	4
Petr	0	3	2	6	0
Karel	2	0	1	5	1
Lucie	2	6	0	1	2
Tereza	5	2	7	0	4

(zdroj: vlastní)

V dalším kroku znovu nakreslíme čáry přes nulové prvky v matici tak, aby všechny nuly byly pokryty a zároveň bylo použito minimální možné množství těchto čar.

Tabulka 9 Pokrýtí nul čarami

Pracovníky	Dny v týdnu				
	Po	Út	St	Čt	Pá
Jan	1	0	4	1	4
Petr	0	3	2	6	0
Karel	2	0	1	5	1
Lucie	2	6	0	1	2
Tereza	5	2	7	0	4

(zdroj: vlastní)

Následně je nutné ověříme, zda se počet pokrývacích čar rovná rozměru čtvercové nákladové matice. Znovu se počet pokrývacích čar nerovná rozměru matice – je jejich 4, proto je nutné pokračujeme s úpravou matice.

Tabulka 10 Minimální nepokrytý prvek

Pracovníky	Dny v týdnu				
	Po	Út	St	Čt	Pá
Jan	1	0	4	1	4
Petr	0	3	2	6	0
Karel	2	0	1	5	1
Lucie	2	6	0	1	2
Tereza	5	2	7	0	4

(zdroj: vlastní)

Po upravě matice vypada takto:

Tabulka 11 Matice po upřadě

Pracovníky	Dny v týdnu				
	Po	Út	St	Čt	Pá
Jan	0	0	4	1	3
Petr	0	4	3	7	0
Karel	1	0	1	5	0
Lucie	1	6	0	1	1
Tereza	4	2	7	0	3

(zdroj: vlastní)

V dalším kroku nakreslíme čary.

Tabulka 12 Matice po pokrytí čarami

Pracovníky	Dny v týdnu				
	Po	Út	St	Čt	Pá
Jan	~~0~~	~~0~~	~~4~~	~~1~~	~~3~~
Petr	~~0~~	4	~~3~~	~~7~~	~~0~~
Karel	~~1~~	0	~~1~~	~~5~~	~~0~~
Lucie	~~1~~	6	~~0~~	~~1~~	~~1~~
Tereza	~~4~~	2	~~7~~	~~0~~	~~3~~

(zdroj: vlastní)

Po pokrytí nul čarami zkontrolujeme, zda je počet čar roven rozměru matice. Teď je počet čar roven rozměru matice, a proto je možné provést optimální přiřazení.

Dalším krokem je provést vlastní přiřazení nul v matici. Provedeme to tak, že vybereme takové nulové prvky, aby v každém řádku a každém sloupci bylo právě jedno přiřazení a žádná dvě přiřazení nebyla ve stejném řádku ani sloupci. Nejprve vybíráme nulové prvky v řádcích nebo sloupcích, které obsahují pouze jednu nulu. Při každém přiřazení zároveň vyškrtneme příslušný řádek a sloupec z dalšího výběru. Tento postup opakujeme, dokud není přiřazení provedeno pro všechny řádky.

Tabulka 13 Přřazené nuly

Pracovníky	Dny v týdnu				
	Po	Út	St	Čt	Pá
Jan	0	0	4	1	3
Petr	0	4	3	7	0
Karel	1	0	1	5	0
Lucie	1	6	0	1	1
Tereza	4	2	7	0	3

(zdroj: vlastní)

Vratíme se k původní nákladové matice a odečteme optimální výsledek.

Tabulka 14 Optimální přiřazení

Pracovníky	Dny v týdnu				
	Po	Út	St	Čt	Pá
Jan	8	5	9	4	9
Petr	3	4	3	6	1
Karel	5	1	2	5	2
Lucie	5	7	1	1	3
Tereza	9	4	9	1	6

(zdroj: vlastní)

Z matice odečteme optimální přiřazení a minimální náklady:

Celkové minimální náklady jsou:

$$8 + 1 + 1 + 1 + 1 = 12$$

Optimální přiřazení je následující:

- Jan bude pracovat v pondělí
- Petr v pátek
- Karel v úterý
- Lucie ve středu
- Tereza ve čtvrtek

3.1.2 Mackův algoritmus

Zadání:

Čtyři týmy zkušených kuchařů se účastní prestižního gastronomického festivalu, kde budou připravovat čtyři různé slavnostní večeře: italskou, francouzskou, japonskou a mexickou. Každý tým má jiné zkušenosti s jednotlivými kuchyněmi, což ovlivňuje dobu přípravy (v hodinách), kterou potřebují na přípravu každé večeře.

Rozhodněte, který tým bude připravovat kterou večeři tak, aby byl celkový čas přípravy všech slavnostních večeří minimální.

Tabulka níže ukazuje, kolik hodin by trvala příprava každé večeře jednotlivými týmy:

Tabulka 15 Nákladová matice

	Italská	Francouzská	Japonská	Mexická
Tým 1	9	2	7	8
Tým 2	6	4	3	7
Tým 3	5	8	1	8
Tým 4	7	6	9	4

(zdroj: vlastní)

Řešení:

1. Podtrhneme v každém řádku minimální hodnoty. Sloupec č. 1 nemá podtržené prvky.

Tabulka 16 Minimalní hodnoty v řádcích

9	<u>2</u>	7	8
6	4	<u>3</u>	7
5	8	<u>1</u>	8
7	6	9	<u>4</u>

(zdroj: vlastní)

2. Označíme sloupec, ve kterém je více než jeden podtržený prvek, znakem „+“

Tabulka 17 Označený sloupec

		+	
9	<u>2</u>	7	8
6	4	<u>3</u>	7
5	8	<u>1</u>	8
7	6	9	<u>4</u>

(zdroj: vlastní)

3. Prohlížíme řádky sloupce 3:

- řádek 2: podtržený prvek 3, minimální rozdíl $4 - 3 = 1$
- řádek 3: podtržený prvek 1, minimální rozdíl $5 - 1 = 4$

Minimální rozdíl je ve řádku č. 2.

4. Zvětšíme všechny prvky 3. sloupce o minimální rozdíl - 1.

Tabulka 18 Přidání minimálního rozdílu k prvkům

		+	
9	<u>2</u>	8	8
6	4	<u>4</u>	7
5	8	<u>2</u>	8
7	6	10	<u>4</u>

(zdroj: vlastní)

5. Označíme prvek 4 v řádku č. 2 hvězdičkou.

Tabulka 19 Označení hvězdičkou

		+	
9	<u>2</u>	8	8
6	4*	<u>4</u>	7
5	8	<u>2</u>	8
7	6	10	<u>4</u>

(zdroj: vlastní)

6. Zkontrolujeme sloupec č. 2, v něm již je podtržený prvek v řadku č. 1. Označíme tento sloupec + a vrátíme se ke kroku č. 3.

Tabulka 20 6. krok - kontrola a označení sloupce

	+	+	
9	<u>2</u>	8	8
6	4*	<u>4</u>	7
5	8	<u>2</u>	8
7	6	10	<u>4</u>

(zdroj: vlastní)

7. V rámci tohoto kroku jsme se vrátili ke kroku c. 3.

Prohlížíme řádky, které obsahují prvky z označených znakem „+” sloupců:

- řádek 1: podtržený prvek 2, minimální rozdíl $8 - 2 = 6$
- řádek 2: podtržený prvek 4, minimální rozdíl $6 - 1 = 2$
- řádek 3: podtržený prvek 2, minimální rozdíl $5 - 2 = 3$

Minimální rozdíl 2 je ve řadku č. 2.

8. Zvětšíme všechny prvky označených znakem „+” sloupců na 2

Tabulka 21 Přidání minimalního rozdílu k prvkům

	+	+	
9	<u>4</u>	10	8
6	6*	<u>6</u>	7
5	10	<u>4</u>	8
7	8	12	<u>4</u>

(zdroj: vlastní)

9. Označíme prvek 6 v řádce č. 2 hvězdičkou.

Tabulka 22 Označený hvězdičkou prvek

	+	+	
9	<u>4</u>	10	8
6*	6*	<u>6</u>	7
5	10	<u>4</u>	8
7	8	12	<u>4</u>

(zdroj: vlastní)

10. Prohlížíme 1. sloupec, který nemá žádný podtržený prvek. Pokračujeme krokem 7.

11. Podtržíme prvek 6 v 1. sloupci a odstraníme hvězdičku z něj.

Tabulka 23 7. krok

	+	+	
9	<u>4</u>	10	8
<u>6</u>	6*	<u>6</u>	7
5	10	<u>4</u>	8
7	8	12	<u>4</u>

(zdroj: vlastní)

12. Pokračujeme krokem 8: odstraníme podtržítka z prvku 6 ve sloupci č. 3.

Tabulka 24 8. krok

	+	+	
9	<u>4</u>	10	8
<u>6</u>	6*	6	7
5	10	<u>4</u>	8
7	8	12	<u>4</u>

(zdroj: vlastní)

13. Zkontrolujeme sloupec č. 3 po odstranění podržítka z prvku 6. Ve sloupci je další podržený prvek, pokračujeme u kroku 10.
14. Jelikož každý sloupec má podržený prvek, optimální řešení je nalezeno. Vratíme se k nákladové matici.

Tabulka 25 Nákladová matice s optimálními hodnotami

9	<u>2</u>	7	8
<u>6</u>	4	3	7
5	8	<u>1</u>	8
7	6	9	<u>4</u>

(zdroj: vlastní)

Celkové minimální náklady jsou:

$$2 + 6 + 1 + 4 = 13 \text{ hodin}$$

Na základě řešení úlohy bylo nalezeno následující optimální rozdělení týmů k večerům:

- Tým 1 - Francouzská
- Tým 2 - Italská
- Tým 3 - Japonská
- Tým 4 - Mexická

3.2 Dopravní problém

V rámci této části ukážeme postup řešení dopravního problému. Základní přípustné řešení bude nalezeno pomocí metody severozápadního rohu, následně řešení bude optimalizováno pomocí MODI metody.

Zadání úlohy:

Společnost zajišťuje přepravu zboží od 3 dodavatelů ke 4 odběratelům. Každý dodavatel má omezenou kapacitu (zásoby) a každý odběratel má přesně daný požadavek (poptávku). Náklady na přepravu jedné jednotky zboží z jednotlivých dodavatelů k jednotlivým odběratelům jsou uvedeny v tabulce níže (v Kč). Najděte takový dopravní plán, který pokryje veškerou poptávku odběratelů a zároveň minimalizuje celkové přepravní náklady.

Tabulka 26 Matice nákladů

	Odběratel A	Odběratel B	Odběratel C	Odběratel D	Zásoby
Dodavatel 1	8	6	10	9	20
Dodavatel 2	9	7	4	2	30
Dodavatel 3	3	4	2	15	20
Poptávka	10	25	20	15	

zdroj:vlastní

Řešení:

Prvním krokem je ověření zda problém je vyrovnaný:

1. Celkové zásoby:

$$20 + 30 + 20 = 70$$

2. Celková poptávka:

$$10 + 25 + 20 + 15 = 70$$

Jelikož zásoby se rovnají poptávce, je úloha vyrovnaná a můžeme přistoupit k jejímu řešení. Pokud by se součty neshodovaly, bylo by třeba doplnit nebo upravit úlohu (například přidáním umělého dodavatele nebo odběratele s odpovídajícími náklady na přepravu).

Pro nalezení základního výchozího přípustného řešení bude použita Metoda severozápadního rohu:

1. Krok:

- Porovnáme hodnoty nabídky a poptávky pro buňky D_1 a O_A

$$\min \{10, 20\} = 10$$

- Menší z těchto dvou hodnot přiřadíme do buňky $D_1 O_A$.
- Tím je plně uspokojena poptávka odběratele O_A a u dodavatele D_1 zbývá 10 jednotek.

Tabulka 27 1. krok výpočtu

	Odběratel A	Odběratel B	Odběratel C	Odběratel D	Zásoby
Dodavatel 1	8 (10)	6	10	9	20
Dodavatel 2	9	7	4	2	30
Dodavatel 3	3	4	2	15	20
Poptávka	10	25	20	15	

(zdroj: vlastní)

2. Krok:

- Porovnáme hodnoty nabídky a poptávky pro uzly $D_1 = 10$ a $O_B = 25$

$$\min \{25, 10\} = 10$$

- Menší z těchto dvou hodnot přiřadíme do buňky $D_1 O_B$.
- Tímto se vyčerpá kapacita dodavatele D_1 a u odběratele O_B zbývá $25 - 10 = 15$ jednotek.

Tabulka 28 2. krok výpočtu

	Odběratel A	Odběratel B	Odběratel C	Odběratel D	Zásoby
Dodavatel 1	8 (10)	6 (10)	10	9	~~20~~ 10
Dodavatel 2	9	7	4	2	30
Dodavatel 3	3	4	2	15	20
Poptávka	10	25	20	15	

(zdroj: vlastní)

3. Krok:

- Porovnáme hodnoty nabídky a poptávky pro $D_2 = 30$ a $O_B = 15$.

$$\min \{30, 15\} = 15$$

- Menší z těchto dvou hodnot přiřadíme do buňky $D_2 O_B$.
- Tím je plně uspokojena poptávka odběratele O_B a u dodavatele D_2 zbývá $30 - 15 = 15$ jednotek.

Tabulka 29 3. krok výpočtu

	Odběratel A	Odběratel B	Odběratel C	Odběratel D	Zásoby
Dodavatel 1	8 (10)	6 (10)	10	9	~~20~~ (0)
Dodavatel 2	9	7 (15)	4	2	~~30~~ (15)
Dodavatel 3	3	4	2	15	20
Poptávka	0	0	20	15	

(zdroj: vlastní)

4. Krok:

- Porovnáme hodnoty nabídky a poptávky pro $D_2 = 15$ a $O_C = 20$.
- Menší z těchto dvou hodnot přiřadíme do buňky $D_2 O_C$.
- Tímto se vyčerpá kapacita dodavatele S_2 a u odběratele O_C zbývá $20 - 15 = 5$ jednotek.

Tabulka 30 4. krok výpočtu

	Odběratel A	Odběratel B	Odběratel C	Odběratel D	Zásoby
Dodavatel 1	8 (10)	6 (10)	10	9	20 (0)
Dodavatel 2	9	7 (15)	4 (15)	2	30 (0)
Dodavatel 3	3	4	2	15	20
Poptávka	0	0	5	15	

(zdroj: vlastní)

5. Krok:

- Porovnáme hodnoty nabídky a poptávky pro uzly $D_3 = 20$ a $O_C = 5$
- Menší z těchto dvou hodnot přiřadíme do buňky $D_3 O_C$.
- Tím je plně uspokojena poptávka odběratele O_C a u dodavatele D_3 zbývá $20 - 5 = 15$ jednotek.

Tabulka 31 5. krok výpočtu

	Odběratel A	Odběratel B	Odběratel C	Odběratel D	Zásoby
Dodavatel 1	8 (10)	6 (10)	10	9	20 (0)
Dodavatel 2	9	7 (15)	4 (15)	2	30 (0)
Dodavatel 3	3	4	2 (5)	15	20 (15)
Poptávka	0	0	0	15	

(zdroj: vlastní)

6. Krok:

- Porovnáme hodnoty nabídky a poptávky pro uzly $D_3 = 15$ a $O_D = 15$.
- Menší z těchto dvou hodnot přiřadíme do buňky $D_3 O_D$.

Tabulka 32 6. krok výpočtu

	Odběratel A	Odběratel B	Odběratel C	Odběratel D	Zásoby
Dodavatel 1	8 (10)	6 (10)	10	9	20 (0)
Dodavatel 2	9	7 (15)	4 (15)	2	30 (0)
Dodavatel 3	3	4	2 (5)	15 (15)	20 (0)
Poptávka	0	0	0	0	

(zdroj: vlastní)

7. Krok:

Tabulka 33 7. krok výpočtu

	Odběratel A	Odběratel B	Odběratel C	Odběratel D	Zásoby
Dodavatel 1	8 (10)	6 (10)	10	9	20
Dodavatel 2	9	7 (15)	4 (15)	2	30
Dodavatel 3	3	4	2 (5)	15 (15)	20
Poptávka	10	25	20	15	540

(zdroj: vlastní)

Celkové náklady na dopravu pro toto řešení:

$$8 \times 10 + 6 \times 10 + 7 \times 15 + 4 \times 15 + 2 \times 5 + 15 \times 15 = 540 \text{ Kč}$$

Pro ověření optimality základního přípustného řešení a nalezení optimálního řešení je použita MODI metoda:

Tabulka 34 Základní přípustné řešení

	Odběratel A	Odběratel B	Odběratel C	Odběratel D	Zásoby
Dodavatel 1	8 (10)	6 (10)	10	9	20
Dodavatel 2	9	7 (15)	4 (15)	2	30
Dodavatel 3	3	4	2 (5)	15 (15)	20
Poptávka	10	25	20	15	

(zdroj: vlastní)

Iterace 1:

1. Najdeme u_i a v_j pro všechna obsazená pole (i,j) pomocí vzorce $c_{ij} = u_i + v_j$:

Zvolíme-li $u_1 = 0$, dostaneme:

$$c_{11} = u_1 + v_1 \Rightarrow v_1 = c_{11} - u_1 \Rightarrow v_1 = 8 - 0 \Rightarrow v_1 = 8$$

$$c_{12} = u_1 + v_2 \Rightarrow v_2 = c_{12} - u_1 \Rightarrow v_2 = 6 - 0 \Rightarrow v_2 = 6$$

$$c_{22} = u_2 + v_2 \Rightarrow u_2 = c_{22} - v_2 \Rightarrow u_2 = 7 - 6 \Rightarrow u_2 = 1$$

$$c_{23} = u_2 + v_3 \Rightarrow v_3 = c_{23} - u_2 \Rightarrow v_3 = 4 - 1 \Rightarrow v_3 = 3$$

$$c_{33} = u_3 + v_3 \Rightarrow u_3 = c_{33} - v_3 \Rightarrow u_3 = 2 - 3 \Rightarrow u_3 = -1$$

$$c_{34} = u_3 + v_4 \Rightarrow v_4 = c_{34} - u_3 \Rightarrow v_4 = 15 + 1 \Rightarrow v_4 = 16$$

Tabulka 35 Tabulkové znázornění řádkových a sloupcových čísel

	Odb. A	Odb.B	Odb.C	Odb.D	Zásoby	u_i
D1	8 (10)	6 (10)	10	9	20	$u_1 = 0$
D2	9	7 (15)	4 (15)	2	30	$u_2 = 1$
D3	3	4	2 (5)	15 (15)	20	$u_3 = -1$
Poptávka	10	25	20	15		
v_j	$v_1 = 8$	$v_2 = 6$	$v_3 = 3$	$v_4 = 16$		

(zdroj: vlastní)

2. Vypočítáme nepřímé sazby d_{ij} pro všechna neobsazená pole (i,j) pomocí vzorce:

$$d_{ij} = c_{ij} - (u_i + v_j)$$

$$d_{13} = c_{13} - (u_1 + v_3) = 10 - (0 + 3) = 7$$

$$d_{14} = c_{14} - (u_1 + v_4) = 9 - (0 + 16) = -7$$

$$d_{21} = c_{21} - (u_2 + v_1) = 9 - (1 + 8) = 0$$

$$d_{24} = c_{24} - (u_2 + v_4) = 2 - (1 + 16) = -15$$

$$d_{31} = c_{31} - (u_3 + v_1) = 3 - (-1 + 8) = -4$$

$$d_{32} = c_{32} - (u_3 + v_2) = 4 - (-1 + 6) = -1$$

Jelikož pole D1C3 má nepřímou sazbu 7, test optimality není splněn.

Tabulka 36 Tabulkové znázornění nepřímých sazeb

	Odb A	OdbB	Odb C	OdbD	Zásoby	u_i
D1	8 (10)	6 (10)	10 [7]	9 [-7]	20	0
D2	9 [0]	7 (15)	4 (15)	2 [-15]	30	1
D3	3 [-4]	4 [-1]	2 (5)	15 (15)	20	-1
Poptávka	10	25	20	15		
v_j	8	6	3	16		

(zdroj: vlastní)

Nové základní řešení:

3. Nyní vybereme nejmenší zápornou hodnotu z všech $d_{ij} : d_{24} = -15$. Vykreslíme uzavřenou cestu začínající v buňce D_2O_D .

Tabulka 37 Znázornění uzavřené cesty

	Odb A	OdbB	Odb C	OdbD	Zásoby	U_i
D 1	8 (10)	6 (10)	10 [7]	9 [-7]	20 (0)	0
D 2	9 [0]	7 (15)	4 (15) -	2 [-15] +	30 (0)	1
D 3	3 [-4]	4 [-1]	2 (5) +	15 (15) -	20 (0)	-1
Poptávka	10	25	20	15		
V_j	8	6	3	16		

4. Nejmenší přidělená hodnota mezi všemi zápornými pozicemi na uzavřené cestě je 15. Od všech hodnot označených mínusem odečteme 15 a k hodnotám označeným plusem přičteme 15.

Tabulka 38 Nové základní řešení

	Odb A	OdbB	Odb C	OdbD	Zásoby
D 1	8 (10)	6 (10)	10	9	20
D 2	9	7 (15)	4	2 (15)	30
D 3	3	4	2 (20)	15	20
Poptávka	10	25	20	15	

(zdroj: vlastní)

Stejným způsobem opakujeme kroky 1-4, dokud nebude nalezeno optimální řešení.

Protože je počet přidělených buněk o jednu menší než $m + n - 1$, řešení je degenerované.

Proto přiřátíme hodnotu d do $D_3 O_A$, která má minimální náklady na přepravu.

Tabulka 39 Nové základní řešení

	Odb A	OdbB	Odb C	OdbD	Zásoby
D 1	8 (10)	6 (10)	10	9	20
D 2	9	7 (15)	4	2 (15)	30
D 3	3 (d)	4	2 (20)	15	20
Poptávka	10	25	20	15	

(zdroj: vlastní)

Iterace 2:

1. Najdeme u_i a v_j pro všechna obsazená pole (i,j) pomocí rovnice $c_{ij} = u_i + v_j$:

Dosadíme $u_1 = 0$, dostaneme:

$$c_{11} = u_1 + v_1 \Rightarrow v_1 = c_{11} - u_1 = 8 - 0 = 8$$

$$c_{31} = u_3 + v_1 \Rightarrow u_3 = c_{31} - v_1 = 3 - 8 = -5$$

$$c_{33} = u_3 + v_3 \Rightarrow v_3 = c_{33} - u_3 = 2 - (-5) = 7$$

$$c_{12} = u_1 + v_2 \Rightarrow v_2 = c_{12} - u_1 = 6 - 0 = 6$$

$$c_{22} = u_2 + v_2 \Rightarrow u_2 = c_{22} - v_2 = 7 - 6 = 1$$

$$c_{24} = u_2 + v_4 \Rightarrow v_4 = c_{24} - u_2 = 2 - 1 = 1$$

Tabulka 40 Znazornění řádkových a sloupcových čísel

	Odb A	OdbB	Odb C	OdbD	Zásoby	u_i
D 1	8 (10)	6 (10)	10	9	20	0
D 2	9	7 (15)	4	2 (15)	30	1
D 3	3 (d)	4	2 (20)	15	20	-5
Poptávka	10	25	20	15		
v_j	8	6	7	1		

(zdroj: vlastní)

2. Vypočítáme nepřímé sazby d_{ij} pro všechna neobsazená pole (i,j) pomocí rovnosti:

$$d_{ij} = c_{ij} - (u_i + v_j)$$

$$d_{13} = c_{13} - (u_1 + v_3) = 10 - (0 + 7) = 3$$

$$d_{14} = c_{14} - (u_1 + v_4) = 9 - (0 + 1) = 8$$

$$d_{21} = c_{21} - (u_2 + v_1) = 9 - (1 + 8) = 0$$

$$d_{23} = c_{23} - (u_2 + v_3) = 4 - (1 + 7) = -4$$

$$d_{32} = c_{32} - (u_3 + v_2) = 4 - (-5 + 6) = 3$$

$$d_{34} = c_{34} - (u_3 + v_4) = 15 - (-5 + 1) = 19$$

Tabulka 41 Tabulkové znázornění nepřímých sazeb

	Odb A	OdbB	Odb C	OdbD	Zásoby	u_i
D 1	8 (10)	6 (10)	10 [3]	9 [8]	20	0
D 2	9 [0]	7 (15)	4 [-4]	2 (15)	30	1
D 3	3 (d)	4 [3]	2 (20)	15 [19]	20	-5
Poptávka	10	25	20	15		
v_j	8	6	7	1		

(zdroj: vlastní)

Jelikož jsou kladné nepřímé sazby, řešení není optimální.

3. Nyní vybereme nejmenší zápornou hodnotu ze všech d_{ij} , což je $d_{23} = -4$

Nakreslíme uzavřenou cestu začínající v buňce D_2O_c .

Tabulka 42 Znazornění obvodu

	Odb A	OdbB	Odb C	OdbD	Zásoby	u_i
D 1	8 (10) -	6 (10) +	10 [3]	9 [8]	20	0
D 2	9 [0]	7 (15) -	4 [-4] +	2 (15)	30	1
D 3	3 (d) +	4 [3]	2 (20) -	15 [19]	20	-5
Poptávka	10	25	20	15		
v_j	8	6	7	1		

(zdroj: vlastní)

4. Nejmenší přidělená hodnota na všech záporných pozicích na uzavřené cestě je 10.
Odečteme 10 od všech buněk se znaménkem mínus a přičteme 10 ke všem buňkám se znaménkem plus.

Tabulka 43 Nové základní řešení

	Odb A	OdbB	Odb C	OdbD	Zásoby
D 1	8	6 (20)	10	9 [15]	20
D 2	9	7 (5)	4 (10)	2 (15)	30
D 3	3 (10)	4	2 (20)	15	20
Poptávka	10	25	20	15	

(zdroj: vlastní)

5. Opakujeme kroky

Iterace 3:

1. Najdeme u_i a v_j pro všechna obsazená pole (i,j) pro které platí $c_{ij} = u_i + v_j$.

Dosadíme $u_2 = 0$, dostaneme:

$$c_{22} = u_2 + v_2 \Rightarrow v_2 = c_{22} - u_2 \Rightarrow v_2 = 7 - 0 = 7$$

$$c_{12} = u_1 + v_2 \Rightarrow u_1 = c_{12} - v_2 \Rightarrow u_1 = 6 - 7 = -1$$

$$c_{23} = u_2 + v_3 \Rightarrow v_3 = c_{23} - u_2 \Rightarrow v_3 = 4 - 0 = 4$$

$$c_{33} = u_3 + v_3 \Rightarrow u_3 = c_{33} - v_3 \Rightarrow u_3 = 2 - 4 = -2$$

$$c_{31} = u_3 + v_1 \Rightarrow v_1 = c_{31} - u_3 \Rightarrow v_1 = 3 + 2 = 5$$

$$c_{24} = u_2 + v_4 \Rightarrow v_4 = c_{24} - u_2 \Rightarrow v_4 = 2 - 0 = 2$$

Tabulka 44 Tabulkové znázornění řádkových a sloupcových čísel

	Odb A	OdbB	Odb C	OdbD	Zásoby	u_i
D 1	8	6 (20)	10	9	20	-1
D 2	9	7 (5)	4 (10)	2 (15)	30	0
D 3	3 (10)	4	2 (20)	15	20	-2
Poptávka	10	25	20	15		
v_j	5	7	4	2		

(zdroj: vlastní)

2. Vypočítáme nepřímé sazby d_{ij} pro všechna neobsazená pole (i,j) pomocí rovnosti:

$$d_{ij} = c_{ij} - (u_i + v_j)$$

$$d_{11} = c_{11} - (u_1 + v_1) = 8 - (-1 + 5) = 4$$

$$d_{13} = c_{13} - (u_1 + v_3) = 10 - (-1 + 4) = 7$$

$$d_{14} = c_{14} - (u_1 + v_4) = 9 - (-1 + 2) = 8$$

$$d_{21} = c_{21} - (u_2 + v_1) = 9 - (0 + 5) = 4$$

$$d_{32} = c_{32} - (u_3 + v_2) = 4 - (-2 + 7) = -1$$

$$d_{34} = c_{34} - (u_3 + v_4) = 15 - (-2 + 2) = 15$$

Tabulka 45 Nepřímé sazby

	Odb A	OdbB	Odb C	OdbD	Zásoby	u_i
D 1	8 [4]	6 (20)	10 [7]	9 [8]	20	-1
D 2	9 [4]	7 (5)	4 (10)	2 (15)	30	0
D 3	3 (10)	4 [-1]	2 (10)	15 [15]	20	2
Poptávka	10	25	20	15		
v_j	5	7	4	2		

(zdroj: vlastní)

3. Nyní vybereme nejmenší zápornou hodnotu ze všech d_{ij} , což je $d_{32} = -1$.

Nakreslíme uzavřenou cestu začínající v buňce D_3O_b .

Tabulka 46 Dantzigův obvod

	Odb A	OdbB	Odb C	OdbD	Zásoby	u_i
D 1	8 [4]	6 (10)	10 [7]	9 [8]	20	-1
D 2	9 [4]	7 (5) -	4 (10) +	2 (15)	30	0
D 3	3 (d)	4 [-1] +	2 (10) -	15 [15]	20	2
Poptávka	10	25	20	15		
v_j	5	7	4	2		

(zdroj: vlastní)

4. Nejmenší přidělená hodnota ze všech záporných pozic na uzavřené cestě je 5.

Odečteme 5 od všech buněk se znaménkem mínus a přičteme 5 ke všem buňkám se znaménkem plus.

Tabulka 47 Nové základní řešení

	Odb A	OdbB	Odb C	OdbD	Zásoby
D 1	8	6 (20)	10	9 [15]	20
D 2	9	7	4 (15)	2 (15)	30
D 3	3 (10)	4 (5)	2 (5)	15	20
Poptávka	10	25	20	15	

(zdroj: vlastní)

5. Opakujeme kroky 1 až 4, dokud nebude dosaženo optimálního řešení.

Čtvrtá iterace:

1. Najdeme u_i a v_j pro všechna obsazená pole (i,j) pro které platí $c_{ij} = u_i + v_j$.

Dosadíme $u_3=0$, dostaneme:

$$c_{31} = u_3 + v_1 \Rightarrow v_1 = c_{31} - u_3 \Rightarrow v_1 = 3 - 0 \Rightarrow v_1 = 3$$

$$c_{32} = u_3 + v_2 \Rightarrow v_2 = c_{32} - u_3 \Rightarrow v_2 = 4 - 0 \Rightarrow v_2 = 4$$

$$c_{12} = u_1 + v_2 \Rightarrow u_1 = c_{12} - v_2 \Rightarrow u_1 = 6 - 4 \Rightarrow u_1 = 2$$

$$c_{33} = u_3 + v_3 \Rightarrow v_3 = c_{33} - u_3 \Rightarrow v_3 = 2 - 0 \Rightarrow v_3 = 2$$

$$c_{23} = u_2 + v_3 \Rightarrow u_2 = c_{23} - v_3 \Rightarrow u_2 = 4 - 2 \Rightarrow u_2 = 2$$

$$c_{24} = u_2 + v_4 \Rightarrow v_4 = c_{24} - u_2 \Rightarrow v_4 = 2 - 2 \Rightarrow v_4 = 0$$

	Odb A	OdbB	Odb C	OdbD	Zásoby	u_i
D 1	8	6 (20)	10	9	20	2
D 2	9	7	4 (15)	2 (15)	30	2
D 3	3 (10)	4 (5)	2 (5)	15	20	0
Poptávka	10	25	20	15		
v_j	3	4	2	0		

(zdroj: vlastní)

2. Najdeme d_{ij} pro všechna neobsazená pole (i,j) , pro které platí

$$d_{ij} = c_{ij} - (u_i + v_j)$$

$$d_{11} = c_{11} - (u_1 + v_1) = 8 - (2 + 3) = 3$$

$$d_{13} = c_{13} - (u_1 + v_3) = 10 - (2 + 2) = 6$$

$$d_{14} = c_{14} - (u_1 + v_4) = 9 - (2 + 0) = 7$$

$$d_{21} = c_{21} - (u_2 + v_1) = 9 - (2 + 3) = 4$$

$$d_{22} = c_{22} - (u_2 + v_2) = 7 - (2 + 4) = 1$$

$$d_{34} = c_{34} - (u_3 + v_4) = 15 - (0 + 0) = 15$$

Tabulka 48 Znázornění nepřímých sazeb

	Odb A	OdbB	Odb C	OdbD	Zásoby	u_i
D 1	8 [3]	6 (20)	10 [6]	9 [15]	20	2
D 2	9 [4]	7 [1]	4 (15)	2 (15)	30	2
D 3	3 (10)	4 (5)	2 (5)	15 [15]	20	0
Poptávka	10	25	20	15		
v_j	3	4	2	0		

(zdroj: vlastní)

Protože všechny hodnoty $d_{ij} \geq 0$, bylo dosaženo konečného optimálního řešení.

Tabulka 49 Optimální řešení

	Odb A	OdbB	Odb C	OdbD	Zásoby
D 1	8	6 (20)	10	9 [15]	20
D 2	9	7	4 (15)	2 (15)	30
D 3	3 (10)	4 (5)	2 (5)	15	20
Poptávka	10	25	20	15	

(zdroj: vlastní)

Minimální celkové náklady na přepravu jsou:

$$6 \times 20 + 4 \times 15 + 2 \times 15 + 3 \times 10 + 4 \times 5 + 2 \times 5 = 270 \text{ Kč.}$$

II PRAKTICKÁ ČÁST

4 CÍL A PŘEHLED PRAKTICKÉ ČÁSTI

4.1.1 Cíl praktické části

Cílem praktické části této bakalářské práce je navrhnout a vyřešit relativně obtížné a prakticky zaměřené příklady pro vybrané optimalizační problémy, konkrétně pro přiřazovací úlohu a transportní úlohu. Tyto příklady budou řešeny pomocí třech dostupných metod, včetně:

- vlastního algoritmu implementovaného v programovacím jazyce Python
- využití existujících knihoven v programovacím jazyce Python
- řešení prostřednictvím nástrojů dostupných v programu Microsoft Excel

Následně budou dosažené výsledky porovnány z hlediska výpočetního času. Tímto způsobem bude možné posoudit vhodnost jednotlivých přístupů pro praktické využití.

5 NÁVRH OPTIMALIZAČNÍCH PROBLÉMŮ

5.1 Přiřazovací úloha

Zadaní úlohy:

Společnost zabývající se logistikou má k dispozici 10 řidičů a 10 nákladních vozidel.

Cílem je přiřadit každému řidiči právě jedno vozidlo tak, aby celkový čas přepravy byl co nejmenší. Každé přiřazení řidiče k vozidlu má definovaný čas doručení zásilky, který závisí na následujících faktorech:

- Zkušenosti řidiče s daným typem vozidla.
- Technická náročnost vozidla.
- Dostupnost ve směně – někteří řidiči mohou řídit pouze dopoledne, zatímco některá vozidla jsou dostupná pouze v odpoledních hodinách.
- Licenční omezení – někteří řidiči nemají oprávnění řídit určitá vozidla, proto jsou tato přiřazení zakázána.

Tyto faktory jsou zapracovány do matice nákladů, kde každý prvek je odhadovaný čas (v minutách), za který řidič pomocí vozidla doručí zásilku. Pokud je přiřazení zakázané (např. kvůli licenci), hodnota je nastavena na velmi vysoké číslo (9999), aby nebylo algoritmem vybráno.

Matice nákladů:

Tabulka 50 Matice nákladů

Řidiče	Vozidla									
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
R1	42	35	9999	41	30	39	50	9999	90	70
R2	56	60	58	63	9999	32	87	91	25	9999
R3	38	40	35	9999	45	31	28	57	9999	73
R4	59	33	9999	67	51	9999	39	48	42	37
R5	61	54	60	58	9999	43	9999	77	36	44
R6	47	42	40	45	33	36	9999	9999	29	38
R7	52	9999	63	9999	49	47	60	54	9999	41
R8	9999	49	56	62	40	9999	45	9999	33	60
R9	50	9999	9999	53	47	44	52	58	9999	39
R10	45	46	9999	50	9999	48	46	9999	41	9999

(zdroj: vlastní)

5.2 Dopravní problém

Zadání úlohy:

Logistická společnost má k dispozici 7 skladů (S1-S7), ze kterých je třeba dopravit zboží do 9 prodejen (P1-P9). Každý sklad má omezené množství zboží, které může odeslat a každá prodejna má specifický poptávku po zboží. Náklady na dopravu jedné jednotky zboží z každého skladu do každé prodejny jsou známy.

Cílem je najít řešení, které:

- pokryje všechny požadavky prodejen
- nepřekročí kapacity skladů
- minimalizuje celkové dopravní náklady

Data:

- Zásoby ve skladech: 100, 80, 120, 90, 110, 95, 105
- Poptávka v prodejnách: 60, 85, 75, 55, 90, 70, 65, 100, 70

Nákladová matice:

Tabulka 51 Nákladová matice

	P1	P2	P3	P4	P5	P6	P7	P8	P9	Zásoby
S1	8	6	10	9	7	12	14	11	13	100
S2	9	7	12	8	6	13	15	10	14	80
S3	14	10	9	12	11	13	16	8	9	120
S4	10	9	11	14	12	88	9	13	15	90
S5	11	13	8	7	10	9	12	15	14	110
S6	13	12	14	10	8	7	9	11	16	95
S7	12	11	13	9	10	14	8	6	7	105
Poptávka	60	85	75	55	90	70	65	100	70	

(zdroj: vlastní)

6 POUŽITÉ NÁSTROJE

6.1 Python

Python je vysokoúrovňový, interpretovaný programovací jazyk, který si získal obrovskou popularitu zejména díky své jednoduché syntaxi, čitelnosti a univerzálnosti. V porovnání s jinými jazyky, jako je C++ nebo Java, umožňuje Python velmi rychlý vývoj softwaru a prototypování algoritmů. Díky tomu je velmi oblíbený v akademickém prostředí i v průmyslové praxi, zejména v oblastech jako je vědecké programování, strojové učení, analýza dat, numerické simulace či vývoj webových aplikací. [22]

6.1.1 Knihovny v Python

Knihovny představují sadu předem napsaných funkcí, tříd a modulů, které rozšiřují možnosti programovacího jazyka a umožňují rychlejší a efektivnější řešení specifických úloh. Použitím knihoven lze výrazně zkrátit čas potřebný k implementaci složitých algoritmů, protože vývojář nemusí psát vše od začátku, ale může využít již ověřené a optimalizované nástroje. V prostředí Pythonu existuje rozsáhlý ekosystém open-source knihoven, které pokrývají široké spektrum úloh a které je možné upravit podle potřeby.[23]

6.1.2 Použité knihovny

V rámci této práce byly využity především knihovny **NumPy** a **SciPy**, které poskytují nástroje pro efektivní práci s maticemi, výpočty a řešeními optimalizačních problémů. Zejména modul **scipy.optimize** nabízí širokou škálu optimalizačních algoritmů – mezi nimi i funkci **linear_sum_assignment**, která řeší úlohy přiřazení. Dále byla využita knihovna **time** pro měření výpočetního času jednotlivých řešení, což umožnilo porovnat výkonnost vlastní implementaci algoritmů a řešení pomocí knihoven. [24]

6.2 Excel

Microsoft Excel je univerzální tabulkový procesor vyvinutý společností Microsoft, který se celosvětově používá pro organizaci a správu dat. Mezi jeho hlavní funkce patří zadávání dat, práce s funkcemi a vzorci, a tvorba grafů pro vizualizaci informací. Excel je široce využíván v různých oborech a umožňuje uživatelům efektivně analyzovat a zpracovávat data díky své flexibilitě a rozsáhlým nástrojům. [25]

6.2.1 Excel Solver

Solver (dále Řešitel) je doplněk Excelu určený k řešení matematických úloh prostřednictvím optimalizace výsledků. Jeho hlavní výhodou je schopnost určovat maximální nebo minimální hodnoty buněk tím, že mění hodnoty ostatních buněk a vyhledávání optimalizovaných cílů pro lineární i nelineární modely.

Řešitel je možné použít například při plánování sortimentu výrobků v určitém časovém období s cílem maximalizovat zisk, při organizaci pracovní síly nebo při řešení problémů s dopravou. [26]

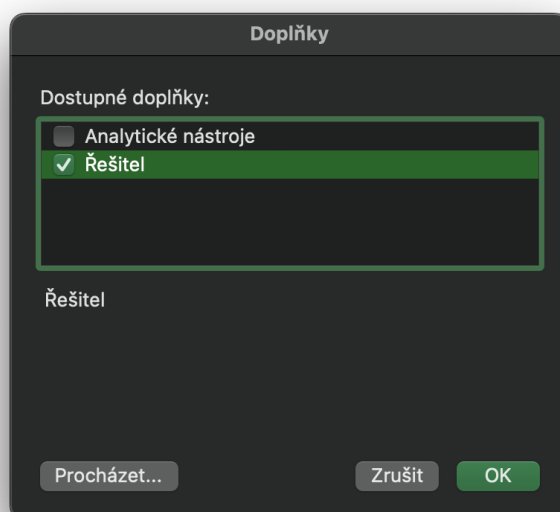
6.2.2 Jak zapnout Excel Solver

Tato podkapitola pomůže uživatelům pochopit, jak najít a aktivovat doplněk Solver Excel.

Postup:

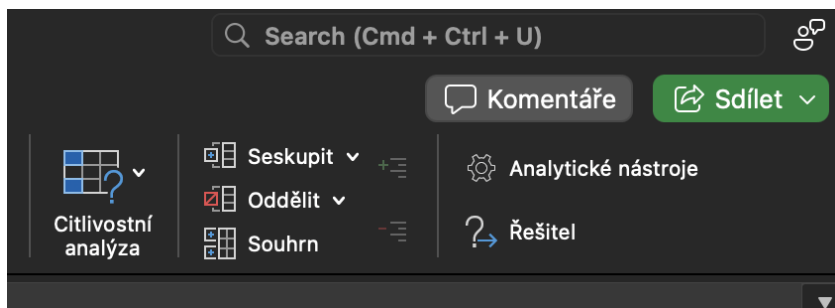
1. V nabídce **Nástroje** vyberte **Doplňky Excelu**.
2. V poli **Dostupné doplňky** zaškrtněte políčko **Doplňěk Řešitel** a klikněte na OK.

Obrázek 3 Doplňky



- Pokud není doplněk **Řešitel** v seznamu **Doplňky** k dispozici uveden, klikněte na tlačítko **Procházet** a vyhledejte jej.

- Jestliže není doplněk **Řešitel** na počítači aktuálně nainstalovaný, klikněte v dialogovém okně na tlačítko **Ano** a nainstalujte ho tak.
3. Po zavedení doplňku **Řešitel** bude k dispozici na kartě **Data** -> tlačítko **Řešitel**. [27]



Obrázek 4 Umištění řešitele

7 IMPLEMENTACE ALGORITMŮ V PYTHON

V rámci této kapitoly se rozebereme postup implementaci algoritmů řešení optimalizačních úloh v programovacím jazyku Python.

7.1 Přiřazovací úloha

Pro řešení praktického příkladu přiřazovací úlohy byla v jazyce Python implementována Maďarská metoda. V této podkapitole představíme nejdůležitější funkce kódu, přičemž celý kód je k dispozici v příloze.

Mezi nejdůležitější funkce patří:

1. Funkce redukce matice, která hledá nejmenší prvky a provádí řádkovou a sloupcovou redukci:

```
def redukce(matrix):  
    n = len(matrix)  
    for i in range(n):  
        min_val = min(matrix[i])  
        for j in range(n):  
            matrix[i][j] -= min_val  
  
    for j in range(n):  
        col = [matrix[i][j] for i in range(n)]  
        min_val = min(col)  
        for i in range(n):  
            matrix[i][j] -= min_val  
    return matrix
```

2. Inicializace pomocné struktury – matice hvězdiček a čárek a vektory pokrytí řádků a sloupců.

```
def initialize_matrices(n):  
    row_covered = [False] * n  
    col_covered = [False] * n  
    star_matrix = [[False]*n for _ in range(n)]  
    prime_matrix = [[False]*n for _ in range(n)]  
    return row_covered, col_covered, star_matrix, prime_matrix
```

Tato funkce připravuje základní datové struktury, které algoritmus potřebuje pro správné sledování stavu při hledání optimálního přiřazení, vytváří a vrací seznamy pro označení zakrytých řádků a sloupců (`row_covered`, `col_covered`) a dvě logické matice (`star_matrix`, `prime_matrix`), které se používají k označování hvězdičkami a primami během řešení úlohy.

3. Funkce označení nezakrytých nul hvězdičkami a zakrytí odpovídajících řádků a sloupců

```
def star_zeros(matrix, row_covered, col_covered, star_matrix):
    n = len(matrix)
    for i in range(n):
        for j in range(n):
            if matrix[i][j] == 0 and not row_covered[i] and not
col_covered[j]:
                star_matrix[i][j] = True
                row_covered[i] = True
                col_covered[j] = True
    clear_covers(row_covered, col_covered)
```

Funkce označí první nezakryté nuly hvězdičkami a zakryje odpovídající řádky a sloupce, aby zabránila překryvu. Poté krytí odstraní. Tento krok vytváří počáteční přiřazení nul pro další zpracování algoritmu.

4. Funkce přepínání označení na cestě pro aktualizaci přiřazení

```
def augment_path(path, star_matrix):
    for r, c in path:
        star_matrix[r][c] = not star_matrix[r][c]
```

Funkce přepíná označení hvězdiček a čárek na dané cestě, čímž aktualizuje přiřazení. Tento krok je klíčový pro postup k optimálnímu řešení.

5. Hlavní funkce algoritmu

```
def hungarian_method(matrix):
    n = len(matrix)
    INF = float('inf')
    original_matrix = copy.deepcopy(matrix)

    redukce(matrix)
```

```
row_covered, col_covered, star_matrix, prime_matrix =
initialize_matrices(n)

star_zeros(matrix, row_covered, col_covered, star_matrix)
clear_covers(row_covered, col_covered)
cover_columns_with_starred_zero(star_matrix, col_covered)

while sum(col_covered) < n:
    row, col = find_a_zero(matrix, row_covered, col_covered)
    while row == -1:
        min_val = INF
        for i in range(n):
            if not row_covered[i]:
                for j in range(n):
                    if not col_covered[j] and matrix[i][j] <
min_val:
                        min_val = matrix[i][j]

        for i in range(n):
            if row_covered[i]:
                for j in range(n):
                    matrix[i][j] += min_val

        for j in range(n):
            if not col_covered[j]:
                for i in range(n):
                    matrix[i][j] -= min_val

    row, col = find_a_zero(matrix, row_covered, col_covered)

    prime_matrix[row][col] = True
    star_col = find_in_line(star_matrix, row, search_row=True)

    if star_col != -1:
        row_covered[row] = True
        col_covered[star_col] = False
    else:
        path = [(row, col)]
        done = False
        while not done:
```

```

        star_row = find_in_line(star_matrix, path[-1][1],
search_row=False)
        if star_row != -1:
            path.append((star_row, path[-1][1]))
            prime_col = find_in_line(prime_matrix, path[-
1][0], search_row=True)
            path.append((path[-1][0], prime_col))
        else:
            done = True

    augment_path(path, star_matrix)
    clear_covers(row_covered, col_covered)
    clear_primes(prime_matrix)
    cover_columns_with_starred_zero(star_matrix, col_covered)

result = [-1]*n
for i in range(n):
    for j in range(n):
        if star_matrix[i][j]:
            result[i] = j
            break

total_cost = sum(original_matrix[i][result[i]] for i in range(n))
return result, total_cost

```

Tato funkce implementuje maďarský algoritmus pro nalezení optimálního přiřazení v nákladové matici. Nejprve upraví matici odečtením minimálních hodnot v řádcích a sloupcích, čímž vytvoří nulové prvky. Poté inicializuje pomocné struktury pro označování zakrytých řádků, sloupců a nul. Označí hvězdičkami vhodné nuly, zakryje odpovídající sloupce a v hlavní smyčce hledá nepokryté nuly, které postupně primuje a podle potřeby upravuje matici odečtením a přičtením nejmenších hodnot, aby se vytvořily nové nuly. Pomocí augmentační cesty aktualizuje přiřazení hvězdiček, dokud nejsou všechny sloupce pokryty. Nakonec vrátí optimální přiřazení a jeho celkovou cenu podle původní matice. Tento postup odpovídá krokům maďarského algoritmu pro minimalizaci nákladů při přiřazování.

Níže v tabulce je uveden krátký popis ostatních funkcí:

Tabulka 52 Popis ostatních funkcí kodu

Název funkce	Popis
<code>cover_columns_with_starred_zero(star_matrix, col_covered)</code>	Zakrytí sloupců obsahujících označené nuly (hvězdičky).
<code>find_a_zero</code>	Hledání nekrytých nul – příprava pro další operace (primování).
<code>find_in_line111</code>	Najde index označené nuly v daném řádku nebo sloupci.
<code>find_star_in_col(star_matrix, col)</code>	Pomocná funkce – hledání hvězdičky ve sloupci.
<code>find_prime_in_row(prime_matrix, row)</code>	Hledání primované nuly (označené čárkou) v řádku – používá se při konstrukci alternativní cesty.
<code>star_zeros</code>	Funkce označí nezakryté nuly hvězdičkami a zakryje příslušné řádky a sloupce, aby zabránila jejich opětovnému použití.
<code>clear_covers</code>	Vymazání všech zakrytí řádků a sloupců – příprava na novou iteraci
<code>clear_primes</code>	Vymazání všech čárek z matice – reset značení.

7.1.1 Použití knihovny pro řešení přiřazovací úlohy

Pro výpočet řešení problému přiřazení byla použita knihovna SciPy a její modul optimize, ze kterého byl importován `linear_sum_assignment`.

Níže je uveden kompletní kód pro řešení dané úlohy pomocí knihovny:

1. Nejprve byla definována nákladová matice

```
cost_matrix = [  
    [42, 35, 9999, 41, 30, 39, 50, 9999, 90, 70],  
    [56, 60, 58, 63, 9999, 32, 87, 91, 25, 9999],  
    [38, 40, 35, 9999, 45, 31, 28, 57, 9999, 73],  
    [59, 33, 9999, 67, 51, 9999, 39, 48, 42, 37],  
    [61, 54, 60, 58, 9999, 43, 9999, 77, 36, 44],  
    [47, 42, 40, 45, 33, 36, 9999, 9999, 29, 381],  
    [52, 9999, 63, 9999, 49, 47, 60, 54, 9999, 41],  
    [9999, 49, 56, 62, 40, 9999, 45, 9999, 33, 60],  
    [50, 9999, 9999, 53, 47, 44, 52, 58, 9999, 39],  
    [45, 46, 9999, 50, 9999, 48, 46, 9999, 41, 9999]  
]
```

2. Následně byla úloha vyřešena a zároveň byla změřena doba výpočtu

```
start_scipy = time.time()  
row_ind, col_ind = linear_sum_assignment(cost_matrix)  
end_scipy = time.time()  
total_cost_scipy = sum(cost_matrix[i][col_ind[i]] for i in  
    range(len(row_ind)))
```

7.2 Dopravní problém

Pro řešení praktického příkladu dopravního problému bylo v jazyce Python implementováno nalezení základního přípustného řešení pomocí metody severozápadního rohu. Následně byla implementována metoda MODI za účelem zlepšení řešení a nalezení řešení optimálního. V této podkapitole představíme nejdůležitější funkce kódu, přičemž celý kód je uveden v příloze.

Mezi nejdůležitější funkce patří:

1. Realizace metody severozápadního rohu

```
def northwest_corner_method(supply, demand):  
    supply = supply[:]  
    demand = demand[:]  
    m = len(supply)  
    n = len(demand)
```

```

allocation = [[0] * n for _ in range(m)]

i = 0
j = 0
while i < m and j < n:
    qty = min(supply[i], demand[j])
    allocation[i][j] = qty
    supply[i] -= qty
    demand[j] -= qty
    if supply[i] == 0:
        i += 1
    elif demand[j] == 0:
        j += 1

return allocation

```

Funkce vytvoří počáteční alokaci zásob mezi zdroji a poptávkou tak, že postupuje od levého horního (severozápadního) rohu matice a postupně přiděluje množství podle dostupné nabídky a poptávky, dokud není celá matice vyplněna.

2. Funkce výpočtu potencialů u_i a v_j

```

def calculate_potentials(allocation, cost_matrix):
    m = len(allocation)
    n = len(allocation[0])
    u = [None] * m
    v = [None] * n
    u[0] = 0

    changed = True
    while changed:
        changed = False
        for i in range(m):
            for j in range(n):
                if allocation[i][j] > 0:
                    if u[i] is not None and v[j] is None:
                        v[j] = cost_matrix[i][j] - u[i]
                        changed = True
                    elif u[i] is None and v[j] is not None:
                        u[i] = cost_matrix[i][j] - v[j]
                        changed = True

    if any(x is None for x in u) or any(x is None for x in v):
        raise Exception()

    return u, v

```

3. Funkce výpočtu nepřímých sazb d_{ij}

```

def calculate_deltas(allocation, cost_matrix, u, v):
    m = len(allocation)
    n = len(allocation[0])
    deltas = [[None] * n for _ in range(m)]
    for i in range(m):
        for j in range(n):
            if allocation[i][j] == 0:
                deltas[i][j] = cost_matrix[i][j] - (u[i] + v[j])
    return deltas

```

4. Funkce vyhledávání cyklu v alokaci pro zlepšení řešení

```

def find_cycle(allocation, d_raw):
    d = parse_d(d_raw)
    rows = len(allocation)
    cols = len(allocation[0])
    min_val = float('inf')
    start_pos = None
    for i in range(rows):
        for j in range(cols):
            if allocation[i][j] == 0 and d[i][j] is not None and
d[i][j] < min_val:
                min_val = d[i][j]
                start_pos = (i, j)

    if start_pos is None:
        return None

    basis = []
    for i in range(rows):
        for j in range(cols):
            if allocation[i][j] > 0:
                basis.append((i, j))
    basis.append(start_pos)

    graph = {}
    for cell in basis:
        i, j = cell
        neighbors = []
        for other in basis:
            if other != cell:
                if other[0] == i or other[1] == j:
                    neighbors.append(other)
        graph[cell] = neighbors

    def dfs(path):
        current = path[-1]
        if len(path) > 3 and current == start_pos and len(path) % 2
== 0:
            return path
        for neighbor in graph[current]:
            if neighbor in path:
                if neighbor == start_pos and len(path) > 3 and
len(path) % 2 == 0:
                    return path + [neighbor]
                continue
            if len(path) >= 2:
                prev = path[-2]
                prev_diff_row = (current[0] != prev[0])
                curr_diff_row = (neighbor[0] != current[0])
                if prev_diff_row == curr_diff_row:
                    continue
            res = dfs(path + [neighbor])
            if res:
                return res
        return None

    cycle = dfs([start_pos])
    if cycle is None:
        return None

```

```

minus_positions = []
for idx in range(1, len(cycle) - 1, 2):
    i, j = cycle[idx]
    minus_positions.append((i, j))

min_alloc = min(allocation[i][j] for i, j in minus_positions)

for idx in range(len(cycle) - 1):
    i, j = cycle[idx]
    if idx % 2 == 0:
        allocation[i][j] += min_alloc
    else:
        allocation[i][j] -= min_alloc

return cycle

```

Funkce hledá cyklus v alokační matici pro zlepšení řešení. Nejprve najde pozici s nejnižší hodnotou d_{ij} , která není alokována. Poté vytvoří graf sousedních alokačních buněk. Rekurzivně prohledává tento graf (DFS) a hledá uzavřený cyklus se střídavými pohyby mezi řádky a sloupci. Když cyklus najde, upraví alokace podél cyklu přičtením a odečtením minimální alokace na odpovídajících pozicích. Nakonec vrátí nalezený cyklus.

5. Hlavní funkce algoritmu

```

def modi_method(supply, demand, cost_matrix):
    supply, demand, cost_matrix = is_balanced(supply, demand,
        cost_matrix)

    print("\n Nákladová matice: ")
    for row in cost_matrix:
        print(" ", " ".join(f"{val:>3}" for val in row))

    allocation = northwest_corner_method(supply, demand)

    print("\n Základní přípustné řešení: ")
    for row in allocation:
        print(" ", " ".join(f"{val:>3}" for val in row))

    if check_degeneracy(allocation):
        fix_degeneracy(allocation, supply, demand)

    while True:
        u, v = calculate_potentials(allocation, cost_matrix)
        deltas = calculate_deltas(allocation, cost_matrix, u, v)
        entering = find_entering_variable_min(deltas)
        if entering is None or deltas[entering[0]][entering[1]] >=
0:
            break
        else:
            d_raw = [[str(val) if val is not None else '-' for val
in row] for row in deltas]
            find_cycle(allocation, d_raw)
            if check_degeneracy(allocation):
                fix_degeneracy(allocation, supply, demand)

    for i in range(len(allocation)):
        for j in range(len(allocation[0])):
            allocation[i][j] = round(allocation[i][j])

```

```

allocation, cost_matrix = remove_dummy(allocation, cost_matrix,
supply, demand)

print("\n Optimální řešení:")
for row in allocation:
    print("  ", "  ".join(f"{val:>3}" for val in row))

total_cost = calculate_total_cost(allocation, cost_matrix)
print(f"\n Celkové minimální náklady: {total_cost}")

return allocation

```

Hlavní funkcí, která koordinuje jednotlivé kroky řešení dopravní úlohy metodou MODI. Nejprve zajistí vyrovnání nabídky a poptávky, vytvoří počáteční základní řešení, opraví degeneraci a pak opakovaně počítá potenciály a delty, aby našla cestu ke zlepšení řešení pomocí cyklů. Proces pokračuje, dokud není dosaženo optimálního řešení s minimálními náklady. Nakonec upraví a vypíše výslednou alokaci a celkové náklady.

Níže v tabulce je uveden krátký popis ostatních funkcí:

Tabulka 53 Popis ostatních funkcí kodu

Název funkce	Popis
is_balanced	Kontroluje, zda je nabídka a poptávka vyrovnaná; přidá fiktivního dodavatele nebo odběratele, pokud je potřeba
find_entering_variable_min	Najde pozici s nejmenší zápornou deltou pro zlepšení řešení
parse_d	Převede surová data delt na číselný formát, nahradí pomlčky hodnotou None
check_cycle	Zkontroluje, zda v základních buňkách vzniká cyklus při přidání nové buňky
check_degeneracy	Ověří, zda je řešení degenerované
get_basis_cells	Určí základní buňky (s nenulovou alokací)
is_connected	Zkontroluje, zda jsou všechny základní buňky spojené (pro správnost řešení)

<code>fix_degeneracy</code>	Opraví degeneraci přidáním malých hodnot do nulových buněk, aniž by vznikl cyklus
<code>calculate_total_cost</code>	Vypočítá celkové přepravní náklady podle aktuální alokace
<code>remove_dummy</code>	Odstraní fiktivní řádky nebo sloupce, pokud byly přidány pro vyrovnání problému

(zdroj:vlastní)

7.2.1 Použití knihovny pro řešení dopravního problému

Pro výpočet optimálního přepravního plánu byla použita knihovna SciPy a její modul `optimize`, přičemž k řešení byl využit algoritmus SLSQP (Sequential Least Squares Programming).

Níže je uveden kompletní kod pro řešení dan úlohy pomocí knihovny:

1. Nejprve byly definovány vstupní parametry problému — nabídka, poptávka a nákladová matice, které byly převedeny na pole NumPy pro efektivní numerické výpočty a také byla určena dimenze problému (m, n), což odpovídá počtu dodavatelů a odběratelů.

```
supply = [100, 80, 120, 90, 110, 95, 105]
demand = [60, 85, 75, 55, 90, 70, 65, 100, 70]
cost_matrix = [
    [8, 6, 10, 9, 7, 12, 14, 11, 13],
    [9, 7, 12, 8, 6, 13, 15, 10, 14],
    [14, 10, 9, 12, 11, 13, 16, 8, 7],
    [10, 9, 11, 14, 12, 8, 9, 13, 15],
    [11, 13, 8, 7, 10, 9, 12, 15, 14],
    [13, 12, 14, 10, 8, 7, 9, 11, 16],
    [12, 11, 13, 9, 10, 14, 8, 6, 7]
]
```

```
supply = np.array(supply)
demand = np.array(demand)
cost = np.array(cost_matrix)
m, n = cost.shape
```

2. Byla stanovena cílová funkce minimalizující celkové náklady na základě přepravovaných množství a nákladů. Také pro algoritmus bylo nastaveno počáteční odhadované řešení (x_0), které obsahuje samé nuly.

```
def objective(x):
    return np.sum(cost.flatten() * x)

x0 = np.zeros(m * n)
```

3. Byla formulována omezení:

- Omezení nabídky zajišťující, že dodávky nepřesáhnou dostupné zásoby každého dodavatele.
- Omezení poptávky garantující, že poptávka odběratelů bude zcela uspokojena.

```
def supply_constraint(i):
    return lambda x: supply[i] - np.sum(x[i*n:(i+1)*n])
def demand_constraint(j):
    return lambda x: np.sum([x[i*n + j] for i in range(m)]) - demand[j]
```

4. Proměnné byly omezeny na nezáporné hodnoty, protože množství přepravovaného zboží nemůže být záporné

```
bounds = [(0, None)] * (m * n)
```

5. Byly stanoveny omezení, které zajišťují, že množství dodané z každého zdroje nepřekročí jeho kapacitu (nerovnostní omezení) a že poptávka každého odběratele bude přesně splněna (rovnostní omezení)

```
constraints = []
for i in range(m):
    constraints.append({'type': 'ineq', 'fun':
        supply_constraint(i)})
for j in range(n):
    constraints.append({'type': 'eq', 'fun': demand_constraint(j)})
```

6. Pomocí funkce *minimize* a metody SLSQP byla provedena optimalizace pro nalezení minimálních celkových nákladů přepravy. Po úspěšném dokončení byly vytištěny optimální přepravní plány, celkové náklady a čas výpočtu.

```
start_scipy = time.time()
res = minimize(objective, x0, bounds=bounds,
    constraints=constraints, method='SLSQP')
end_scipy = time.time()
```

8 ŘEŠENÍ PROBLEMŮ POMOCÍ EXCEL

Zde bude popsán postup řešení optimalizačních problémů v excel.

8.1 Přiřazovací úloha

8.1.1 Příprava vstupních dat

Jako první krok vstupní matice nákladů byla přenesena do excelu pro následující manipulace a řešení daného příkladu.

		Auta									
		A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
Řidiče	B1	42	35	9999	41	30	39	50	9999	90	70
	B2	56	60	58	63	9999	32	87	91	25	9999
	B3	38	40	35	9999	45	31	28	57	9999	73
	B4	59	33	9999	67	51	9999	39	48	42	37
	B5	61	54	60	58	9999	43	9999	77	36	44
	B6	47	42	40	45	33	36	9999	9999	29	38
	B7	52	9999	63	9999	49	47	60	54	9999	41
	B8	9999	49	56	62	40	9999	45	9999	33	60
	B9	50	9999	9999	53	47	44	52	58	9999	39
	B10	45	46	9999	50	9999	48	46	9999	41	9999

Obrázek 5 Matice nákladů

8.1.2 Tvorba modelu pro Solver

V rámci této části byla přidána rozhodovací matice, se kterou bude Solver pracovat a měnit ji, aby optimalizoval výsledek.

		Decision variables										Row Sum	Supply
		A1	A2	A3	A4	A5	A6	A7	A8	A9	A10		
B1	0	0	0	0	1	0	0	0	0	0	0	1 =	1
B2	0	0	0	0	0	0	1	0	0	0	0	1 =	1
B3	0	0	0	0	0	0	0	1	0	0	0	1 =	1
B4	0	1	0	0	0	0	0	0	0	0	0	1 =	1
B5	0	0	0	0	0	0	0	0	0	1	0	1 =	1
B6	0	0	0	1	0	0	0	0	0	0	0	1 =	1
B7	0	0	0	0	0	0	0	0	1	0	0	1 =	1
B8	0	0	0	0	0	1	0	0	0	0	0	1 =	1
B9	0	0	0	0	0	0	0	0	0	0	1	1 =	1
B10	1	0	0	0	0	0	0	0	0	0	0	1 =	1
Column Sum	1	1	1	1	1	1	1	1	1	1	1		
Demand	1	1	1	1	1	1	1	1	1	1	1		

Obrázek 6 Rozhodovací matice

Pro korektní výpočet matice byla upravena z původního tvaru, tedy bunky obsahují pouze hodnoty 0 nebo 1 v závislosti na tom, jestli konkrétní přiřazení nastane (1 – ano, 0 – ne), jelikož jedná se o kombinatorický problém.

Dálší uprava se týká přidání nových sloupců, konkrétně Row Sum, Supply, Column Sum, Demand. Sloupec Row Sum zajišťuje, že každý pracovník je přiřazen právě k jednomu úkolu. Sloupec Column Sum zajišťuje, že každý úkol je přiřazen právě

k jednomu pracovníkovi. Sloupce Supply a Demand jsou omezení a rovnají se 0, aby Solver vynutil, že:

- Z každého řádku bude vybrána právě jedna jednička (1).
- V každém sloupci bude právě jedna jednička (1).

Tím vznikne jednoznačné a úplné přiřazení, bez duplicit a bez vynechání.

Pro výpočet celkových nákladů přiřazení byla vytvořena buňka s cílovou funkcí, která obsahovala vzorec =SOUČIN.SKALÁRNÍ(). Tento vzorec násobí nákladovou matici a rozhodovací matici s hodnotami 0/1. Solver měl za úkol minimalizovat tuto hodnotu, aby našel optimální přiřazení s nejnižšími náklady.

8.1.3 Řešení pomocí makra

Jelikož součástí mého zadání bylo nejen nalezení řešení pomocí Solveru, ale i měření doby výpočtu, zvolil jsem použití makra, které tuto funkcionalitu umožňuje. Měření času přímo pomocí Solveru není možné, a proto bylo nutné využít makro.

```
Sub SolveAssignmentProblem()
    Dim startTime As Double
    Dim endTime As Double
    Dim elapsedTime As Double
    Dim Result As Integer
    On Error Resume Next
    Dim ObjectiveValue As Double

    SolverReset
    SolverOk SetCell:="$T$40", MaxMinVal:=2, ValueOf:=0,
    ByChange:="$C$31:$L$40", _
        Engine:=1, EngineDesc:="Simplex LP"
    SolverAdd CellRef:="$C$31:$L$40", Relation:=5,
    FormulaText:="binary"
    SolverAdd CellRef:="$C$41:$L$41", Relation:=2,
    FormulaText:="$C$43:$L$43"
    SolverAdd CellRef:="$M$31:$M$40", Relation:=2,
    FormulaText:="$O$31:$O$40"

    SolverOptions AssumeNonNeg:=True
    startTime = Timer
    Result = SolverSolve(UserFinish:=True)
    endTime = Timer
```

```

elapsedTime = endTime - startTime
ObjectiveValue = Range("T40").Value
If Result = 0 Then
    MsgBox "Řešení nalezeno!" & vbCrLf & _
        "Hodnota cílové funkce: " & Format(ObjectiveValue,
"0.000") & vbCrLf & _
        "Doba výpočtu: " & Format(elapsedTime, "0.000") & "
sekund.", vbInformation
Else
    MsgBox "Error: " & Result & ". Time: " & Format(elapsedTime,
"0.000") & " seconds.", vbExclamation
End If
End Sub

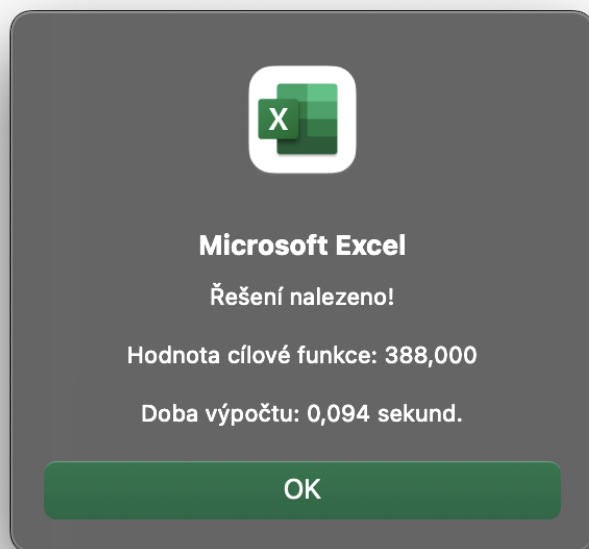
```

Tabulka 54 Popis hlavních příkazu ve VBA v makro

Příkaz VBA	Popis
SolverReset	Resetuje všechna předchozí nastavení doplňku Solver.
SolverOk	Nastaví parametry úlohy: cílovou buňku, proměnné, typ optimalizace.
SetCell:="\$T\$40"	Určuje buňku s cílovou funkcí, která má být minimalizována.
ByChange:="\$C\$31:\$L\$40"	Definuje oblast proměnných (matice přiřazení), které může Solver měnit.
SolverAdd	Přidává omezení pro proměnné (např. binární hodnoty, součty v řádcích/sloupcích).
Relation:=5, FormulaText:="binary"	Určuje, že proměnné mohou nabývat pouze hodnot 0 nebo 1 (binární).
SolverOptions AssumeNonNeg:=True	Předpokládá, že všechny proměnné jsou nezáporné.
Timer	Vestavěná funkce VBA pro měření času před a po spuštění řešitele.

zdroj:vlastní

Po spuštění makra vidíme messageBox, ve kterém jsou uvedeny hodnota účelové funkce a doba výpočtu.



Obrázek 7 Nalezené řešení

8.2 Dopravní problém

8.2.1 Příprava vstupních dat

Jako první krok vstupní matice nákladů byla přenesena do excelu pro následující manipulace a řešení daného příkladu.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	Zásoby
S1	8	6	10	9	7	12	14	11	13	100
S2	9	7	12	8	6	13	15	10	14	80
S3	14	10	9	12	11	13	16	8	7	120
S4	10	9	11	14	12	8	9	13	15	90
S5	11	13	8	7	10	9	12	15	14	110
S6	13	12	14	10	8	7	9	11	16	95
S7	12	11	13	9	10	14	8	6	7	105
Poptávka	60	85	75	55	90	70	65	100	70	

Obrázek 8 Matice nákladů

8.2.2 Tvorba modelu pro Solver

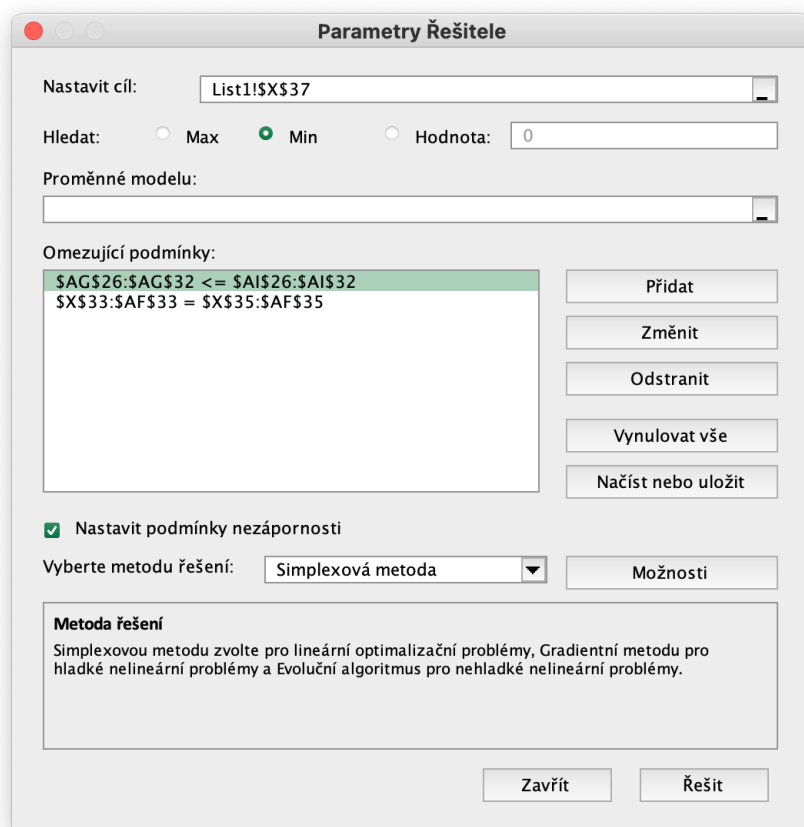
V rámci této části byla přidána rozhodovací matice, se kterou bude Solver pracovat a měnit jí, aby optimalizoval výsledek.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	Zásoby	Znak	Zásoby
S1										0	<=	100
S2										0	<=	80
S3										0	<=	120
S4										0	<=	90
S5										0	<=	110
S6										0	<=	95
S7										0	<=	105
Poptávka	0	0	0	0	0	0	0	0	0	0		
Znak	=	=	=	=	=	=	=	=	=	=		
Poptávka	60	85	75	55	90	70	65	100	70			

Obrázek 9 Příprava rozhodovací matice

Než jsem spustil Řešitel, musel jsem v Excelu připravit prázdnou matici, která bude sloužit jako základ pro optimalizaci. Vytvořil jsem prázdnou tabulku o 7 řádcích pro dodavatele a 9 sloupcích pro odběratele, která slouží Řešiteli jako prostor pro výpočty a bude naplněna optimálními hodnotami přeprav. Do posledního sloupce "Zásoby" jsem zadal maximální kapacitu každého dodavatele - například S1 má k dispozici 100 jednotek, S2 80 jednotek atd. V dolním řádku "Poptávka" jsem pak vyplnil požadavky jednotlivých odběratelů, kde P1 potřebuje 60 jednotek, P2 85 jednotek a tak dále, a to za účelem určení pravidel, která musí výsledek splňovat.

Klíčové bylo správně nastavit omezení. U dodavatelů jsem pomocí znaku "<=" zajistil, že celkový objem jejich dodávek nikdy nepřekročí jejich kapacitu. Naopak u odběratelů znak "=" zaručuje, že jejich poptávka bude přesně uspokojena.



Obrázek 10 Nastavení Řešitele

Po vytvoření prázdné matice a definování všech omezení jsem přistoupil ke konfiguraci nástroje Řešitel následujícím způsobem:

- Cílová buňka:

Nastavil jsem buňku *List1!\$X\$37* jako cílovou, která obsahuje vzorec pro celkové náklady (funkce SUMPRODUCT mezi maticí přeprav a cenovou maticí). Zvolil jsem možnost minimalizace, protože cílem je najít řešení s nejnižšími náklady.

- Omezující podmínky:

Přidal jsem dva omezení:

$\$AG\$26:\$AG\$32 \leq \$AI\$26:\$AI\$32$ - zajistí, že celkové dodávky od každého dodavatele nepřekročí jeho kapacitu (zásoby).

$\$X\$33:\$AF\$33 = \$X\$35:\$AF\$35$ - garantuje přesné splnění poptávky všech odběratelů.

- Další nastavení:

Zaškrtnul jsem volbu Nastavit podmínky nezápornosti, aby všechny přepravní množství byla kladná nebo nulová.

Jako metodu řešení jsem zvolil Simplexovou metodu, která je optimální pro lineární problémy, jako je dopravní úloha.

Po kontrole všech nastavení jsem klikl na tlačítko Řešit, čímž se spustil výpočet optimálního řešení.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	Zásoby	Znak	Zásoby
S1	15	85	0	0	0	0	0	0	0	100	<=	100
S2	0	0	0	0	0	80	0	0	0	80	<=	80
S3	0	0	20	0	0	0	0	0	70	90	<=	120
S4	45	0	0	0	0	0	0	45	0	90	<=	90
S5	0	0	55	55	0	0	0	0	0	110	<=	110
S6	0	0	0	0	0	10	70	15	0	95	<=	95
S7	0	0	0	0	0	0	0	5	100	105	<=	105
Poptávka	60	85	75	55	90	70	65	100	70			
Znak	=	=	=	=	=	=	=	=	=			
Poptávka	60	85	75	55	90	70	65	100	70			
OFV	4805											

Obrázek 11 Rozhodovací matice

Po spuštění Řešitele byla prázdná matice naplněna konkrétními hodnotami přeprav, které představují optimální řešení. Cílová funkce (OFV) dosahuje hodnoty 4805, což představuje minimální celkové náklady na dopravu při daném rozdělení.

8.2.3 Řešení pomocí makra

Jelikož součástí mého zadání bylo nejen nalezení řešení pomocí Solveru, ale i měření doby výpočtu, zvolil jsem použití makra, které tuto funkcionalitu umožňuje. Měření času přímo pomocí Solveru není možné, a proto bylo nutné využít makro:

```
Sub SolveTransportationProblem()
    Dim startTime As Double
    Dim endTime As Double
    Dim elapsedTime As Double
    Dim result As Integer
    Dim objectiveValue As Double

    On Error Resume Next

    ' Reset řešitele
    SolverReset

    ' Nastavení cíle a rozhodovacích proměnných
```

```
SolverOk SetCell:="$X$37", MaxMinVal:=2, ValueOf:=0,
ByChange:="$X$26:$AF$32", _
    Engine:=1, EngineDesc:="Simplex LP"

' Přidání omezení
SolverAdd CellRef:="$AG$26:$AG$32", Relation:=1,
FormulaText:="$AI$26:$AI$32" ' <=
SolverAdd CellRef:="$X$33:$AF$33", Relation:=2,
FormulaText:="$X$35:$AF$35" ' =
' Nezáporné proměnné
SolverOptions AssumeNonNeg:=True

' Spuštění časovače
startTime = Timer
result = SolverSolve(UserFinish:=True)
endTime = Timer
elapsedTime = endTime - startTime

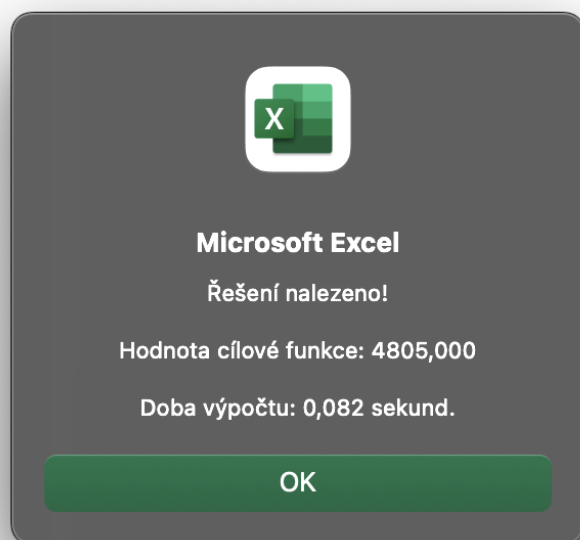
' Hodnota cílové funkce
objectiveValue = Range("$X$37").Value

' Výsledek
If result = 0 Then
    MsgBox "Řešení nalezeno!" & vbCrLf & _
        "Hodnota cílové funkce: " & Format(objectiveValue,
"0.000") & vbCrLf & _
        "Doba výpočtu: " & Format(elapsedTime, "0.000") & "
sekund.", vbInformation
Else
    MsgBox "Chyba: " & result & ". Čas: " & Format(elapsedTime,
"0.000") & " sekund.", vbExclamation
End If
End Sub
```

Tabulka 55 Popis částí kodu VBA

Příkaz VBA	Popis
<code>SolverReset</code>	Resetuje všechna předchozí nastavení doplňku Solver.
<code>SolverOk</code>	Nastaví parametry úlohy: cílovou buňku, proměnné, typ optimalizace.
<code>SetCell:="\$X\$37"</code>	Určuje buňku s cílovou funkcí, která má být minimalizována.
<code>ByChange:="\$X\$26:\$AF\$32"</code>	Definuje oblast proměnných (matice přiřazení), které může Solver měnit.
<code>SolverAdd</code>	Přidává omezení pro proměnné (např. binární hodnoty, součty v řádcích/sloupcích).
<code>Engine:=1, EngineDesc:="Simplex LP"</code>	Používá lineární Simplexový řešič.
<code>SolverOptions AssumeNonNeg:=True</code>	Předpokládá, že všechny proměnné jsou nezáporné.
<code>Timer</code>	Vestavěná funkce VBA pro měření času před a po spuštění řešitele.

Po spuštění makra vidíme `messageBox`, ve kterém jsou uvedeny hodnota účelové funkce a doba výpočtu



Obrázek 12 Nalezený výsledek

9 HODNOCENÍ VÝSLEDKU

Porovnáváme výsledky získané vlastní implementací a s využitím knihovných řešení a tabulkového procesoru Excel.

9.1 Přiřazovací úloha

1. Výsledek dosažený pomocí vlastní implementace algoritmu:

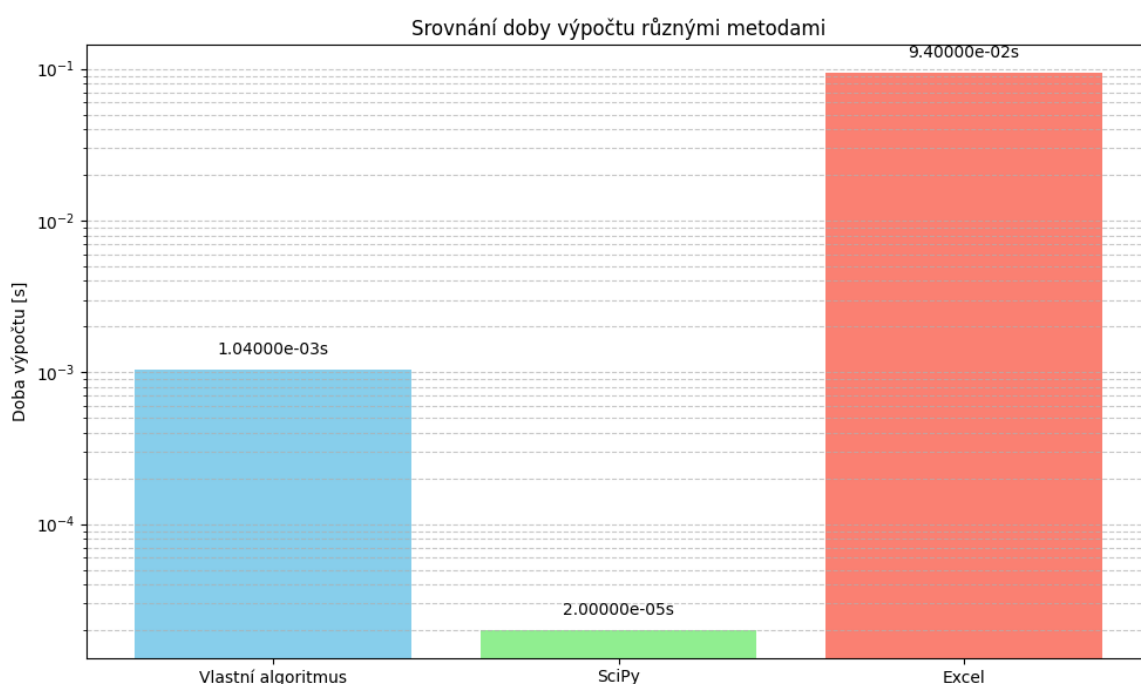
- Doba výpočtu: 0.00104s

2. Výsledek dosažený pomocí knihovny SciPy:

- Doba výpočtu: 0.00002s

3. Výsledek dosažený pomocí Excelu:

- Doba výpočtu: 0.094s



Obrázek 13 Porovnání doby výpočtu

Hodnocení výsledku měření:

Na základě provedeného srovnání můžeme pozorovat výrazné rozdíly v době výpočtu mezi jednotlivými způsoby řešení optimalizační úlohy. Vlastní implementace algoritmu dosáhla výsledku za 0.00104 sekundy, zatímco knihovna SciPy byla výrazně rychlejší s časem 0.00002 sekundy. Excel byl nejpomalejší s časem 0.094 sekundy.

Ačkoli vlastní implementace využívá vhodný algoritmus pro daný typ úlohy, její nižší výkon může být způsoben tím, že není tak optimalizovaná jako profesionálně vyvíjené knihovny. Knihovní řešení jako SciPy využívají vysoce optimalizované algoritmy a efektivní práci s pamětí i paralelizaci, což jim umožňuje dosáhnout mnohem rychlejších výsledků. Proto pozorujeme tak výrazný rozdíl ve výkonu. Naopak nástroje jako Excel jsou sice uživatelsky přívětivé, ale nejsou vhodné pro časově náročné výpočty ve větším měřítku.

9.2 Dopravní problém

1. Výsledek dosažený pomocí vlastní implementace algoritmu:

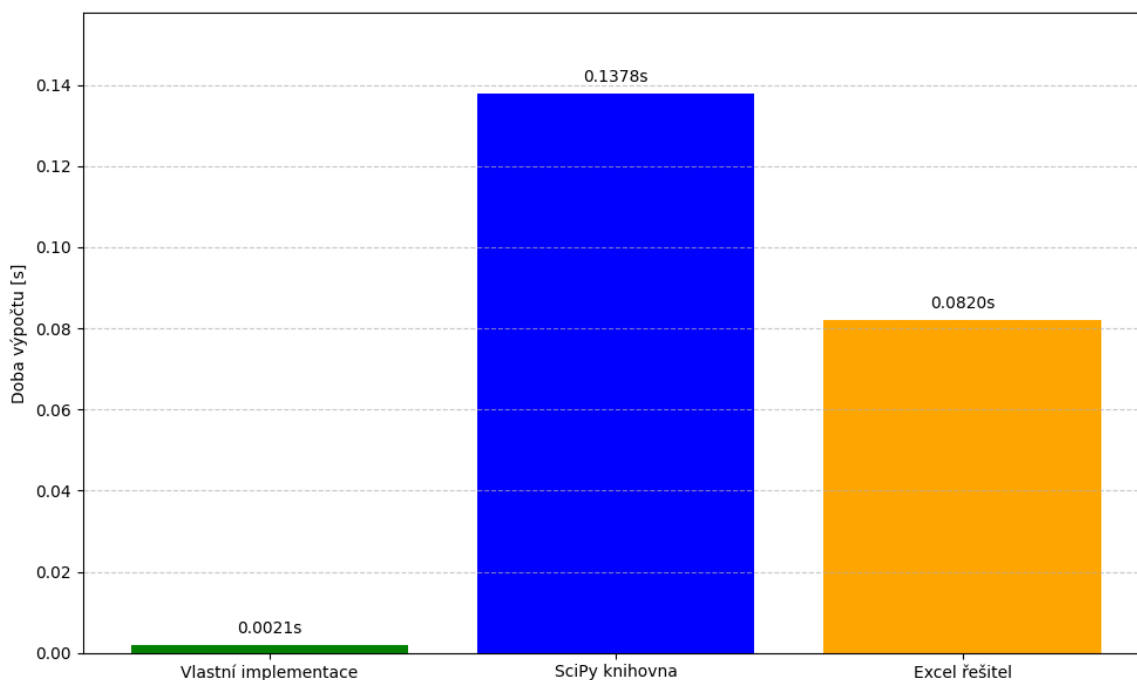
- Doba výpočtu: 0.0021s

2. Výsledek dosažený pomocí knihovny SciPy:

- Doba výpočtu: 0.1378 s

3. Výsledek dosažený pomocí Excelu:

- Doba výpočtu: 0.082 s



Obrázek 14 Porovnání doby výpočtu

Hodnocení výsledku měření:

Vlastní implementace tentokrát nejrychlejší, protože jsem použil specifický algoritmus přizpůsobený přímo pro transportní úlohu. Tento přístup nevyužívá obecné numerické metody pro zpracování omezení, a proto má velmi nízkou výpočetní režii. Výsledného řešení bylo dosaženo za 0.0021 sekundy.

Pro výpočet pomocí knihovny SciPy jsem využil metodu SLSQP. Tento přístup je velmi flexibilní a univerzální, což umožňuje řešit širokou škálu optimalizačních problémů. Nicméně jeho obecnost a způsob práce s omezeními vedly k delší době výpočtu — 0.1378 sekundy.

V Excelu jsem použil Řešitel s lineárním modelem a Simplexovou metodou pro minimalizaci nákladů. Tento způsob výpočtu byl rychlejší než u SciPy (0.082 sekundy), ale stále pomalejší než u vlastní implementace. Excel je vhodný zejména pro menší úlohy nebo rychlé testování, ale neposkytuje tolik možností optimalizace nebo úpravy algoritmu jako programové řešení.

ZÁVĚR

Tato bakalářská práce si kladla za cíl demonstrovat praktickou aplikaci metod optimalizace na konkrétních příkladech dvou klasických úloh – problému přiřazení a dopravní úlohy. Teoretická část byla zaměřená na definice daných problémů, jejich význam a široké uplatnění v různých oblastech lidské činnosti. Praktická část pak nabídla řešení vybraných úloh pomocí vlastních implementací a následné porovnání s nástroji jako jsou knihovna SciPy a Excel.

Výsledky ukázaly, že vlastní implementace může při vhodné volbě algoritmu dosahovat velmi konkurenceschopných výsledků – například v případě dopravní úlohy překonala jak knihovní řešení, tak Excel. Naopak u úlohy přiřazení byla knihovna SciPy výrazně rychlejší, což odráží vysokou míru její optimalizace a efektivní využití výpočetních zdrojů.

Z porovnání jednotlivých přístupů vyplývá, že nejlepších výsledků z hlediska výpočetního času lze dosáhnout kombinací specializovaných algoritmů určených pro konkrétní typ úlohy a nástrojů, které jsou pro tyto účely důkladně optimalizovány. Zatímco univerzální knihovny nabízejí širokou použitelnost a jednoduchost integrace, specializovaná řešení mohou být při správném návrhu výrazně efektivnější.

Z provedené práce je patrné, že schopnost aplikovat optimalizační metody v praxi je cennou dovedností nejen v technických a vědeckých oborech, ale i při každodenním rozhodování. Tato práce mi poskytla cennou příležitost prohloubit své teoretické znalosti a současně získat praktické zkušenosti s implementací a vyhodnocováním různých přístupů k řešení optimalizačních problémů.

SEZNAM POUŽITÉ LITERATURY

- [1] COLLEGE HIVE. *Introduction to Assignment Problem*. Online. 2025. Dostupné z: <https://notes.collegehive.in/books/business-mathematics/page/introduction-to-assignment-problem>. [cit. 2025-05-30].
- [2] AMPONSAH, Samuel Kofi; OTOO, Daniel; SALHI, Said a QUAYSON, Ebenezer. *Proposed Heuristic Method for Solving Assignment Problems*. Online. American Journal of Operations Research. 2016, vol. 6, no. 6, article 66040, s. 436–441. ISSN 2160-8849. Dostupné z: <https://doi.org/10.4236/ajor.2016.66040>. [cit. 2025-05-30].
- [3] JONKER, R a VOLGENANT, A.T. *Teaching linear assignment by Mack's algorithm*. Online. In: Twenty-Five Years of Operations Research in the Netherlands: Papers Dedicated to Gijs de Leve. Amsterdam: Centre for Mathematics and Computer Science, 1989, s. 54–60. Dostupné z: <https://web.eecs.umich.edu/~pettie/matching/Jonker-Volgenant-teaching-Macks-Bradford-method.pdf>. [cit. 2025-05-30].
- [4] ЦЬОНЬ, О. П.; БОБК, Ю. Я.; ДЗЮРА, В. О. а БОБК, І. П. *Дослідження операцій в транспортних системах: методичні вказівки до виконання лабораторних робіт для студентів ОКР «Бакалавр» з галузі знань 27 “Транспорт”, спеціальності 275 “Транспортні технології (автомобільний транспорт)”*. Online. Тернопіль: ФОП Паляниця В. А., 2017. Dostupné z: <https://elartu.tntu.edu.ua/handle/lib/21523>. [cit. 2025-05-30].
- [5] AMPONSAH, Samuel Kofi; OTOO, Daniel; SALHI, Said a QUAYSON, Ebenezer. *Proposed Heuristic Method for Solving Assignment Problems*. Online. American Journal of Operations Research. 2016, vol. 6, no. 6, article 66040, s. 436–441. ISSN 2160-8849. Dostupné z: <https://doi.org/10.4236/ajor.2016.66040>. [cit. 2025-05-30].
- [6] HILLIER, Frederick S. a LIEBERMAN, Gerald J. *Introduction to operations research*. 9th ed. Boston, Mass.: McGraw-Hill, 2010. ISBN 978-007-126767-0.
- [7] PERZINA, Radomír. *Dopravní problém*. Online. 2021. Dostupné z: https://is.slu.cz/el/opf/zima2021/INMBPOAE/3113831/Prednaska_7_ayrteesv.pdf. [cit. 2025-05-30].

- [8] *Mathematical Formulation of Transportation Problems*. Online. 2018. Dostupné z: <https://arts.brainkart.com/article/mathematical-formulation-of-transportation-problems-1126/>. [cit. 2025-05-31].
- [9] SEKNIČKOVÁ, Jana. *4EK213 – Lineární modely: 12. Dopravní problém – výchozí řešení*. Online. Praha, 2019. Dostupné z: <http://jana.kalcev.cz/vyuka/kestazeni/4EK213-LM/LM-pred12.pdf>. [cit. 2025-05-31].
- [10] БЕРЕЩАК, Ростислав. *Транспортна задача лінійного програмування*. Online. 2022. Dostupné z: <https://www.mathros.net.ua/transportna-zadacha-matematychna-postanovka-zadachi.html>. [cit. 2025-05-31].
- [11] *The North-West Corner Method*. Online. StudySmarter. 25.08.2023. Dostupné z: <https://www.studysmarter.co.uk/explanations/math/decision-maths/the-north-west-corner-method/>. [cit. 2025-05-31].
- [12] *Least Cost Method for Transportation Problems*. Online. College Hive. Dostupné z: https://collegehive.in/docs/1st_sem/site/BM/Unit_04_Transportation_Problem/4.d_Least_Cost_Method.html. [cit. 2025-05-31].
- [13] *Vogel's Approximation Method for Transportation Problems*. Online. College Hive. Dostupné z: https://collegehive.in/docs/1st_sem/site/BM/Unit_04_Transportation_Problem/4.e_Vogel_Approximation_Method.html. [cit. 2025-05-31].
- [14] CHAURAN, Deepa. *Transportation Problems: Modified Distribution Method (MODI)*. Online. Kanpur: Axis Institute of Technology, Applied Science & Humanities Department. Dostupné z: <https://www.slideshare.net/slideshow/optimal-solution-by-modi-method/248307482>. [cit. 2025-05-31].
- [15] Assignment problem applications: Real world scenarios and solutions. Online. Dostupné z: <https://fastercapital.com/content/Assignment-problem-applications--Real-world-scenarios-and-solutions.html#Optimizing-Delivery-Routes>. [cit. 2025-05-31].
- [16] KIRA, Akifumi; NAGANO, Kiyohito; SUGIYAMA, Manabu a KAMIYAMA, Naoyuki. *Optimal class assignment problem: a case study at Gunma University*. Online. In: . 2021, s. 1-15. Dostupné z: <https://arxiv.org/abs/2103.16879v1>. [cit. 2025-05-31].
- [17] ČANGALOVIĆ, Mirjana; KOVAČEVIĆ-VUJČIĆ, Vera; IVANOVIĆ, Lav a DRAŽIĆ, Milan. *Modeling and solving a real-life assignment problem at universities*.

Online. European Journal of Operational Research. 1998, vol. 110, no. 2, s. 223-233. ISSN 0377-2217. Dostupné z: [https://doi.org/10.1016/S0377-2217\(98\)00028-9](https://doi.org/10.1016/S0377-2217(98)00028-9). [cit. 2025-05-31].

[18] GHADLE, Kirtiwant P a MULEY, Yogesh M. *An Application of Assignment Problem in Laptop Selection Problem Using MATLAB*. Online. Applied Mathematics and Sciences: An International Journal (MathSJ). 2015, vol. 2, no. 1, s. 41-49. Dostupné z: <https://airccse.com/mathsj/papers/2115mathsj04.pdf>. [cit. 2025-05-31].

[19] SAHANI, Suresh Kumar; MULEY, Saroj Kumar; SAHANI, Kameshwar a ABDULLAHI, Ibrahim. *Real Life Applications of Transportation Problem Model: A Study and Complex Analysis of Hyundai Plants*. Online. Educational Administration: Theory and Practice. 2024, vol. 30, no. 5, s. 11546–11550. ISSN 2148-2403. Dostupné z: <https://kuey.net/index.php/kuey/article/view/4967>. [cit. 2025-05-31].

[20] AZIZI, Mohamed; BIRAFANE, Mahdi a BOUEDDINE, Hamza. *Solving Transportation Problem with LP: Biopharma Case Study*. Online. IOSR Journal of Business and Management. 2015, vol. 17, no. 3, article Ver. II, s. 15–24. ISSN 2278-487X. Dostupné z: www.iosrjournals.org, <http://www.iosrjournals.org/iosr-jbm/papers/Vol17-issue3/Version-2/C017321524.pdf>. [cit. 2025-05-30].

[21] SHAH, Karsh R. *Real Life Application of Transportation Problem in Malaysian Petrochemical Company*. Online. IJARIE (International Journal of Advanced Research in Innovative Ideas in Education). 2020, vol. 6, no. 4, s. 1189-1197. ISSN 2395-4396. Dostupné z: https://ijariie.com/AdminUploadPdf/Transportation_Problem_of_Malaysian_Petrochemical_Company_ijariie12442_docx_new.pdf?srsltid=AfmBOooi04DUaJfVef6CTukdV15Y6GqmaNjf0rTFjjsQBZBtbSxpQsBp. [cit. 2025-05-31].

[22] CEDER, Vernon L. *The quick Python book*. 2nd ed. Greenwich: Manning Publications, 2010. ISBN 978-1-935182-20-7.

[23] *Python Libraries: Top Lists, Uses, How to Choose*. Online. 27.02.2025. Dostupné z: <https://www.wscubetech.com/resources/python/libraries>. [cit. 2025-05-31].

[24] *Optimization and root finding (scipy.optimize)*. Online. Dostupné z: <https://docs.scipy.org/doc/scipy/reference/optimize.html#module-scipy.optimize>. [cit. 2025-05-31].

[25] *What Is Excel? Formulas, Functions, and More*. Online. 25.02.2025. Dostupné z: <https://www.coursera.org/articles/what-is-excel>. [cit. 2025-05-31].

[26] *Optimization with Excel Solver*. Online. Dostupné z: https://www.tutorialspoint.com/excel_data_analysis/advanced_data_analysis_optimization_with_excel_solver.htm. [cit. 2025-05-31].

[27] MICROSOFT. *Zavedení doplňku Řešitel v Excelu*. Online. Dostupné z: <https://support.microsoft.com/cs-cz/office/zavedení-doplňku-řešitel-v-excelu-612926fc-d53b-46b4-872c-e24772f078ca>. [cit. 2025-05-31].

SEZNAM OBRÁZKŮ

Obrázek 1 Schématické zobrazení Maďarské metody.....	16
Obrázek 2 Vajadření dopravní úlohy pomocí tabulky.....	17
Obrázek 3 Doplnky	57
Obrázek 4 Umištění řešitele.....	58
Obrázek 5 Matice nákladů	71
Obrázek 6 Rozhodovací matice	71
Obrázek 7 Nalezené řešení	74
Obrázek 8 Matice nákladů	74
Obrázek 9 Příprava rozhodovací matice	75
Obrázek 10 Nastavení Řešitele	76
Obrázek 11 Rozhodovací matice	77
Obrázek 12 Nalezený výsledek.....	80
Obrázek 13 Porovnání doby výpočtu.....	81
Obrázek 14 Porovnání doby výpočtu.....	82

SEZNAM TABULEK

Tabulka 1 Nákladová matice.....	25
Tabulka 2 Nalezení minimalního prvku v řádku	26
Tabulka 3 Matice po odečtení min řádku.....	26
Tabulka 4 Minimalní hodnoty ve sloupcích	26
Tabulka 5 Zredukováná matice.....	27
Tabulka 6 Pokrytí nul čarami.....	27
Tabulka 7 Nejmenší nepokrytý čarou prvek	28
Tabulka 8 Matice po odečtení a přičtení minimalního prvku	28
Tabulka 9 Pokrytí nul čarami.....	29
Tabulka 10 Minimální nepokrytý prvek	29
Tabulka 11 Matice po upřadě.....	30
Tabulka 12 Matice po pokrytí čarami	30
Tabulka 13 Přirazené nuly	31
Tabulka 14 Optimální přiřazení	31
Tabulka 15 Nákladová matice.....	32
Tabulka 16 Minimalní hodnoty v řádcích.....	32
Tabulka 17 Označený sloupec	33
Tabulka 18 Přidání minimálního rozdílu k prvkům.....	33
Tabulka 19 Označení hvězdičkou	34
Tabulka 20 6. krok - kontrola a označení sloupce	34
Tabulka 21 Přidání minimalního rozdílu k prvkům.....	35
Tabulka 22 Označený hvězdičkou prvek	35
Tabulka 23 7. krok	35
Tabulka 24 8. krok	36
Tabulka 25 Nákladová matice s optimálními hodnotami	36
Tabulka 26 Matice nákladů.....	37
Tabulka 27 1. krok výpočtu	38
Tabulka 28 2. krok výpočtu	39
Tabulka 29 3. krok výpočtu	39
Tabulka 30 4. krok výpočtu	40
Tabulka 31 5. krok výpočtu	40
Tabulka 32 6. krok výpočtu	41
Tabulka 33 7. krok výpočtu	41
Tabulka 34 Základní přípustné řešení	41

Tabulka 35 Tabulkové znazornění řádkových a sloupcových čísel	42
Tabulka 36 Tabulkové znazornění nepřímých sazeb.....	43
Tabulka 37 Znazornění uzavřené cesty	43
Tabulka 38 Nové základní řešení.....	44
Tabulka 39 Nové základní řešení.....	44
Tabulka 40 Znazornění řádkových a sloupcových čísel	45
Tabulka 41 Tabulkové znazornění nepřímých sazeb.....	45
Tabulka 42 Znazornění obvodu	46
Tabulka 43 Nové základní řešení.....	46
Tabulka 44 Tabulkové znazornění řádkových a sloupcových čísel	47
Tabulka 45 Nepřímé sazby	48
Tabulka 46 Dantzigův obvod.....	48
Tabulka 47 Nové základní řešení.....	49
Tabulka 48 Znazornění nepřímých sazeb	50
Tabulka 49 Optimalní řešení.....	50
Tabulka 50 Matice nákladů.....	54
Tabulka 51 Nákladová matice	55
Tabulka 52 Popis ostatních funkcí kodu	63
Tabulka 53 Popis ostatních funkcí kodu	68
Tabulka 54 Popis hlavních příkazů ve VBA v makro	73
Tabulka 55 Popis částí kodu VBA.....	79

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

OFV – Objective function variable (proměnná cílové funkce)

SEZNAM PŘÍLOH

Příloha P I: BP_prilohy

PŘÍLOHA P I: BP_PRILOHY

Tato příloha se skládá ze dvou hlavních souborů:

1. Transportation.py – obsahuje implementaci řešení dopravního problému v jazyce Python.
2. Hungarian.py – obsahuje implementaci Maďarského algoritmu pro řešení přiřazovací úlohy