Ekonomická fakulta
Faculty of Economics

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích
Ekonomická fakulta
Katedra aplikované matematiky a informatiky

Bakalářská práce

Využití nástrojů AI pro práci s grafy

Vypracovala: Barbora Rutová
Vedoucí práce: Mgr. Irena Štrausová Ph.D.

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

Ekonomická fakulta

Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Barbora RUTOVÁ**
Osobní číslo:	**E23666**
Studijní program:	**B0688A140010 Podniková informatika**
Téma práce:	**Využití nástrojů AI pro práci s grafy**
Zadávající katedra:	**Katedra aplikované matematiky a informatiky**

Zásady pro vypracování

Cílem práce bude prozkoumat a analyzovat možnosti využití nástrojů AI při práci s grafy a jejich aplikaci při výuce diskrétní matematiky. Metodický postup:

- Úvod do problematiky teorie grafů a umělé inteligence.
- Studium příslušné literatury zaměřené na AI metody a teorii grafů.
- Experimentální část – generování, analýza a optimalizace grafů pomocí AI, záznam výsledků a jejich vyhodnocení.
- Identifikace výhod a nevýhod jednotlivých přístupů.
- Závěry a možnosti budoucího využití AI nástrojů při výuce teorie grafů a řešení úloh diskrétní matematiky.

Rozsah pracovní zprávy:	**40-50 stran**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování bakalářské práce:	**tištěná/elektronická**

Seznam doporučené literatury:

Matoušek, J., & Nešetřil, J. (2022). *Kapitoly z diskrétní matematiky* (5th ed.). Praha: Karolinum.

Rosen, K. H. (2012). *Discrete Mathematics and Its Applications* (7th ed.). New York, NY: McGraw-Hill.

Hutter, M., & Hoos, H. H. (2019). *Automated Machine Learning: Methods, Systems, Challenges*. Springer Nature.

UMÍME INFORMATIKU. *Základy umělé inteligence*. Online. Dostupné z: https://www.umimeinformatiku.cz/book/cviceni-zaklady-umele-inteligence.

Vedoucí bakalářské práce: **Mgr. Irena Štrausová, Ph.D.**
Katedra aplikované matematiky a informatiky

Datum zadání bakalářské práce: **13. února 2025**
Termín odevzdání bakalářské práce: **10. dubna 2026**

JIHOČESKÁ UNIVERZITA
V ČESKÝCH BUDĚJOVICÍCH
EKONOMICKÁ FAKULTA
Studentská 13
370 05 České Budějovice

doc. RNDr. Zuzana Dvořáková Líšková, Ph.D.
děkanka

Mgr. Irena Štrausová, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 13. února 2025

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracovala pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

Datum Podpis

Ráda bych poděkovala vedoucímu mé bakalářské práce Mgr. Ireně Štrausové Ph.D. za vedení této práce, cenné rady a čas věnovaný konzultacím během zpracování.

Barbora Rutová

Obsah

1. Úvod, cíl(e) práce

Bakalářská práce je zaměřená na problematiku využití nástrojů umělé inteligence pro práci s grafy. Tento přístup kombinuje tradiční postupy diskrétní matematiky s metodami hlubokého učení, což umožňuje zkoumat grafové struktury z odlišného úhlu pohledu než pouze skrze exaktní výpočty.

V teoretické části je provedeno seznámení se základními pojmy diskrétní matematiky zaměřených na teorii grafů. Přiblížena bude problematiky základních pojmů týkajících se grafů, jejich vnitřní struktury a různé způsoby matematických zápisů. Dále se věnuje vymezení grafových problémů a algoritmů, což zahrnuje úlohy od hledání nejkratší cesty a konstrukce minimální kostry až po ověření existence hamiltonovských kružnic. Část teorie se také věnuje nástrojům umělé inteligence, přesněji grafovým neuronovým sítím (GNN). Tyto sítě představují specifickou architekturu schopnou pracovat přímo s topologií grafu.

Hlavní náplní práce je porovnání deterministických algoritmů se schopnostmi GNN řešit zadané úlohy teorie grafů. Základem je vytvoření spolehlivých referenčních datasetů s implementací klasických grafových algoritmů. Tyto datasety vznikají s využitím programovacího jazyka Python a knihoven NetworkX a PyTorch Geometric. Následně bude navázáno vytvořením a trénováním GNN na tyto typy grafů a úloh. Cílem je porovnání chování klasických grafových algoritmů s GNN modelem v rámci tří experimentů.

První experiment se zaměří na strukturální charakteristiky grafu, jako je průměr a poloměr, a způsob jeho reprezentace pomocí matic. Druhý experiment se bude zabývat vybranými grafovými problémy počítané deterministickými algoritmy, například konstrukce minimální kostry Kruskalovým algoritmem. V třetím experimentu bude navržen a natrénován model GNN pro čtyři úlohy, například klasifikaci grafů nebo hledání hamiltonovské kružnice. U těchto úloh se prověří, zda je grafová neuronová síť schopna samostatně identifikovat a uplatnit matematické principy pouze na základě naučených vzorců, aniž by jí byla tato pravidla explicitně naprogramována.

Po experimentální části budou vypsané výsledky experimentů, jejich analýza a identifikace výhod a nevýhod přístupů. Diskuse se zaměří na stabilitu učení, rychlost výpočtu, a především na limity lokálního kontextu u grafových neuronových sítí. V závěru práce

bude provedeno zhodnocení celé práce a jejich možného vývoje a využití v rámci výuky diskrétní matematiky.

2. Přehled řešené problematiky

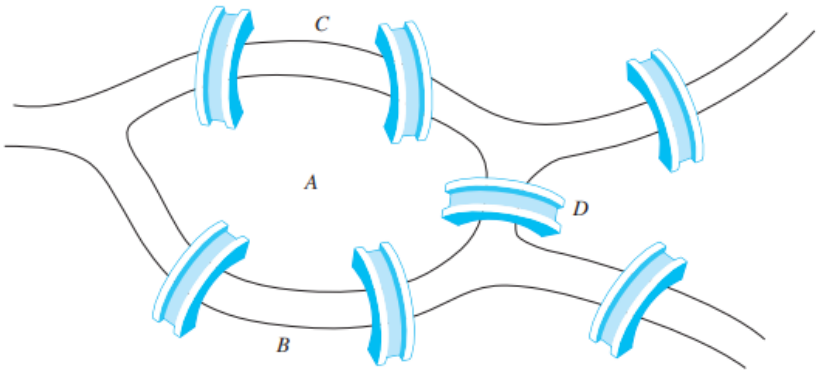
2.1. Teorie grafů

Teorie grafů je jednou z hlavních disciplín diskrétní matematiky, která se zabývá studiem relací mezi prvky v rámci abstraktních struktur. Základním objektem zkoumání je graf, formálně definován jako uspořádaná dvojice množin tvořená vrcholy a hranami, které tyto vrcholy propojují. Zatímco vrcholy reprezentují jednotlivé entity, hrany zachycují existenci a charakter vazeb mezi nimi, a to v podobě orientovaných či neorientovaných vztahů (Matoušek & Nešetřil, 2009). Tato oblast matematiky umožňuje modelovat a analyzovat topologické vlastnosti systémů, jako jejich souvislost, hustotu či vnitřní hierarchie. Studium uvedených struktur poskytuje nezbytný aparát pro řešení širokého spektra úloh zaměřených na optimalizaci cest, toků nebo konektivity v definovaných sítích.

2.1.1. Historie teorie grafů

Počátky teorie grafů jsou spojeny s Leonhardem Eulerem, který v roce 1736 publikoval řešení problému sedmi mostů města Královce. Euler tehdy poukázal, že hledaná cesta městem neexistuje, čímž položil základy topologického nahlížení na sítě bez nutnosti uvažovat jejich přesné geometrické rozměry (Matoušek & Nešetřil, 2009). Tento moment je v literatuře považován za vznik disciplíny, která je místo fyzické podoby objektů zkoumá jejich vzájemná propojení.

Obrázek 1: Sedm mostů města Královce



Zdroj 1:(Rosen, 2013)

V 19. století se oblast teorie grafů dále rozvíjela skrze praktické aplikace v technických vědách. Gustav Kirchoff v letech 1845 a 1847 využil grafové struktury při analýze elektrických obvodů pro výpočet proudů v jednotlivých větvích, což vedlo k formálnímu vymezení nezávislých obvodů (Šišma, 1998). Souběžně s těmito fyzikálními aplikacemi rozvíjel Arthur Cayley v roce 1857 využití grafů, konkrétně stromů, v chemii k identifikaci molekul alkanů. Ve stejném období sir William Rowan Hamilton představil tzv. ikosaedrickou hru, čímž do diskrétní matematiky vnesl koncept hamiltonovské kružnice.

Zásadním teoretickým impulsem se stal problém čtyř barev, který v roce 1852 formuloval Francis Guthrie. Snaha o důkaz, že jakoukoliv rovinnou mapu lze obarvit pouze čtyřmi barvami tak, aby sousední území neměla shodnou barvu, podnítila rozsáhlý výzkum v oblasti rovinných grafů (Šišma, 1998). Ačkoliv se o správný důkaz pokoušela řada matematiků, například Alfred Kempe či Percy Heawood, definitivní potvrzení přinesli až Kenneth Appel a Wolfgang Haken v roce 1976, přičemž jejich řešení vyžadovalo značný rozsah počítačových výpočtů.

Významný přínos pro teorii grafů představuje česká matematická škola 20. století. Otakar Borůvka v roce 1926 publikoval algoritmus pro nalezení minimální kostry grafu, přičemž jeho motivace byla ryze praktická – hledání ekonomicky nejvýhodnějšího způsobu výstavby elektrických sítí na Moravě (Šišma, 1998). Na tyto výsledky navázal v roce 1930 Vojtěch Jarník, čímž oba matematici položili základy pro moderní optimalizační algoritmy, které tvoří nezbytný základ současné informatiky.

2.1.2. Využití v informatice a modelování sítí

Implementace grafových struktur v informatice představuje klíčový mechanismus pro formalizaci datových objektů a jejich vzájemných interakcí. Pro výpočetní zpracování je určující volba datové reprezentace, nejčastěji v podobě matice sousednosti nebo seznamu sousedů, což přímo ovlivňuje časovou a prostorovou složitost operací (Cormen et al., 2022). V síťové infrastruktuře grafy umožňují exaktní popis topologie systému, kde uzly představují aktivní prvky a hrany fyzické propojení. Tato problematika, zaměřená na optimalizaci toků a hledání nejkratších cest, přímo navazuje na historické základy analýzy technických sítí, které v českém prostředí rozpracovali například Borůvka a Jarník (Šišma, 1998).

Moderní analýza digitálního prostředí využívá grafové modely také pro studium webu a sociálních sítí. Orientované hrany v podobě hypertextových odkazů umožňují algoritmům typu PageRank hodnotit relevanci informací, zatímco v sociálních sítích se sleduje existence vysoce propojených center, tzv. hubů (Barabási & Pósfai, 2016). Prohledávání těchto struktur do hloubky (Depth-first search) či do šířky (Breadth-first search) představuje základní aparát pro identifikaci komunit a sledování šíření dat v rámci sítě (Rosen, 2013). Nezastupitelnou roli hrají grafy rovněž v softwarovém inženýrství při modelování stavových prostorů. Konečné automaty, definované jako orientované grafy, umožňují verifikovat chování softwaru a efektivně spravovat systémové závislosti nebo plánování složitých úloh (Rosen, 2013).

2.2. Základní definice a pojmy

Pro porozumění možnostem, které teorie grafů nabízí, je nezbytné zavést jednotnou a přesnou terminologii. Grafy, ačkoliv je jejich vizuální zobrazení v pedagogické praxi nejčastější, představují v matematickém smyslu abstraktní struktury definované prostřednictvím teorie množin (Matoušek & Nešetřil, 2009). Tento způsob definice slouží jako univerzální nástroj pro modelování široké škály vztahů a propojení mezi objekty, čímž se teorie grafů stává jedním z pilířů moderní informatiky a diskrétní matematiky jako celku.

Právě tato abstrakce umožňuje popisovat komplexní systémy, od sociálních a komunikačních sítí až po dopravní infrastrukturu či molekulární struktury (Bondy & Murty, 2008). Jednotné zavedení pojmů dovoluje systematicky analyzovat základní vlastnosti grafů, mezi které patří zejména souvislost, stupně vrcholů nebo existence specifických cyklů a cest. Tyto definice jsou klíčové pro studium vnitřní struktury grafů a pro pochopení jejich chování v různých kontextech, což vytváří teoretický základ pro následnou aplikaci nástrojů umělé inteligence.

Matematický popis grafů je rovněž nepostradatelný pro analýzu a návrh algoritmů. Většina moderních výpočetních postupů, například hledání nejkratší cesty, nepracuje s vizuálním zobrazením, ale využívá formální reprezentace, jako je množinový zápis nebo matice sousednosti (Cormen et al., 2022). Díky tomu lze řešit úlohy vyžadující analýzu vztahů mezi prvky systému s vysokou přesností a efektivitou. Teorie grafů se tak projevuje jako univerzálně použitelná disciplína, jejíž všestrannost je v současné vědě a technice naprosto zásadní.

2.2.1. Definice grafu, vrcholu a hrany

Základním pojmem teorie grafů je graf, který matematicky definujeme jako strukturu složenou ze dvou množin. Tato definice tvoří formální rámec pro veškeré operace v diskrétní matematice a umožňuje exaktní modelování vztahů v reálných i abstraktních systémech.

Definice: Graf G je uspořádaná dvojice (V, E), kde V je konečná neprázdná množina vrcholů a E je množina hran.

V rámci této struktury představuje V neprázdnou množinu vrcholů a E množinu hran. Pro nejzákladnější typy grafů, označované jako jednoduché, platí, že hrany jsou definována jako dvouprvkové podmnožiny množiny vrcholů V. Formálně to lze vyjádřit jako $E \subseteq P2(V)$, což znamená, že každá hrana spojuje dvě různé vrcholy a mezi libovolnou dvojicí vrcholů neexistuje více než jedna hran. Vizuálně jsou vrcholy v grafickém znázornění interpretovány jako body v rovině, nejčastěji v podobě teček. Hrany jsou pak zobrazeny jako spojnice mezi těmito body, přičemž mohou mít podobu úseček nebo zakřivených čar.

Z hlediska kombinatoriky je důležitým údajem celkový počet všech neorientovaných jednoduchých grafů, které lze na n-prvkové množině vrcholů vytvořit. Tento počet je roven $2^{\binom{n}{2}} = 2^{\frac{n(n-1)}{2}}$ (Matoušek & Nešetřil, 2009). Vzorec vyjadřuje skutečnost, že pro každou z možných hran mezi dvěma vrcholy existují pouze dvě možnosti, hrana je v grafu buď obsažena, nebo není.

Tento výpočet se týká takzvaných jednoduchých grafů, pro které platí určitá omezení jejich vnitřní struktury. V jednoduchém grafu nemohou mezi stejnou dvojicí vrcholů existovat vícenásobné hrany a zároveň se v něm nevyskytují smyčky, tedy hrany spojující vrchol se sebou samým. Porozumění těmto základním principům je nezbytné pro studium uspořádání grafů a pro jejich další matematickou analýzu.

V oblasti umělé inteligence transformují grafové neuronové sítě (GNN) množinové definice vrcholů a hran na vektorové reprezentace, čímž umožňují efektivní analýzu topologie sítě a odhalování skrytých souvislostí v složitých strukturách (Hamilton, 2020).

2.2.2. Okolí a stupeň vrcholu, skóre

Kromě celkové struktury grafu je pro jeho detailnější analýzu nezbytné definovat vlastnosti specifické pro jednotlivé vrcholy. Mezi tyto základní lokální atributy patří především okolí vrcholu, jeho stupeň a výsledné skóre grafu (Matoušek & Nešetřil, 2009).

Okolí vrcholu

Okolí vrcholu představuje základní lokální charakteristiku, která určuje přímé vazby mezi jednotlivými prvky struktury a definuje bezprostřední dosah každého vrcholu.

Definice: Okolí vrcholu v v jednoduchém grafu $G = (V, E)$ je množina $N(v) = \{ u \in V : \{u, v\} \in E\}$.

Tato množina obsahuje všechny vrcholy, které jsou s vrcholem v přímo spojeny hranou. Identifikace sousedních prvků umožňuje analyticky popsat přímé vazby mezi vrcholy, což tvoří základ pro následné zkoumání celkové souvislosti grafů.

Stupeň vrcholu

S pojmem okolí vrcholu úzce souvisí stupeň vrcholu, který kvantifikuje míru propojení v grafech.

Definice: Stupeň vrcholu v v grafu G, značený jako $\deg(v)$*, je počet hran incidentních s tímto vrcholem.*

V případě jednoduchých grafů odpovídá stupeň velikosti okolí $|N(v)|$, tedy počet jeho sousedů. Základní vlastnost těchto struktur popisuje věta o součtu stupňů, podle níž je součet stupňů všech vrcholů roven dvojnásobku počtu všech hran (Matoušek & Nešetřil, 2009).

Skóre grafu

Skóre grafu představuje globální charakteristiku, která je odvozena z množiny stupňů všech jejích vrcholů.

Definice: Skóre grafu G je posloupnost stupňů $(\deg(v_1), \deg(v_2), \ldots, \deg(v_n))$ *všech vrcholů grafu.*

Pro účely srovnávání a analýzy se tato posloupnost standardně řadí sestupně. Přestože konkrétní pořadí vrcholů v rámci definice není určující, jednotné řazení umožňuje

efektivní porovnávání různých grafových struktur (Matoušek & Nešetřil, 2009). Skóre grafu slouží jako důležitá strukturní charakteristika, která slouží k rozhodnutí o izomorfismu dvou grafů.

2.2.3. Klasifikace vrcholů

Rozlišování různých typů vrcholů podle jejich specifických vlastností a postavení v systému představuje základní krok k pochopení vnitřní struktury grafů a k analýze jejich celkového chování ((Matoušek & Nešetřil, 2009).

Izolovaný vrchol

Izolovaný vrchol představuje prvek grafu, který nevykazuje žádnou vazbu na ostatní vrcholy v systému a netvoří incidenci s žádnou hranou.

Definice: Vrchol v je izolovaný, pokud platí $\deg(v) = 0$.

V grafické reprezentaci stojí tento vrchol samostatně bez jakékoliv hrany. V matici sousednosti se jeho přítomnost projevuje výhradně nulovými hodnotami v příslušném řádku a sloupci, což v grafech s více vrcholy způsobuje nesouvislost celé struktury, neboť izolovaný vrchol tvoří samostatnou komponentu (Matoušek & Nešetřil, 2009).

Koncový vrchol (list)

Koncový vrchol neboli list je definován jako vrchol, který je spojen hranou právě jen s jedním sousedním vrcholem.

Definice: Vrchol v je koncový vrchol (list), pokud platí $\deg(v) = 1$.

Z této definice vyplývá, že list je spojen právě s jedním sousedem, což je základem v teorii stromů, kde tyto vrcholy reprezentují koncové (terminální) uzly jednotlivých větví (Matoušek & Nešetřil, 2009).

2.2.4. Typy grafů

V diskrétní matematice představuje klasifikace grafů základní nástroj pro pochopení jejich struktur a vlastností. Toto rozlišení umožňuje přesně popsat chování grafových modelů a slouží jako vodítko pro výběr nejvhodnějších algoritmů (Matoušek & Nešetřil, 2009). Správná identifikace typu grafu je proto kritickým krokem k řešení jakékoliv úlohy, neboť v informatice přímo určuje volbu datové struktury a ovlivňuje efektivitu celého výpočtu.

Grafy se klasifikují podle několika klíčových kritérií. Prvním je orientace hran, která rozlišuje, zda je vztah mezi vrcholy symetrický (neorientovaný graf), nebo zda má určitý směr (orientovaný graf). Dále je důležitá kvantita vztahů mezi vrcholy, pokud mezi dvěma vrcholy existuje maximálně jedna hrana, hovoříme o grafu jednoduchém, zatímco u multigrafu je povolena existence více hran a u pseudografu i smyčky (Demel, 2002). Posledním podstatným kritériem je ohodnocení hran, které rozděluje grafy na nevážené a vážené grafy, u nichž hrany nesou dodatečné numerické informace, jako jsou váha, cena, kapacita nebo vzdálenost (Cormen et al., 2022).

Pro moderní přístupy v oblasti umělé inteligence, zejména pro grafové neuronové sítě (GNN), má tato klasifikace zásadní význam, neboť konkrétní typ grafu určuje způsob extrakce vlastností i interpretaci dat v praktické části práce. Znalost těchto struktur je zároveň nezbytná pro korektní generování syntetických datových sad, které jsou klíčové pro trénování a testování modelu GNN u specifických úloh.

Orientovaný graf

Orientovaný graf je struktura, v níž mají vztahy mezi vrcholy pevně definovaný směr.

Definice: Orientovaný graf G je uspořádaná dvojice $G = (V, E)$, kde V je množina vrcholů a $E \subset V \times V$ je množina orientovaných hran.

Tato definice připouští i existenci smyček. Zásadním rozdílem oproti neorientovaným grafům je asymetrie vztahů, existence hrany (u, v) není ekvivalentní s existencí hrany (v, u), neboť reprezentují odlišné orientované vazby (Demel, 2002). V grafickém zobrazení se směr hrany vyjadřuje šipkou.

Neorientovaný graf

Neorientovaný graf představuje základní typ grafu, v němž vztahy mezi vrcholy nemají určený směr a jsou vždy symetrické.

Definice: Neorientovaný graf G je uspořádaná dvojice $G = (V, E)$, kde V je neprázdná množina vrcholů a E množina hran, přičemž každá hrana je dvouprvková podmnožina množiny V.

Hrana je tedy definována jako neuspořádaná dvojice (u, v), což znamená, že vztah je identický v obou směrech, tj. $(u, v) = (v, u)$.

Vážený (ohodnocený) graf

V situacích, kdy pouhá existence hrany neposkytuje dostatečnou informaci, se využívají vážené (ohodnocené) grafy, které umožňují vztahy kvantifikovat.

Definice: Vážený (ohodnocený) graf je trojice $G = (V, E, w)$, kde V je množina vrcholů, E je množina hran a $w : E \to \mathbb{R}$ je ohodnocovací funkce (váhová funkce), která každé hraně $e \in E$ přiřazuje reálné číslo $w(e)$, nazývané váha hrany.

Váha hrany v praxi reprezentuje různorodé veličiny, jako jsou fyzická vzdálenost, čas, ekonomické náklady či kapacita sítě. Vážené grafy jsou základem pro klasické optimalizační úlohy, například Dijkstrův algoritmus pro hledání nejkratší cesty nebo konstrukce minimální kostry.

Diskrétní graf

Diskrétní graf, označovaný jako D_n, představuje nejjednodušší grafovou strukturu tvořenou n vrcholy bez existence hran.

Definice: Diskrétní graf je uspořádaná dvojice $G = (V, E)$, kde V je neprázdná množina vrcholů a E je prázdná množina hran ($E = \emptyset$).

Takový graf, výjimkou případ s jedním vrcholem, je vždy nesouvislý a skládá se z n komponent souvislosti, přičemž z hlediska teorie grafů tvoří přímý komplement úplného grafu K_n.

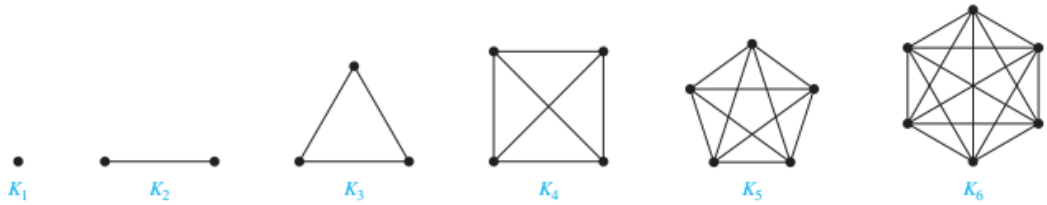
Úplný graf

Úplný graf, označovaný K_n, představuje specifický typ neorientovaného grafu, ve kterém existuje hrana mezi každou dvojicí vrcholů.

Definice: Úplný graf K_n je uspořádaná dvojice $G = (V, E)$, kde V je množina n vrcholů a E je množina všech dvouprvkových podmnožin V, tedy $E = \binom{V}{2}$.

Každý vrchol má v tomto grafu stupeň $n - 1$ a celkový počet hran odpovídá kombinačnímu číslu $\binom{V}{2}$. Úplný graf vykazuje maximální míru symetrie.

Obrázek 2: Úplné grafy



Zdroj 2: ***(Rosen, 2013)***

Bipartitní graf

Bipartitní graf je charakterizován rozdělení množiny vrcholů do dvou disjunktních podmnožin, tak, že každá hrana spojuje výhradně vrchol z jedné množiny s vrcholem z množiny druhé.

Definice: Graf $G = (V, E)$ *je bipartitní, jestliže existují dvě disjunktní podmnožiny* $A, B \subseteq V$ *takové, že* $A \cup B = V$ *a* $A \cap B = \emptyset$*, a pro každou hranu* $\{a, b\} \in E$ *platilo* $a \in A$*,* $b \in B$ *nebo naopak.*

Klíčovou strukturální vlastností těchto grafů je absence cyklů liché délky což představuje základní matematické kritérium pro jejich identifikaci i algoritmické zpracování (Matoušek & Nešetřil, 2009).

Úplný bipartitní graf

Úplný bipartitní graf, standardně označovaný $K_{m,n}$, představuje mezní případ bipartitního grafu, v níž je každý vrchol z množiny A o velikosti m spojen hranou se všemi n vrcholy z množiny B.

Definice: Úplný bipartitní graf $K_{m,n}$*, o* m *a* n *vrcholech je graf* $G = (V, E)$*, existují dvě disjunktní podmnožiny* $A, B \subseteq V$ *takové, že* $A \cup B = V$ *a* $A \cap B = \emptyset$*,* $|A| = m$ *a* $|B| = n$*, a pro každou dvojici* $a \in A$*,* $b \in B$ *existuje právě jedna hrana* $\{a, b\} \in E$*.*

Celkový počet hran je dán součinem $m \times n$, přičemž všechny stupně v množině A jsou rovny n a v množině B pak odpovídají m (Matoušek & Nešetřil, 2009).

Obrázek 3: Úplné bipartitní grafy

Zdroj 3: ***(Rosen, 2013)***

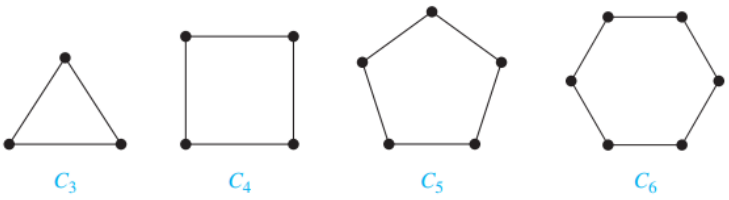
Kružnice

Kružnice C_n, označována též jako cyklus délky n, představuje graf tvořený jediným uzavřeným tahem, který prochází všemi svými vrcholy.

Definice: Kružnice C_n *je graf* $G = (V, E)$, *kde* $V = \{v_0, v_1, \ldots, v_n\}$ *a* $E = \{\{v_i, v_{(i+1)}\}: 1 \leq i \leq n-1\} \cup \{\{v_n, v_1\}\}$, *přičemž* $n \geq 3$.

Mezi základní charakteristiky patří rovnost počtu vrcholů a hran ($|V| = |E| = n$) a stupeň každého vrcholu v grafu který je roven dvěma.

Obrázek 4: Kružnice



Zdroj 4: ***(Rosen, 2013)***

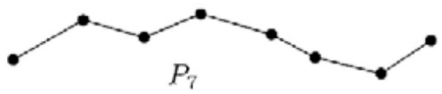
Cesta

Cesta P_n představuje základní souvislý graf modelující lineární sekvenci vrcholů.

Definice: Graf P_n *(cesta na* n *vrcholech) je graf* $G = (V, E)$ *s množinou* $V = \{v_0, v_1, \ldots, v_n\}$ *a množinou hran* $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{(n-1)}), v_n\}\}$.

Tato struktura o n vrcholech obsahuje vždy právě $n-1$ hran, přičemž vnitřní vrcholy mají stupeň 2 a dva koncové vrcholy stupeň 1.

Obrázek 5: Cesta

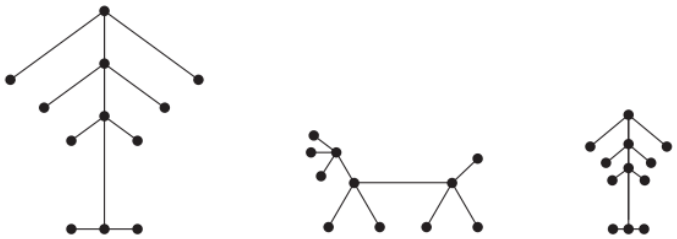


Zdroj 5: ***(Matoušek & Nešetřil, 2009)***

Stromy

Strom je definován jako souvislý graf, který neobsahuje žádný cyklus. Každý strom o n vrcholech disponuje právě $n - 1$ hranami. Graf, jehož každá souvislá komponenta odpovídá definici stromu, se označuje jako les. Významným prvkem těchto struktur jsou listy, tedy koncové vrcholy se stupněm 1, přičemž platí, že každý strom s alespoň dvěma vrcholy disponuje minimálně dvěma listy. Pokud ve stromu zvolíme zvláštní vrchol jako kořen, hovoříme o kořenovém stromu, v němž lze přirozeně definovat vztahy rodič–potomek.

Obrázek 6: Tři grafy stromů dohromady tvořící les



Zdroj 6: ***(Rosen, 2013)***

2.3. Struktura grafu

Struktura grafu představuje, jakým způsobem jsou jednotlivé vrcholy propojeny. Tato struktura neurčuje pouze statické relace mezi vrcholy, ale přímo ovlivňuje dynamické procesy, jako je efektivita vyhledávání cest nebo rychlost šíření informací v sítích. V širším pojetí moderní informatiky složí studium grafových struktur jako základ pro modelování a optimalizaci rozsáhlých systémů od topologie internetu a sociálních interakcí až po architektury hlubokých neuronových sítí (Barabási & Pósfai, 2016).

V teorii grafů je rozlišení mezi vlastnostmi lokálními a globálními zásadní. Zatímco lokální charakteristiky se zaměřují na okolí jednotlivých vrcholů, typicky jde o stupeň vrcholu, globální vlastnosti popisují graf jako nedělitelný celek. Mezi tyto atributy patří například souvislost grafu, jeho průměr či přítomnost specifických cyklů a podgrafů. Důležitou roli hrají vlastnosti, které se nemění při jakémkoli překreslení nebo přejmenování vrcholů. Tyto charakteristiky neboli grafové invarianty, například počet vrcholů, hran, stupňová posloupnost nebo existence určitého podgrafu, jsou důležité pro rozpoznávání stejných struktur.

Kvantitativní popis grafu se opírá o základní metriky, jimiž jsou řád grafu (n), odpovídající počtu vrcholů $|V|$, a velikosti grafu (m), vyjádřená počtem hran $|E|$. Vzájemný vztah mezi těmito veličinami definuje hustotu grafu, kde vysoká hustota znamená blízkost úplnému grafu, zatímco nízká hustota zase řídký graf, což v informatice ovlivňuje volbu algoritmů a datových struktur.

Pro výuku diskrétní matematiky je podstatné pochopení faktu, že matematická podstata grafu je zcela nezávislá na jeho geometrickém znázornění. Jeden a tentýž graf může být vykreslen v mnoha vizuálně odlišných podobách, aniž by se změnily jeho matematické vlastnosti. Schopnost nesoustředit se na vizuální podobu grafu a zaměřit se na jeho čistou strukturální propojenost je dovedností, kterou by si studenti měli osvojit. Vizuální zobrazení slouží pouze jako kognitivní pomůcka a nemělo by být chápáno jako definující znak samotného modelu.

Tento abstraktní pohled na grafy přebírají i moderní nástroje umělé inteligence, zejména grafové neuronové sítě. Na rozdíl od lidského pozorovatele, který se často opírá o prostorovou intuici, systému umělé inteligence analyzují grafy prostřednictvím jejich matematických reprezentací. Hamilton (2020) vysvětluje, že tyto modely využívají procesy agregace informací z okolních uzlů, čímž extrahují strukturální příznaky přímo z maticové podoby dat, nejčastěji v podobě matic sousednosti a matic příznaků vrcholů. Pro umělou inteligenci tedy vizuální podoba grafu nehraje roli, relevantní je pouze algebraická struktura, která umožňuje odhalovat skryté vzorce v datech.

2.3.1. Sled, tah, cesta

Analýza grafových struktur se nezaměřuje pouze na to, které dvojice vrcholů jsou spojeny hranou, ale také způsobu, jak lze mezi nimi procházet, což je formálně popsané

jako sledy, tahy a cesty. Tyto posloupnosti vrcholů a hran tvoří základní nástroj pro určování vzdáleností a dosažitelností v grafu, přičemž i jemné rozdíly v pravidlech pro opakování vrcholů či hran zásadně mění vlastnosti výsledného modelu (Matoušek & Nešetřil, 2009).

Sled

Sled představuje v teorii grafů nejobecnější formální vyjádření pohybu mezi vrcholy, přičemž definuje základní způsob, jakým lze v grafové struktuře sledovat relace mezi vrcholy.

Definice: Sled v grafu je posloupnost $v_0, e_1, v_1, e_2, \ldots, e_k, v_k$*, kde každá hrana* e_i *spojuje vrcholy* v_{i-1} *a* v_i*.*

Sled je tedy konečná posloupnost střídajících se vrcholů a hran grafu. Délka takto definovaného sledu je rovna číslu k, které odpovídá celkovému počtu hran obsažených v dané posloupnosti.

Charakteristickým rysem sledu, který jej odlišuje od specifičtějších struktur, jako jsou tahy či cesty, je úplná absence omezení na počet opakování jednotlivých prvků. V rámci sledu se mohou libovolně opakovat jak vrcholy, tak i hrany. Z hlediska diskrétní matematiky je tento koncept stěžejní především pro studium dosažitelnosti grafu. Existuje-li mezi dvěma vrcholy sled, jsou tyto vrcholy vzájemně dosažitelné, což tvoří jeden ze základních východisek pro analýzu souvislosti grafu.

Tah

Tah představuje specifický typ sledu, který do pohybu v grafu vnáší omezení týkající se opakování hran. Zatímco v obecném sledu byla přípustná libovolná opakování všech prvků, tah je definován jako sled, v němž jsou všechny hrany navzájem různé (Matoušek & Nešetřil, 2009).

Definice: Tah je sled $v_0, e_1, v_1, e_2, \ldots, e_k, v_k$*, kde každá hrana* e_i *spojuje vrcholy* v_{i-1} *a* v_i *a žádná hrana se v posloupnosti nevyskytuje více než jednou.*

Toto omezení znamená, že žádná hrana nesmí být v rámci jednoho tahu použita více než jednou, avšak vrcholy se v tahu opakovat mohou. Tato vlastnost umožňuje, aby tah „protnul“ sám sebe v jednom nebo více vrcholech.

Z hlediska pozice počátečního a koncového vrcholu se rozlišují dvě základní formy. Otevřený tah začíná v jiném vrcholu, než ve kterém končí $v_0 \neq v_k$, zatímco v případě uzavřeného tahu platí rovnost $v_0 = v_k$.

Zcela zásadním je poté eulerovský tah, který je definován jako tah, jenž projde každou hranou daného grafu právě jednou.

Cesta

Cesta představuje v uspořádání pohybů v grafu nejpřísněji vymezenou strukturu, která do posloupnosti vrcholů a hran vnáší absolutní omezení ohledně opakování všech jejích prvků.

Definice: Cesta je sled $v_0, e_1, v_1, e_2, \ldots, e_k, v_k,$, kde každá hrana e_i spojuje vrcholy v_{i-1} a v_i a všechny vrcholy $v_0, \ldots, v_k$, tudíž v posloupnosti se nevyskytuje žádná hrana ani vrchol více než jednou.

Z této definice vyplývá, že v rámci cesty musí být unikátní všechny použité hrany, nejen vrcholy. Tato vlastnost je určující pro ideální model cesty vyjádření přímého, necyklického a efektivního spojení mezi dvěma vrcholy, které neobsahuje žádné nadbytečné smyčky či opakování se navštívených vrcholů. Délka cesty odpovídá počtu jejich hran a představuje základní parametr pro definici metriky vzdálenosti v grafu, a to zejména jako délka nejkratší cesty mezi dvěma vrcholy (Cormen et al., 2022).

Shrnutí rozdílů

Rozdíly mezi základními typy pohybů v grafech lze shrnout následovně, sled umožňuje libovolné opakování jak vrcholů, tak i hran. Tah zakazuje pouze opakované použití hran, zatímco vrcholy se v něm mohou opakovat (jako při kreslení obrázku jedním tahem). Cesta je nepřísnější formou, v níž se nesmí opakovat vrcholy ani hrany. Tato hierarchie stupňujících se omezení postupně zvyšuje informační hodnotu každé struktury pro analýzu grafu.

2.3.2. Souvislost grafu, komponenty, doplněk grafu, hranový graf

Studium vnitřní struktury grafů se neomezuje pouze na analýzu cest nebo tahů, ale zahrnuje také zkoumání celkové integrity a vzájemné provázanosti. Souvislost grafu a identifikace jeho komponent určují, zda lze informaci přenášet mezi všemi vrcholy,

nebo zda je model rozdělen do izolovaných částí (Matoušek & Nešetřil, 2009). Pomocí doplňkových a hranových grafů lze analyzovat graf z odlišných pohledů. Doplněk pracuje s neexistujícími hranami, zatímco hranový graf převádí hrany původního grafu na vrcholy nového grafu. V GNN tyto vlastnosti určují, jak efektivně mohou modely šířit informace napříč celou strukturou a jak přesně dokážou předpovídat vlastnosti jednotlivých vazeb (Hamilton, 2020).

Souvislý graf

Souvislost grafu patří mezi základní strukturální charakteristiky grafu určující jeho propojenost a integritu.

Definice: Neorientovaný graf $G = (V, E)$ *je souvislý, pokud pro každou libovolnou dvojici vrcholů* u, v *existuje alespoň jedna cesta spojující tyto vrcholy.*

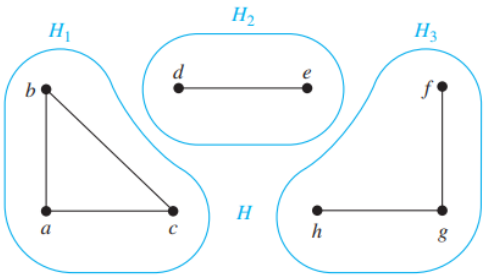
Tato vlastnost zaručuje, že graf tvoří jeden kompaktní celek, v němž neexistují žádné izolované, či vzájemně nedostupné vrcholy. Z hlediska kvantitativních vlastností platí, že každý souvislý graf řádu n musí mít alespoň $n - 1$ hran, což odpovídá minimálnímu počtu spojení nutnému k udržení souvislosti, grafy, které této meze dosahují jsou právě stromy.

Komponenta souvislosti

Pokud graf není souvislý, dochází k rozpadu na několik souvislých podgrafů, které jsou označovány jako komponenty. Počet komponent v nesouvislém grafu se standardně označuje $c(G)$ nebo $k(G)$, přičemž platí, že graf je souvislý právě tehdy, když $c(G) = 1$.

Definice: Komponenta je takový souvislý podgraf H grafu G, ke kterému nelze z G přidat žádný další vrchol ani hranu, aniž by H přestal být souvislý.

Obrázek 7: Nesouvislý graf H s komponentami H_1, H_2 a H_3



Zdroj 7: ***(Rosen, 2013)***

Doplněk grafu

Doplněk grafu neboli komplement, označovaný jako $\bar{G}$, představuje graf, který sdílí identickou množinu vrcholů V, avšak jeho množina hran sestává právě z těch dvojic vrcholů, které nejsou propojeny v původním grafu G.

Definice: Doplněk (komplement) neorientovaného grafu $G = (V, E)$ je graf $G' = (V, E')$ se stejnou množinou vrcholů, kde E' obsahuje právě ty neuspořádané dvojice vrcholů, které nejsou hranami v G.

Sjednocením grafu a jeho doplňku tak vzniká úplná graf K_n, v němž jsou spojeny všechny možné dvojice vrcholů.

Hranový graf

Hranový graf, označovaný jako $L(G)$, neorientovaného grafu G reprezentuje sousednost mezi hranami původního grafu.

Definice: Vrcholy hranového grafu $L(G)$ odpovídají hranám grafu G a dva vrcholy v $L(G)$ jsou spojeny hranou právě tehdy, když odpovídající hrany v G sdílejí společný koncový vrchol.

2.3.3. Vzdálenost v grafu, excentricita, průměr, poloměr, matice vzdálenosti a dosažitelnosti

Základní strukturální charakteristikou je vzdálenost v grafu, která umožňuje měřit, jak „daleko“ od sebe leží jednotlivé vrcholy. Z něj poté vycházejí klasické charakteristiky jako excentricita, průměr a poloměr. V praktických příkladech se tyto vztahy reprezentují pomocí matic vzdáleností a dosažitelnosti, což usnadňuje výpočty na velkých grafech

(Cormen et al., 2022). V kontextu GNN právě tyto charakteristiky vymezují dosah předávání zpráv (message passing) a kolik vrstev je potřeba k zachycení globálních souvislostí (Hamilton, 2020).

Vzdálenost

Vzdálenost mezi dvěma vrcholy u a v, standardně označována jako $d(u, v)$, je definována jako délka (počet hran) nejkratší cesty spojující tyto vrcholy. Pokud mezi vrcholy neexistuje cesta, pokládáme $d(u, v) = \infty$.

Definice: Nechť $G = (V, E)$ je graf. Vzdálenost $d(u, v)$ mezi vrcholy $u, v \in V$ definujeme jako délku (počet hran) nejkratší cesty z u do v. Neexistuje-li mezi u a v žádná cesta, položíme $d(u, v) = \infty$.

Neboli, u neorientovaných a nehodnocených grafů tato hodnota odpovídá minimálnímu počtu hran, které je nutné překonat při průchodu z jednoho vrcholu do druhého.

Takto definovaná vzdálenost splňuje jisté základní předpoklady. Vzdálenost je vždy nezáporná $d(u, v) > 0$, v neorientovaném grafu je symetrická $d(u, v) = d(v, u)$ a zároveň pro každé tři vrcholy platí trojúhelníková nerovnost $d(u, v) \leq d(u, v) + d(v, w)$.

Excentricita

Excentricita vrcholu v, značená $e(v)$, definuje polohu vrcholu v rámci daného grafu. Tento parametr je určen jako maximální vzdálenost z uvažovaného vrcholu v do jakéhokoliv jiného vrcholu u nacházejícího v daném grafu.

Definice: Excentricita vrcholu v v grafu je maximální vzdálenost z vrcholu v do jiného vrcholu grafu u, tj. $e(v) = max_{u \in V(G)} d(v, u)$.

Průměr grafu

Průměr grafu, označovaný jako $diam(G)$, představuje maximální hodnotu excentricity vyskytující se mezi všemi jeho vrcholy.

Definice: Průměr grafu G je definován jako maximum z excentricit všech jeho vrcholů, tj. $diam\,(G) = max_{v \in V(G)} e(v)$.

Tato veličina odpovídá největší možné vzdálenosti mezi libovolnou dvojicí uzlů v souvislém grafu. Zatímco excentricita je vlastností jednotlivých vrcholů, průměr slouží jako globální charakteristika grafu.

Poloměr grafu

Poloměr grafu, označovaný jako $rad(G)$, představuje minimální excentricitu vyskytující se v množině vrcholů souvislého grafu. Na rozdíl od průměru grafu, který určuje jeho maximální rozlehlost, poloměr identifikuje vrcholy s nejlepší dosažitelností k ostatním vrcholům.

Definice: Poloměr grafu G je definován jako minimální excentricita mezi všemi vrcholy grafu, tj. $rad\ (G) = min\ _{v \in V(G)}\ e(v)$.

Poloměr grafu také slouží k určení tzv. středu grafu, což je množina všech vrcholů V, pro které platí rovnost $e(v) = rad(G)$. Intuitivně jde o vrcholy, které jsou z hlediska nejkratších cest „nejlépe dostupné“ ke zbytku grafu.

Matice vzdáleností

Matice vzdáleností představuje způsob, jak reprezentovat metrické vztahy mezi všemi dvojicemi vrcholů v grafu. Jedná se o čtvercovou matici rozměru $n \times n$, kde n odpovídá počtu vrcholů v grafu. Prvek na pozici (i, j) udává vzdálenost $d(v_i, u_j)$, tedy délku nejkratší cesty mezi těmito body (Rosen, 2013). Na hlavní diagonále této matice se vždy vykytují nulové hodnoty, což znamená, že vzdálenost vrcholu od sebe sama se rovná nule.

Matice zachycuje všechny párové vzdálenosti v grafu a z jejích hodnot lze odvodit globální charakteristiky, například průměr grafu nebo průměrnou vzdálenost.

Matice dosažitelnosti

Matice dosažitelnosti představuje v teorii grafů čtvercovou matici o rozměru $n \times n$, která binárním způsobem vyjadřuje existenci cesty mezi libovolnými dvojicemi vrcholů. Prvek na pozici (i, j) nabývá hodnoty 1, pokud v grafu existuje cesta libovolné délky z vrcholu v_i do vrcholu u_j, v případě, pokud cesta neexistuje je roven hodnotě 0. Matice dosažitelnosti tak přehledně zobrazuje, které dvojice vrcholů jsou v grafu navzájem propojeny cestou.

2.4. Grafové problémy a algoritmy

Grafy představují v rámci diskrétní matematiky jeden ze základních matematických modelů používaných k popisu a analýze reálných systémů. Tyto struktury umožňují popis prvků systému prostřednictvím vrcholů a jejich vzájemných relací pomocí hran. V informatice se graf často využívá k modelování sítí, dopravních systémů či vztahů mezi entitami. Hlavním nástrojem pro práci s těmito modely jsou algoritmy, které v daných systémech identifikují specifické vlastnosti, hledají optimální cesty nebo konstruují podgrafy splňující předem definovaná kritéria (Cormen et al., 2022).

Klíčovým aspektem pro hodnocení efektivity těchto grafových algoritmů je jejich výpočetní náročnost, označována jako algoritmická složitost. Ta udává, jak se kvantifikuje růst nároků na časové a paměťové zdroje v závislosti na velikosti grafu, která je standardně definovaná počtem vrcholů n a počtem hran m.

Podle Rosena (2013) lze úlohy rozdělit na problémy řešitelné v polynomiálním čase (třída P), kam patří například hledání nejkratší cesty nebo minimální kostry, které jsou řešitelné podle známých algoritmů. Poté na NP-těžké úlohy, například problém obchodního cestujícího, pro které zatím není známý žádný algoritmus s polynomiální složitostí. U těchto úloh se v současnosti stále častěji využívají metody umělé inteligence, které místo hledání teoreticky nejlepšího výsledku využívají heuristiky a aproximace k získání dostatečně kvalitních řešení v prakticky přijatelném čase. Tato strategie nabízí kompromis mezi přesností a výpočetní účinností (Russell & Norvig, 2021).

Grafové problémy lze rozdělit do tří základních kategorií. První skupina řeší existenční problémy, jejich cílem je ověřit, zda graf obsahuje objekt s danou vlastností, například eulerovský tah či hamiltonovskou kružnici. Druhou kategorií jsou problémy optimalizační, zaměřené na výběr nejlepšího možného řešení z množiny přípustných variant, což je typicky úloha hledání nejkratší cesty. Poslední skupinou jsou problémy rozhodovací, které posuzují platnost určitého tvrzení o grafu, jako je například izomorfismus grafů nebo otázka barvitelnosti grafu (Rosen, 2013).

Následující podkapitoly se věnují podrobnému rozboru těchto kategorií s důrazem na jejich algoritmické řešení a teoretické limity.

2.4.1. Eulerovský tah, hamiltonovská kružnice

Eulerovský tah

Eulerovský tah se v rámci průchodnosti grafových struktur definuje jako posloupnost vrcholů a hran, v níž se každá hrana daného grafu G vyskytuje právě jednou.

Definice: Nechť $G = (V, E)$ je neorientovaný graf. Eulerovský tah v G je posloupnost vrcholů $v_0, e_1, v_1, e_2, \ldots, e_k, v_k$, kde $v_i \in V$, $e_j \in E$ pro všechna příslušná i, j, každá hrana $e \in E$ se v posloupnosti vyskytuje právě jednou a pro každý $j = 1, \ldots, k$ je hrana e_j incidentní s vrcholy v_{i-1} a v_i.

Je to tedy tah, jehož množina hran je shodná s množinou E a žádná hrana se v něm neopakuje. Pokud tento tah začíná i končí ve stejném vrcholu, hovoříme o uzavřeném eulerovském tahu neboli eulerovském cyklu. Graf, který takový cyklus obsahuje, se nazývá eulerovským grafem.

Existence eulerovského tahu je přímo vázána na stupně jednotlivých vrcholů. To znamená, že v souvislém neorientovaném grafu existuje eulerovský tah právě tehdy, mají-li všechny jeho vrcholy sudý stupeň. Pokud graf obsahuje právě dva vrcholy lichého stupně, existuje v něm otevřený eulerovský tah, který začíná v jednom z těchto vrcholů a končí v druhém. Tato vlastnost umožňuje efektivnější ověření existence eulerovského tahu bez nutnosti přímého hledání všech možných cest, protože rozhodovací proces omezuje na výpočet stupňů vrcholů a ověření souvislosti grafu.

Hledání eulerovského tahu je algoritmicky snadno řešitelná úloha třídy P, neboť existují efektivní deterministické algoritmy, například Hierholzerův s časovou složitostí $O(m)$ (Cormen et al., 2022). Využití metod umělé inteligence proto u této základní úlohy není nutné a uplatňuje se až u zobecněných variant, jako je problém čínského listonoše, které již mají charakter náročných optimalizačních problémů.

Hamiltonovská cesta a kružnice

Hamiltonovská cesta v grafu G je definovaná jako prostá cesta, která prochází všemi vrcholy grafu právě jednou. Pokud tato cesta začíná a končí ve stejném vrcholu, hovoříme o hamiltonovské kružnici. Graf, který obsahuje hamiltonovskou kružnici, se nazývá hamiltonovský graf.

Definice: Nechť $G = (V, E)$ je neorientovaný graf. Hamiltonovská cesta v grafu G je jednoduchá cesta $v_0, v_1, \ldots, v_k$, kde $v_i \in V$ pro všechna i, všechny vrcholy $v_0, v_1, \ldots, v_k$ jsou navzájem různé a množina $\{v_0, v_1, \ldots, v_k\}$ je rovna V, tj. každý vrchol grafu se v cestě vyskytuje právě jednou.

Definice: Hamiltonovská kružnice v grafu $G = (V, E)$ je jednoduchý cyklus $v_0, v_1, \ldots, v_{k-1}, v_k = v_0$, v němž jsou vrcholy $v_0, v_1, \ldots, v_{k-1}$ navzájem různé a množina $\{v_0, v_1, \ldots, v_k\} = V$, tedy každá vrchol se v cyklu vyskytuje právě jednou.

Zatímco u eulerovských tahů je pozornost věnována hranám, v tomto případě se zaměřuje na úplné pokrytí množiny vrcholů V. Zásadní rozdíl oproti eulerovským grafům spočívá v absenci jednoduché a univerzální charakteristiky, která by umožnila okamžité ověření existence takové struktury na základě stupňů vrcholů (Matoušek & Nešetřil, 2009).

Hledání hamiltonovských kružnic patří mezi NP-úplné problémy, u nichž výpočetní náročnost roste s počtem vrcholů exponenciálně (Rosen, 2013). V praxi, zejména v logistice, se tato bariéra překonává využitím metaheuristik a nástrojů umělé inteligence, které v reálném čase generují kvalitní suboptimální řešení (Cormen et al., 2022). Tento přístup je nezbytný pro efektivní zvládnutí úloh typu problému obchodního cestujícího v rozsáhlých síťových strukturách.

2.4.2. Problém nejkratší cesty

Problém nejkratší cesty představuje jednu z optimalizačních úloh definovaných na grafových strukturách, jejímž základem je model váženého grafu. Nechť graf G je orientovaný nebo neorientovaný graf a nechť na množině hran E je definovaná funkce $w: E \rightarrow \mathbb{R}$, která každé hraně přiřazuje reálnou hodnotu reprezentující fyzickou vzdálenost, čas, finanční náklad či jinou sledovanou veličinu.

Definice: Nechť graf $G = (V, E)$ je vážený graf s váhovou funkcí $w: E \rightarrow \mathbb{R}$ a nechť $s, t \in V$. Cesta z s do t je posloupnost vrcholů $P = (v_0 = s, v_1, \ldots, v_k = t)$ taková, že pro každý index $i = 0, \ldots, k-1$ je $(v_i, v_{i+1}) \in E$. Délka cesty P je pak dána součtem $L(P) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$.

Cílem úlohy je nalézt takovou cestu P, pro niž je hodnota $L(P)$ minimální mezi všemi existujícími cestami spojující vrcholy s a t.

V praxi se tento problém objevuje v několika variantách. Nejčastěji jde o hledání nejkratší cesty mezi dvěma pevně danými vrcholy nebo o tzv. jednozdrojový problém, kdy se z jednoho zvoleného vrcholu určují nejkratší cesty do všech ostatních vrcholů grafu. Existuje rovněž varianta hledání nejkratších cest mezi všemi dvojicemi vrcholů. Zatímco v nevážených grafech se délka cesty obvykle chápe jako prostý počet hran, ve vážených grafech výpočet plně reflektuje hodnoty váhové funkce.

Pro řešení jednozdrojového problému ve vážených grafech s nenegativními vahami hran se standardně využívá Dijkstrův algoritmus. Tento postup je založen na principu zpřesňování (relaxaci) odhadů vzdáleností a na výběru vrcholů s aktuálně nejmenší známou vzdáleností od počátečního vrcholu, čímž postupně určuje minimální délky cest do všech dosažitelných vrcholů (Cormen et al., 2022).

Bellman-Fordův algoritmus rozšiřuje tento přístup i na orientované grafy se zápornými vahami hran. Tento postup je založen na principu opakované relaxace všech hran, čímž určuje délky nejkratších cest a současně detekuje záporné cykly. Přítomnost takového cyklu v grafu znamená, že problém nejkratší cesty nemá korektně definované řešení, neboť délku trasy lze v takovém případě nekonečně snižovat (Cormen et al., 2022).

Pochopení problému nejkratší cesty představuje jeden ze základních stavebních kamenů pro návrh metod umělé inteligence nad grafovými strukturami. Matematický aparát nejkratších cest zde slouží jako referenční rámec, k němuž se vztahují pokročilejší přístupy řešící obecnější a výpočetně náročnější varianty těchto úloh. V kontextu umělé inteligence proto klasické algoritmy pro nejkratší cesty fungují nejen jako praktický nástroj, ale také jako teoretický základ pro konstrukci a analýzu učených modelů zaměřených na optimalizační problémy na grafech.

2.4.3. Izomorfismus grafů

Izomorfismus představuje v teorii grafů základní mechanismus pro posuzování strukturální identity dvou grafů.

Definice: Dva grafy G a H jsou izomorfní, pokud existuje bijekce $f : V(G) \to V(H)$ taková, že dvojice vrcholů u, v jsou v G sousední právě tehdy, když $f(u)$ a $f(v)$ jsou sousední v H.

Izomorfismus lze interpretovat tak, že se oba grafy liší pouze přejmenováním svých prvků nebo způsobem geometrického znázornění v rovině, zatímco jejich topologická struktura zůstává identická (West, 2018). Z hlediska diskrétní matematiky je izomorfismus zásadní tím, že definuje relaci ekvivalence na množině všech grafů, a umožňuje tak klasifikovat grafy do tříd se stejnými strukturálními vlastnostmi.

Existenci izomorfního zobrazení mezi dvěma grafy zaručuje shodu ve všech strukturálních charakteristikách, které se v literatuře označují jako grafové invarianty. Pokud jsou grafy izomorfní, musí vykazovat identický počet vrcholů a hran, stejnou posloupnost stupňů vrcholů, shodný počet souvislých komponent a stejné hodnoty průměru grafu. Jakákoliv vlastnost, která je zachována izomorfismem, musí platit pro oba grafy současně. Je však nutné zdůraznit, že shoda v základních invariantech je podmínkou nutnou, nikoliv však postačující.

Problém izomorfismu grafů představuje v informatice specifickou výpočetní výzvu, neboť jeho přesné zařazení mezi lehké a těžké úlohy zůstává nejednoznačné. V oblasti umělé inteligence se k jeho řešení využívají především grafové neuronové sítě, které se pokoušejí rozpoznat strukturální shodu pomocí učení se z dat. Tyto systémy sice pracují s vysokou efektivitou u rozsáhlých databází, avšak narážejí na teoretická omezení, která jim brání v naprosto přesném rozlišení všech neizomorfních struktur (Hamilton, 2020). Propojení matematické teorie s metodami umělé inteligence tak nabízí možnost demonstrovat kontrast mezi exaktním výpočtem a pravděpodobnostním odhadem moderních modelů.

2.4.4. Kruskalův algoritmus

Kruskalův algoritmus patří mezi základní postupy diskrétní matematiky a teorie grafů pro hledání minimální kostry v ohodnoceném grafu. Řadí se mezi hladové algoritmy, protože v každém kroku volí lokálně nejlepší možnost s cílem dosáhnout globálně optimálního řešení (Cormen et al., 2022). Podstata tohoto algoritmu spočívá v postupném přidávání hran s nejnižší vahou za podmínky, že jejich zařazením nevznikne v grafu cyklus, dokud nejsou spojeny všechny vrcholy.

Definice: Nechť $G = (V, E)$ *je neorientovaný ohodnocený graf, kde* $w: E \to \mathbb{R}_{\geq 0}$. *Kostrou grafu G rozumíme takový podgraf* $T = (V, E_T)$, *který je stromem a obsahuje*

všechny vrcholy množiny V. Minimální kostrou grafu G nazýváme takovou kostru T, pro kterou je součet vah hran $\sum_{e \in E_T} w(e)$ minimální mezi všemi kostrami grafu G.

Realizace algoritmu vychází z postupného rozšiřování kostry o jednotlivé hrany. Na začátku se hrany uspořádají podle jejich vah v neklesajícím pořadí. Poté se prochází tato uspořádaná množina hran a každá hrana je do vznikající kostry přidána jen tehdy, pokud její přidání nevytvoří cyklus v dosud vzniklém podgrafu. Díky tomu zůstává výsledná struktura stromem. Algoritmus končí ve chvíli, kdy jsou všechny vrcholy propojeny, případně není možné přidat další hranu, aniž by došlo k porušení acykličnosti.

Pro efektivní zjišťování, zda hrana spojuje dvě různé komponenty, lze využít datovou strukturu disjunktních množin (Union-Find), která umožňuje operace Find a Union v téměř konstantním amortizovaném čase (Cormen et al., 2022). Celková časová složitost Kruskalova algoritmu je pak $O(m \setminus log\ m)$, kde $m = |E|$, přičemž převažující část tvoří třídění hran (Rosen, 2013).

Praktické využití Kruskalova algoritmu se uplatňuje například při návrhu počítačových sítí, plánování dopravní infrastruktury či ve shlukové analýze v datové vědě. V těchto úlohách minimální kostra představuje nejlevnější způsob propojení daných bodů bez nadbytečné redundance. V současnosti se zkoumá také propojení tohoto algoritmu s metodami umělé inteligence, které mohou pomáhat s predikcí dynamicky se měnících vah hran a s optimalizací rozsáhlých grafů pomocí heuristických či neuronových modelů.

2.4.5. Barvení grafů

Barvení grafů představuje jednu z nejvýznamnějších oblastí teorie grafů, která nachází uplatnění v široké škále optimalizačních úloh. Základním konceptem je obarvení vrcholů grafu a hlavní podmínkou je, aby žádné dva sousední vrcholy nebyly označeny stejnou barvou.

Definice: Nechť $G = (V, E)$ je neorientovaný graf. Vlastním (vrcholovým) obarvením grafu G nazveme zobrazení $c : V \rightarrow \{1, 2, \ldots, k\}$, takové, že pro každou hranu $\{u, v\} \in E$ platí $c(u) \neq c(v)$.

Takto definované barvení se v odborné literatuře označuje jako vrcholové vhodné barvení a slouží k modelování situací, kde je nutné eliminovat konflikty mezi vzájemně propojenými prvky (Matoušek & Nešetřil, 2009).

S problematikou barvení souvisí pojem chromatické číslo, značeno $\chi(G)$, které udává minimální počet barev k, pro které existuje vlastní k-obarvení vrcholů grafu G. Chromatické číslo tedy vystihuje vnitřní barevnou složitost grafu a patří k nejpoužívanějším parametrům při jeho klasifikaci. Jeho přesné určení je však výpočetně náročné a patří mezi NP-těžké problémy (Rosen, 2013). V praxi se proto často využívají heuristiky a metody umělé inteligence, které hledají kvalitní obarvení v přijatelném čase místo exaktního výpočtu chromatického čísla.

Historicky nejvýznamnějším výsledkem barvení grafů je věta o čtyřech barvách. Tato věta tvrdí, že každý rovinný graf lze vhodně obarvit nejvýše čtyřmi barvami, tedy pro každý takový graf platí $\chi(G) \leq 4$. Tento problém byl původně motivován snahou o barvení politických map tak, aby jakékoliv dva sousedící státy byly odlišeny různými barvami. Definitivní důkaz věty podali v roce 1976 Appel a Haken, kteří k analýze tisíců specifických konfigurací využili jako první v historii matematiky pomoc výpočetní techniky.

Praktické využití barvení grafů se uplatňuje především při optimalizaci rozvrhů, přidělování síťových frekvencí nebo správu paměti v počítačích. Chromatické číslo v těchto úlohách určuje minimální počet zdrojů potřebných k tomu, aby systém fungoval bez vzájemných konfliktů. Nasazení metod umělé inteligence pak umožňuje tyto složité problémy řešit efektivněji i v situacích, kde běžné matematické postupy narážejí na své limity.

2.5. Způsob zápisu a reprezentace grafů

Reprezentace grafových struktur tvoří nezbytný základ pro jejich analýzu v matematické teorii i v praktických aplikacích. Zatímco v rámci diskrétní matematiky slouží různé způsoby zápisu především k přesné definici objektů a následnému dokazování jejich vlastností prostřednictvím abstrakce, v informatice je zvolený formát určujícím faktorem pro efektivitu algoritmů a celkové nároky na systémové prostředky. Specifický význam pak tento výběr získává v oblasti umělé inteligence, zejména u grafových neuronových sítí, kde forma zápisu přímo definuje, jakým způsobem model vnímá a interpretuje vztahy mezi jednotlivými datovými entitami.

2.5.1. Matematický zápis

Základní formou matematického vymezení grafu je množinový zápis, v němž je graf definován jako uspořádaná dvojice G = (V, E), kde V je konečná neprázdná množina vrcholů a E množina hran mezi nimi. Tento zápis umožňuje studovat strukturu grafu na úrovni vztahů mezi prvky, nezávisle na jejich konkrétní reprezentaci.

Vedle množinového zápisu se graf často prezentuje graficky jako uspořádání bodů a úseček či křivek v rovině. Tato podoba je ve studiu zásadní, protože umožňuje okamžitou vizuální analýzu vlastností. Zároveň platí, že jeden a tentýž graf lze zakreslit nekonečně mnoha způsoby.

Alternativou k množinovému pojetí je zápis pomocí zobrazení, v němž hrany tvoří samostatnou množinu a každé hraně zobrazení přiřazuje její koncové vrcholy. Tento přístup umožňuje rozlišit jednotlivé vazby mezi vrcholy a je vhodný i pro případy, jako jsou multigrafy s více paralelními hranami.

2.5.2. Reprezentace v informatice

Zatímco matematický zápis primárně definuje vztahy, informatická reprezentace se zaměřuje na uložení těchto vztahů v paměti počítače tak, aby s nimi mohly efektivně pracovat algoritmy. Volba konkrétní struktury zásadně ovlivňuje časovou i prostorovou složitost operací (Cormen et al., 2022).

Matice sousednosti

Matice sousednosti se definuje jako čtvercová matice typu $n \times n$, kde n odpovídá celkovému počtu vrcholů v grafu. Jednotlivé prvky a_{ij} nabývají hodnoty 1 v případě existence hrany mezi vrcholy v_i a v_j, zatímco hodnota 0 znázorňuje absenci hrany. U neorientovaných grafů je tato matice symetrická podle hlavní diagonály a nenulové hodnoty na diagonále signalizují přítomnost smyček. Zásadní výhodou této reprezentace je možnost ověřit existenci hrany mezi dvěma libovolnými uzly v konstantním čase. V oblasti umělé inteligence slouží matice sousednosti jako základní vstupní formát pro grafové neuronové sítě (GNN), kde popisuje šíření informací mezi vrstvami v rámci procesu message passing (Rosen, 2013).

Matice incidence

Matice incidence je matice rozměru $n \times m$, kde n je počet vrcholů a m je počet hran. Řádky reprezentují vrcholy, sloupce jednotlivé hrany a prvek a_{ij} je roven 1 právě tehdy, když je vrchol v_i incidentní s hranou e_j. U neorientovaných grafů obsahuje každý sloupec právě dvě jedničky určující koncové vrcholy hrany. Tato reprezentace je vhodná zejména tam, kde hrany nesou vlastní vlastnosti (např. v modelech toků v sítích), ačkoli její paměťová náročnost roste úměrně počtu hran, což může být u hustých grafů limitující.

Seznam sousedů

Seznam sousedů pro každý vrchol uchovává množinu všech uzlů, s nimiž je spojen hranou. Tento zápis je paměťově úsporný, protože ukládá pouze existující vazby, a proto je vhodný zejména pro řídké grafy. Nevýhodou je pomalejší zjišťování existence konkrétní hrany, které vyžaduje lineární prohledání příslušného seznamu (Rosen, 2013).

Seznam hran

V seznamu hran je graf popsán jako množina dvojic (u, v), případně trojic (u, v, w), při zahrnutí vah. Jde o velmi jednoduchou, ale praktickou reprezentaci, která je vhodná pro algoritmy operující přímo s hranami (např. Kruskalův algoritmus) a v knihovnách typu PyTorch Geometric funguje jako standardní vstupní formát pro grafová data (Hamilton, 2020).

2.6. Grafové neuronové sítě (GNN)

Grafové neuronové sítě představují moderní propojení teorie grafů s metodami hlubokého učení a jsou navrženy přímo pro analýzu síťových strukturu složených z uzlů, v diskrétní matematice vrcholů, a hran místo obrázků či textu. Základním princip spočívá v mechanismu, kdy každý uzel v grafu postupně sbírá a vyhodnocuje informace od svých sousedů. (Hamilton, 2020) v této souvislosti uvádí, že tento mechanismus umožňuje převést složité síťové vazby do zjednodušené číselné podoby, se kterou lze výpočetně efektivně pracovat. Díky této schopnosti je možné s vysokou přesností automaticky předpovídat chybějící spojení v sítích nebo zařazovat jednotlivé prvky do odpovídajících kategorií.

2.6.1. Princip fungování a architektura

Architektura grafových neuronových sítí vychází z nezbytnosti transformovat diskrétní topologické struktury do spojitého matematického prostředí. Tento proces, známý jako vnořování (embedding), převádí vrcholy a jejich vztahy na nízkorozměrné vektory či tenzory, které v sobě koncentrují informaci o pozici uzlu i charakteru jeho lokálního okolí (Hamilton, 2020). Právě tato numerická reprezentace umožňuje výpočetním systémům provádět operace hlubokého učení, které by nad čistě diskrétním popisem grafu nebylo možné efektivně realizovat.

Jádrem procesu učení je mechanismus předávání zpráv (messsage passing). V každé vrstvě sítě dohází k tomu, že jednotlivé uzly sbírají a agregují informace od svých bezprostředních sousedů, čímž dochází k jejich bezprostřední aktualizaci jejich vnitřního stavu (Hamilton, 2020). Díky opakovanému provádění tohoto mechanismu se uzly učí vnímat svůj význam v širší síti a model může zachytit i složitější topologické vztahy.

Počet konvolučních vrstev v grafové neuronové síti přímo určuje rozsah tzv. receptivního pole každého uzlu, tedy vzdálenost, ze které je schopen čerpat strukturální data. Každá další vrstva rozšiřuje tento dosah o jednu hranu, takže například třívrstvá architektura umožňuje uzlu integrovat informace ze sousedství vzdáleného až tři kroky (Hamilton, 2020). Tento mechanismus je klíčový pro zachycení širšího topologického kontextu a odborná literatura zároveň upozorňuje, že se s rostoucí hloubkou GNN zvyšuje riziko přehlazení (oversmoothingu), kdy se embeddingy jednotlivých uzlů stávají postupně méně rozlišitelnými.

Pro zachycení složitějších vztahů v datech používají grafové neuronové sítě nelineární aktivační funkce. Nejčastěji jde o funkci ReLU (Rectified Linear Unit), která se po lineárních výpočtech ve vrstvách aplikuje na vektory uzlů a dává modelu schopnost učit se i nelineární závislosti (Hamilton, 2020). Bez těchto nelinearit by síť byla jen soustavou lineárních zobrazení a její vyjadřovací síla by byla výrazně omezená.

Aby se předešlo situaci, kdy se model příliš úzce přizpůsobí konkrétním trénovacím datům, využívá se regularizační technika dropout. Ta spočívá v náhodném a dočasném vyřazování části neuronů během trénovacího procesu, což brání vzniku nežádoucích korelací a tzv. přetrénování (Hamilton, 2020). Tímto způsobem je systém nucen identifikovat robustní a obecná strukturální pravidla místo mechanického zapamatování trénovací

sady. Tato schopnost generalizace je klíčová pro aplikaci naučených principů na nové a dosud neviděné grafové struktury.

2.6.2. Agregace a globální vlastnosti

Pro efektivní fungování GNN je nezbytné definovat počáteční číselnou charakteristiku uzlů, která představuje vstupní příznaky modelu. V případech, kdy uzly nenesou vlastní datový obsah, se k popis lokální topologie sítě využívají strukturální statistiky, nejčastěji v podobě stupně vrcholu (Hamilton, 2020). Tyto počáteční hodnoty tvoří základní číselnou informaci, na jejímž základě může model v prvních vrstvách sítě zahájit proces učení.

Zatímco základní modely GNN generují reprezentace pro jednotlivé uzly, pro predikci vlastností celého grafu je nezbytný globální pooling. Tento proces slučuje vnoření všech vrcholů do jediného vektoru, který reprezentuje graf jako celek (Hamilton, 2020). Takto získaná data umožňují vyhodnocovat globální charakteristiky, jako je souvislost nebo typologie grafu.

2.6.3. Vztah GNN a teorie grafů

Grafové neuronové sítě představují přímou aplikaci principů diskrétní matematiky v oblasti umělé inteligence. Jejich architektura využívá matici sousednosti k definici cest pro šíření informací (message passing) a lokální invarianty, jako je stupeň vrcholu, coby výchozí příznaky (Hamilton, 2020); (Rosen, 2013). Bez těchto matematických základů by modely postrádaly strukturu nezbytnou pro realizaci výpočetních operací.

GNN pomáhají rychle odhadovat řešení u velmi složitých úloh, jako je barvení grafů nebo hledání cest, které by tradiční algoritmy řešily příliš dlouho (Rosen, 2013). V odborné literatuře jsou tyto modely přirovnávány ke klasickým testům izomorfismu, kdy se shoda dvou struktur určuje jednoduše podle vzdálenosti mezi jejich číselnými reprezentacemi (Hamilton, 2020).

Tato synergie nachází uplatnění v mnoha praktických odvětvích. Ve farmacii GNN predikují vlastnosti molekulárních grafů, zatímco v logistice slouží k odhadu optimálních tras v dopravních sítích. V rámci kybernetické bezpečnosti pak analýza transakčních grafů dovoluje identifikovat podezřelé, hustě propojené komponenty, což odpovídá detekci specifických podgrafů či klik (Hamilton, 2020).

3. Metodika

3.1. Cíle a hypotézy

Základem praktické části je porovnání dvou odlišných přístupů při řešení úloh teorie grafů. Klasický přístup využívá přesné deterministické algoritmy, které zaručují správné řešení, ale u NP-úplných problémů rychle narážejí na výpočetní limity. Oproti tomu GNN pracují na principu naučených strukturálních reprezentací a poskytují odhady řešení vycházející z rozpoznaných vzorců v datech. Hlavním cílem práce je kriticky zhodnotit, nakolik tyto modely založené na hlubokém učení dokážou spolehlivě nahrazovat či doplňovat klasické metody a kde leží hranice jejich použitelnosti v kontextu diskrétní matematiky.

K dosažení tohoto hlavního cíle jsou vymezeny následující dílčí cíle:

- sestrojit spolehlivé referenční datasety (ground truth) na základě exaktních výpočtů, v nichž budou grafy doplněny o přesně určené strukturální charakteristiky,
- implementovat klasické grafové algoritmy pro optimalizační a existenční úlohy (např. hledání nejkratších cest, minimálních koster či detekci souvislosti) a vizualizovat jejich průběh pro lepší porozumění jejich činnosti,
- vytvořit, natrénovat a experimentálně ověřit grafovou neuronovou síť určenou k rozpoznávání a klasifikaci strukturálních vlastností v grafech různých parametrů.

Na základě teoretických poznatků o grafových algoritmech a vnitřním fungování GNN jsou formulovány následující hypotézy:

- Implementované klasické grafové algoritmy budou vykazovat chování v souladu s teorií grafů, včetně správného zobrazení a postupu výpočtu minimální kostry (Cormen et al., 2022).
- Model GNN dosáhne vysoké úspěšnosti při klasifikaci jednoduchých typů grafů a při detekci nesouvislosti díky výrazným strukturálním rozdílům.

- U složitějších úloh, jako je odhalování hamiltonovských kružnic a izomorfismu, bude přesnost modelu GNN výrazně nižší, což odhalí limity tohoto přístupu oproti klasickým metodám (Hamilton, 2020).

3.2. Použité technologie

Nástroje zvolené pro praktickou část práce byly vybrány tak, aby podporovaly jak implementaci tradičních algoritmů teorie grafů, tak experimenty s grafovými neuronovými sítěmi.

3.2.1. Základní programovací prostředí

Základním programovacím jazykem byl zvolen Python, a to především pro jeho širokou podporu knihoven zaměřených na teorii grafů a strojové učení (Python Software Foundation, 2025). Nabízí vysokou míru abstrakce, která umožňuje přehledný zápis matematických vztahů do algoritmické podoby, což usnadňuje implementaci složitějších struktur.

Pro vývoj a správu kódu bylo využito prostředí Visual Studio Code (VS Code), zvolené pro svou modularitu a podporu specializovaných rozšíření pro jazyk Python (Microsoft, 2025). Vestavěné ladící nástroje a terminál přispívají k přehledné správě projektu a rychlejší identifikaci chyb (Pyvec, 2022).

Pro iterativní vývoj, vizualizaci průběžných výsledků a rychlé prototypování modelu GNN byl využit Jupyter Notebook (Project Jupyter, 2025). Tento nástroj umožňuje integrovat programový kód, textové poznámky a vizualizace v rámci jediného dokumentu. Možnost spouštět výpočty v izolovaných blocích kódu značně usnadňuje průběžnou kontrolu správnosti dílčích algoritmických kroků a transformací dat.

3.2.2. Knihovny pro práci s grafy a GNN

Pro analytickou část práce byla zvolena knihovna NetworkX, která slouží k formální definici grafových struktur a k výpočtu jejich strukturálních parametrů. Umožňuje rovněž generovat standardní topologie a obsahuje implementace klasických algoritmů, jakými jsou Dijkstrův či Kruskalův algoritmus (Hagberg et al., 2008).

Knihovna PyTorch slouží jako základ pro metody hlubokého učení, která poskytuje infrastrukturu pro tenzorové operace a funguje jako výpočetní rámec pro GNN (PyTorch

Foundation, 2025). Na ni navazuje knihovna PyTorch Geometric (PyG), specializovaná pro práci s grafovými strukturami (Fey & Lenssen, 2019). PyG umožňuje reprezentovat grafy jako datové objekty, stavět architektury GNN pomocí připravených vrstev a převádět data z NetworkX do tenzorového formátu, což zajišťuje plynulé propojení analytické části s prediktivním modelem (PyG Team, 2025).

3.2.3. Zpracování dat a vizualizace

Pro numerické výpočty a práci s maticemi byla využita knihovna NumPy, která nabízí datové struktury pro vícerozměrné pole a širokou škálu lineárně-algebraických operací používaných v Pythonu (NumPy Developers, 2024).

Vizualizaci úloh a průběhu algoritmů zajišťuje knihovna Matplotlib, určená pro tvorbu dvourozměrných grafických výstupů v prostředí Pythonu (Hunter, 2007). V této práci slouží k vykreslování topologie grafů, grafickému znázornění matic sousednosti a k vizuální kontrole predikcí generovaných grafovými neuronovými sítěmi.

Pro efektivní správu a tabulkové zobrazení dat byla zvolena knihovna pandas (The pandas development team, 2025). Výsledky strukturální analýzy i úspěšnost modelů jsou ukládány do objektů typu DataFrame, což zajišťuje jejich přehlednost a snadnou interpretaci (Pyvec, 2024). Tato struktura umožňuje systematickou organizaci dat a efektivní porovnávání výkonnosti jednotlivých experimentů v závěru práce.

3.2.4. Použité dokumentace

Při implementaci praktické části byla klíčovým zdrojem informací oficiální technická dokumentace použitých nástrojů. Ta sloužila k zajištění správné syntaxe, nastavení parametrů a metodické přesnosti správnosti postupu:

- Python Language Reference – pro definování logických struktur a syntaxe (Python Software Foundation, 2025).
- PyTorch a PyTorch Geometric Documentation – pro konfiguraci vrstev GNN a definování trénovacích cyklů ((PyTorch Foundation, 2025); (PyG Team, 2025)).
- pandas Documentation – pro správu a strukturování výsledků analýzy (The pandas development team, 2025).

Využití těchto autoritativních podkladů zajistilo technickou integritu a reprodukovatelnost všech provedených experimentů.

3.3. Způsob generování dat a příprava datasetů

Tato kapitola popisuje proces tvorby datových souborů, které tvoří základ pro trénování a testování GNN. Celý postup byl rozvržen do několika fází, od teoretického návrhu grafových struktur, přes jejich ověření pomocí klasických algoritmů, až po transformace dat do tenzorové podoby vhodné pro strojové učení.

3.3.1. Vývoj od jednotlivých grafů k rozsáhlým datasetům

V úvodním experimentu E0 (Příloha 1) byly nejprve generovány izolované grafy reprezentující základní typy struktur, na kterých lze demonstrovat základní grafové vlastnosti. Pomocí funkcí knihovny NetworkX byly sestrojeny čtyři specifické typy:

- náhodný strom - `nx.random_labeled_tree(8)`
- kružnice C_8 - `nx.cycle_graph(8)`
- úplný bipartitní graf $K_{4,2}$- `nx.complete_bipartite_graph(4, 2)`
- nesouvislý graf vzniklý sjednocením kružnice a cesty – `nx.disjoint_union(nx.cycle_graph(4), nx.path_graph(3))`.

Pro náhodný strom a kružnici byl daný počet vrcholů $n = 8$, pro úplný bipartitní graf $n = 6$ a pro nesouvislý graf $n = 8$.

Navazující experiment E1 (Příloha 4) se zaměřil na generování grafů vhodných pro testování deterministických algoritmů. Pomocí Erdős-Rényiho modelu (`nx.erdos_renyi_graph`) s parametry $n = 7$ pro počet vrcholů a $p = 0{,}5$ pro 50% pravděpodobnost vytvoření hrany. Tím vznikaly náhodné grafy, u nichž byla souvislost vynucena opakovaným generováním, dokud podmínka `nx.is_connected` nebyla splněna. Následně byly hranám náhodně přiřazeny váhy $w \in \langle 5{,}20 \rangle$ pomocí modulu random, čímž vznikly ohodnocené grafy pro určité algoritmy.

3.3.2. Generování datasetů pro GNN

K sestavení datasetů v experiment E2 (Příloha 6) byla využita knihovna NetworkX, která umožňuje přesně generovat grafy, které striktně odpovídají matematickým definicím. Celkově byly vytvořeny čtyři tematické datasety:

Klasifikace typů grafů

Dataset pro úlohu Klasifikace typů grafů (Příloha 6) obsahuje 1000 vzorků, 200 pro každý graf. K jejich konstrukci byly využity generátory s počtem vrcholů v rozsahu $n \in \langle 6,12 \rangle$:

- stromy - `nx.random_labeled_tree`
- kružnice (cykly) - `nx.cycle_graph`
- úplné bipartitní graf - `nx.complete_bipartite_graph`
- úplné grafy - `nx.complete_graph`
- cesty - `nx.path_graph`.

Detekce souvislosti

Pro druhou úlohu Detekce souvislosti (Příloha 6) čítá dataset 600 vzorků rozdělených do dvou tříd. Pozitivní vzorky představují souvislé stromy (`nx.random_labeled_tree`), zatímco negativní vzorky vznikly disjunktním spojením dvou menších úplných grafů pomocí funkce `nx.disjoint_union`, čímž byla garantovaná existence oddělených komponent.

Rozpoznávání izomorfismu

Ve třetí části Rozpoznávání izomorfismu (Příloha 6) bylo vytvořeno 600 párů grafů. U izomorfních dvojic byl jeden z grafů náhodně přeznačen permutací indexů vrcholů pomocí funkce `nx.relabel_nodes`. Tím vznikly struktury topologicky identické, avšak s odlišným maticovým zápisem. Neizomorfní páry byly vytvořeny kombinací odlišných struktur, například cesta a kružnice.

Hamiltonovská kružnice

Pro závěrečnou část Hamiltonovská kružnice (Příloha 6) byl vytvořen dataset o 600 grafech, který využívá jako pozitivní vzorky cykly a úplné grafy, u nichž je existence hamiltonovské kružnice zaručena. Jako negativní vzorky byly zvoleny stromy a cesty, které z definice (vzhledem k přítomnosti listů se stupněm 1) hamiltonovskou kružnici obsahovat nemohou.

3.3.3. Struktura datového vzorku

V rámci experimentu E2 (Příloha 6) byla grafová data generovaná pomocí knihovny NetworkX transformována do formátu objektů `Data` knihovny PyG (Fey & Lenssen, 2019). Základním stavebním prvkem této reprezentace je tenzor. V kontextu knihovny PyTorch se jedná o vícerozměrné pole čísel, které zobecňuje koncepty matic a vektorů. Tenzory v rámci realizovaného modelu umožňují efektivní provádění paralelních matematických operací a jsou klíčové pro proces gradientního učení, neboť uchovávají číselné hodnoty příznaků i topologie grafu v optimalizované podobě. Každý vygenerovaný vzorek je definován následujícími tenzorovými atributy:

- Atribut x (příznaky uzlů): Matice příznaků o rozměrech *[N, 1]*, kde *N* značí počet uzlů. Jako vstupní příznak je využit stupeň uzlu, například hodnota *[2.]* (viz Obrázek 8) znamená, že daný vrchol je spojen se dvěma sousedy.

- Atribut edge_index (indexy hran): Tenzor reprezentující propojení vrcholů ve formátu Coordiante List (COO). První řádek definuje počáteční vrcholy a druhý řádek vrcholy cílové, například sloupec [0,1] indikuje existenci hrany mezi vrcholy 0 a 1. Hrany jsou uloženy obousměrně, což představuje neorientovaný graf a umožňuje mechanismu message passing šířit informace mezi sousedy v obou směrech (Fey & Lenssen, 2019).

- Atribut y (štítek/label): Cílová hodnota (tzv. ground truth), kterou se model učí predikovat. Například hodnota 0 odpovídá typu grafu „Strom“.

Obrázek 8: Detail jednoho grafu v datsetu

```
Počet uzlů: 11
Počet hran (obousměrně): 20
Příznaky uzlů (x) - prvních 5:
tensor([[2.],
        [1.],
        [2.],
        [2.],
        [1.]])
Matice sousednosti (edge_index) - prvních 5 spojů:
tensor([[0, 0, 2, 2, 3],
        [1, 9, 4, 8, 7]])
Štítek (y): 0 (Název: Strom)
```

Zdroj 8: Vlastní výpočet

3.3.4. Rozdělení dat

Aby bylo možné objektivně posoudit schopnost modelu byla data rozdělena na trénovací a testovací sady. Přesné počty vzorků pro jednotlivé úlohy, tak jak byly realizovány v experimentu E2, shrnuje Tabulka 1 níže.

Tabulka 1: Přehled rozdělení datových sad pro jednotlivé úlohy

Úloha	**Celkem vzorků**	**Trénovací sada**	**Testovací sada**	**Poměr (cca)**
Klasifikace typů	1 000	800	200	80 : 20
Souvislost grafu	600	500	100	83 : 17
Izomorfismus	600 párů	480	120	80 : 20
Hamiltonovská kružnice	600	480	120	80 : 20

Zdroj 9: Vlastní data

3.4. Návrh a parametry trénování GNN

Tato kapitola specifikuje vnitřní logiku navrženého modelu a konfiguraci trénovacího procesu, která byla použita v experimentu E2. Zatímco teoretické základy GNN byly rozebrány v kapitole 2.6. Grafové neuronové sítě, následující část se zaměřuje na konkrétní volbu architektury a parametrů pro řešení úloh z diskrétní matematiky.

3.4.1. Architektura modelu (návrh)

Pro účely experimentu byl navržen model v třídě `model_GNN` (Příloha 6). Architektura vychází z konceptu grafových konvolučních sítí (GCN) a je optimalizována pro klasifikaci grafových struktur nižšího řádu.

Základem modelu jsou tři konvoluční vrstvy `GCNConv`, jak ukazuje následující definice třídy (Ukázka kódu 1)

Ukázka kódu 1: Model GNN

```
class model_GNN(nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GNN_Bakalarka, self).__init__()
        torch.manual_seed(42)
        self.conv1 = GCNConv(in_channels, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, hidden_channels)
        self.conv3 = GCNConv(hidden_channels, hidden_channels)
        self.lin = nn.Linear(hidden_channels, out_channels)

    def forward(self, x, edge_index, batch):
        x = self.conv1(x, edge_index).relu()
        x = self.conv2(x, edge_index).relu()
        x = self.conv3(x, edge_index).relu()
        x = global_mean_pool(x, batch)
        x = F.dropout(x, p=0.2, training=self.training)
        return self.lin(x)
```

Zdroj 10: Vlastní kód viz Příloha 6

Pro zajištění reprodukovatelnosti výsledků byla inicializace vnitřních parametrů modelu fixována funkcí `torch.manual_seed(42)`. Architektura využívá tři konvoluční vrstvy, čímž je receptivní pole vrcholů vymezeno na vzdálenost tří hran (Fey & Lenssen, 2019). Dimenze skrytého prostoru, určená parametrem `hidden_channels`, umožňuje kódování složitých strukturních rysů, které jsou následně transformovány nelineární aktivační funkcí `ReLU` pro zachycení složitých topologických vazeb.

Pro klasifikaci grafu jako celku je klíčová funkce `global_mean_pool`. Ta agreguje vnoření všech vrcholů a vypočítává jejich průměr, čímž vytváří jednotný vektorový „otisk“ celého grafu. Před finální lineární vrstvou je aplikována technika `dropout` s pravděpodobností 0,2. Toto opatření slouží jako regularizační prvek, který během trénování náhodně deaktivuje část neuronů, čímž brání přetrénování na specifické instance a nutí model hledat spolehlivější topologické vzorce (Hamilton, 2020).

3.4.2. Transformace dat a vstupní příznaky

Klíčovým hlediskem návrhu je převod diskrétních grafů do tenzorové podoby, kde proces definuje vstupní příznaky, se kterými model pracuje. V rámci implementace byla vytvořena funkce `nx_to_pyg` (Příloha 6), která zajišťuje transformaci objektů z NetworkX do formátu `Data`, který je nezbytný pro zpracování v rámci GNN.

Ukázka kódu 2: Funkce pro převod grafových struktur do tenzorové reprezentace

```
def nx_to_pyg(G, label):
    edges = list(G.edges())
     if len(edges) == 0:
            edge_index = torch.empty((2, 0), dtype=torch.long)
     else:
              edge_index = torch.tensor(edges, dtype=torch.long).t().conti-
    guous()
            edge_index = torch.cat([edge_index, edge_index[[1, 0]]], dim=1)
      x = torch.tensor([[deg] for _, deg in G.degree()], dtype=torch.float)
      y = torch.tensor([label], dtype=torch.long)
      return Data(x=x, edge_index=edge_index, y=y)
```

Zdroj 11: Vlastní kód viz. Příloha 6

Jako primární příznak (atribut x) byl zvolen stupeň vrcholů (`G.degree()`), kde se touto volbou testuje hypotéza, zda skóre grafu (stupně vrcholů) poskytují dostatečnou informaci pro klasifikaci grafů i detekci globálních vlastností. Tenzor `edge_index` pak reprezentuje souvislost grafu, přičemž zdvojení hran je nezbytné pro symetrické šíření informací v neorientovaném prostředí (Fey & Lenssen, 2019).

3.4.3. Konfigurace trénovacího procesu

Proces učení byl realizován pomocí optimalizačního modelu Adam. Tento optimalizér představuje metodu adaptivního odhadu momentů pro jednotlivé parametry a tím podporuje stabilní konvergenci tréninku (Kingma & Ba, 2017). Počet epoch neboli kompletních průchodů modelu celým datasetem během trénování, byl nastaven na 30 pro každou úlohu v experimentu E2. Tento rozsah se ukázal jako dostatečný pro ustálení ztrátové funkce bez zbytečné výpočetní zátěže.

Nastavení tréninkového prostředí v kódu (Příloha 6) odpovídá následujícímu schématu:

Ukázka kódu 3: Optimalizér, trénovací smyčka

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
criterion = nn.CrossEntropyLoss()

for epoch in range(1, 31):
    model_typy.train()
```

Zdroj 12: Vlastní kód viz Příloha 6

Nastavení parametrů bylo přizpůsobeno typu úlohy, pro Klasifikaci typů grafů, Detekci souvislosti a Hamiltonvskou kružnici byla použita učící rychlost $lr = 0{,}01$ a ztrátová funkce `.CrossEntropyLoss()` (detailně popsané v kapitole 3.7. Vyhodnocovací metriky). Úloha Rozpoznávání izomorfismu pracovala s nižší učící rychlostí $lr = 0{,}001$, aby docházelo k jemnější úpravě vah při porovnávání strukturně identických dvojic. Model zde neměl za cíl klasifikovat třídy, ale minimalizovat euklidovskou vzdálenost mezi vnořeními izomorfních grafů. Síť se tak učila generovat pro izomorfní struktury totožné „otisky“ bez ohledu na zvolenou permutaci vrcholů.

3.5. Postupy experimentů

Postup v experimentální části je rozdělen do tří logických celků, které jsou dokumentovány ve zdrojových kódech v přílohách 1 až 7. První experiment E0 je zaměřen na výpočet základních strukturálních charakteristik a generování grafů včetně jejich maticových reprezentací. Druhý experiment E1 zahrnuje implementaci klasických algoritmů pro hledání nejkratší cesty, minimální kostry a barevní grafu. Závěrečný experiment pak spočívá v návrhu a trénovaní GNN, u které je testována schopnost automaticky rozpoznat typy grafů a její topologické vlastnosti.

3.5.1. Klasická strukturální analýza

První experiment E0 (Příloha 1) navazuje na teoretické základy popsané v kapitole 2, konkrétně na problematiku strukturálních charakteristik a způsob reprezentace grafů. Cílem této části bylo ověřit této části bylo ověřit funkčnost standardních algoritmů na vybraném vzorku grafových struktur a připravit data pro navazující úlohy.

Inicializace prostředí a výběr testovacích prvků

Realizace experimentu nejdříve vyžadovala inicializaci vývojového prostředí v jazyce Python. Klíčovými nástroji byly knihovny NetworkX, pandas a Matplotlib, které byly rozebrány v kapitole 3.2.Grafické výstupy a vizualizace matic zajišťovaly vlastní moduly `vykresleni_matice` a `vykresleni_tabulky` (Příloha 2 a 3), které oddělují logiku výpočtů od formátování prezentovaných dat.

Pro analýzu byly zvoleny čtyři typy grafových struktur, které reprezentují, odlišné topologické vlastnosti (Ukázka kódu 4).

Ukázka kódu 4: Definice testovacích grafů

```
test_grafy = [
    (nx.random_labeled_tree(8), "Strom"),
    (nx.cycle_graph(8), "Kružnice $C_8$"),
    (nx.complete_bipartite_graph(4, 2), "Bipartitní $K_{4,2}$"),
    (nx.disjoint_union(nx.cycle_graph(4), nx.path_graph(3)), "Nesou-
vislý graf")
]
```

Zdroj 13:Vlastní kód viz Příloha 1

Náhodný strom byl generován funkcí `nx.random_labeled_tree(8)` s přesně definován počtem vrcholů $n = 8$. Kružnice C_8 byla vytvořena pomocí `nx.cycle_graph(8)`, kde jsou uzly uspořádány do uzavřeného cyklu. Úplný bipartitní graf byl pomocí funkce `nx.complete_bipartite_graph(4, 2)` rozdělen do dvou disjunktních skupin (4 a 2 uzly) se zajištěným propojením každého vrcholu z první skupiny s každým vrcholem ze skupiny druhé. Nesouvislý graf byl konstruován pomocí `nx.disjoint_union(nx.cycle_graph(4), nx.path_graph(3))`, kde tento objekt spojuje kružnici o 4 vrcholech a cestu o 3 vrcholech. Tato funkce automaticky přečísluje vrcholy, aby nedocházelo ke kolizím v indexaci.

Algoritmické výpočty

Pro systematickou analýzu byla vytvořena funkce `analyzuj_vlastnosti` (Příloha 1), která postupně volá matematické algoritmy pro zjištění lokálních i globálních charakteristik.

Ukázka kódu 5: Struktura pro výpočty vlastností grafu

```
vlastnosti = {
    "Typ grafu": nazev,
    "Souvislý": nx.is_connected(G),
    "Komponenty": nx.number_connected_components(G),
    "Skóre": sorted([d for n, d in G.degree()], reverse=True),
    }
```

Zdroj 14: Vlastní kód viz Příloha 1

Nejdříve byla zjišťována souvislost grafu funkcí `nx.is_connected(G)` a následně stanoven počet komponent funkcí `nx.number_connected_components(G)`. Stupeň vrcholu byla vypočtena metodou `G.degree()`, přičemž výsledné skóre bylo seřazeno sestupně pomocí funkce `sorted()` s parametrem `reverse=True` (sestupné seřazení).

Při výpočtech vzdálenostních charakteristik jako je průměr `nx.diameter(G)`, poloměr `nx.radius(G)` a excentricita `nx.eccentricity()`, nastal technický problém. Standardní algoritmy pro výpočty průměru grafu končí chybovým stavem, pokud narazí na nesouvislou strukturu, kde je vzdálenost mezi uzly v různých komponentách definována jako nekonečno. Tento problém byl vyřešen logickou podmínkou a pokud je graf nesouvislý vypíše pro průměr ∞ a pro poloměr a excentricitu „N/A“ (tento výpočet neproběhl).

Ukázka kódu 6: Výpočet vlastností s ošetřením nesouvislosti grafu

```
if nx.is_connected(G):
    vlastnosti["Průměr (diam)"] = nx.diameter(G)
    vlastnosti["Poloměr (rad)"] = nx.radius(G)
    vlastnosti["Excentricita(v0)"]=nx.eccentricity(G, v=list(G.nodes())[0])
else:
    vlastnosti["Průměr (diam)"] = "∞"
    vlastnosti["Poloměr (rad)"] = "N/A"
    vlastnosti["Excentricita (v0)"] = "N/A"
```

Zdroj 15: Vlastní kód viz Příloha 1

Maticová reprezentace a transformace dat

Další krok byl převedení grafických modelů do algebraických struktur. Byly vygenerována tři základní typy matic (Ukázka kódu 7).

Ukázka kódu 7: Generování maticových reprezentací grafu

```
adj = nx.to_numpy_array(g)
inc = nx.incidence_matrix(g).toarray()
dist = nx.floyd_warshall_numpy(g)
```

Zdroj 16: Vlastní kód viz Příloha 1

Matice sousednosti (adj) získána přes `nx.to_numpy_array(g)`. Tato metoda vrací čtvercovou matici $n \times n$ s binárními hodnotami.

Matice incidence (inc) generována funkcí `nx.incidence_matrix(g).toarray()`. Převedení na pole NumPy (`.toarray()`) zajišťuje, že výsledkem je plná matice zachycující incidenci vrcholů a hran.

Matice vzdálenosti vypočítaná Floyd-Warshallovým algoritmem pomocí `nx.floyd_warshall_numpy(g)`, jenž vrací pole délek nejkratších cest mezi všemi dvojicemi vrcholů.

Veškeré výstupy analýzy jsou shromažďovány v objektu `vysledky` pomocí knihovny pandas. Tato datová struktura je následně převáděna do přehledné tabulky funkcí `vykresli_tabulku_vlastnosti` (Příloha 3), což umožňuje efektivní správu a vizuální kontrolu získaných výsledků.

Výsledkem celkového experimentu je poté vykreslení 4 grafů a ke každému tří matic pomocí modulu `vykresli_matici` (Příloha 2).

3.5.2. Implementace algoritmů

Druhý experiment E1(Příloha 4) je zaměř na praktickou aplikaci klasických grafových algoritmů, která byly teoreticky vymezeny v kapitole 2.4. Grafové problémy a algoritmy. Zatímco v experimentu E0 se pracovalo s fixními typy grafů, cílem tohoto experimentu je demonstrovat funkčnost optimalizačních a rozhodovacích algoritmů nad identickou strukturou náhodně generovaného grafu.

Inicializace prostředí

Pro druhý experiment byly využity knihovny NetworkX a pandas, které byly doplněné o modul random pro generování pseudo-náhodných hodnot. Pro vizualizaci výsledků byl využit vlastní modul `vykresleni_ulohy` (Příloha 5).

Generování grafu

Pro objektivní srovnání algoritmů byla vytvořena jedna instance náhodného grafu založená na Erdős-Rényiho modelu, vytvořena funkcí `nx.erdos_renyi_graph()` s parametry *n=7* pro počet vrcholů a *p=0,4* pro pravděpodobnost hrany. Toto nastavení umožnilo zajištění optimální hustoty grafu.

Ukázka kódu 8: Generování ohodnoceného souvislého grafu

```
n_uzlu = 7
G = nx.erdos_renyi_graph(n=n_uzlu, p=0.5)
while not nx.is_connected(G):
    G = nx.erdos_renyi_graph(n=n_uzlu, p=0.5)
for (u, v) in G.edges():
    G[u][v]['weight'] = random.randint(5, 20)
```

Zdroj 17: Vlastní kód viz Příloha 4

Během procesu generování bylo nutné ošetřit dva zásadní technické problémy. Prvním z nich byla častá nesouvislost náhodně generovaných grafů, což bylo problematické pro algoritmy hledání nejkratších cest. Toto bylo vyřešeno implementací smyčky, která vynutí opakování procesu, dokud není splněna podmínka souvislosti (`while not nx.is_connected(G)`).

Druhým problémem byla absence vah hran u standardního generátoru hran. Byl proto přidán cyklus, který každé hraně přiřadil celočíselnou váhu z intervalu $\langle 5{,}20 \rangle$ (`G[u][v]['weight'] = random.randint(5, 20)`), čímž vznikl plnohodnotný ohodnocený graf.

Analýza nejkratších cest

Pro algoritmy hledání nejkratších cest byly zadány počáteční a koncový vrchol, pro daný graf jsou 1. a 6. vrchol. Prvním přístupem bylo zvoleno prohledávání do šířky (BFS – Breadth-First Search), které ignoruje váhy hran a hledá trasu s minimálním počtem hran. Druhým přístupem byl zvolen Dijkstrův algoritmus, který naopak vyhledává cestu s nejmenším součtem vah. V kódu byla pro obě úlohy využita totožná funkce `nx.shortest_path()`, kde byl pro Dijkstrův algoritmus přidán parametr `weight`.

Ukázka kódu 9: Implementace BFS a Dijkstrova algoritmu

```
nodes = list(G.nodes())
start_node, end_node = nodes[0], nodes[-1]
path_bfs = nx.shortest_path(G, source=start_node, target=end_node)
path_dijkstra = nx.shortest_path(G, source=start_node, target=end_node,
weight='weight')
total_weight = int(nx.path_weight(G, path_dijkstra, 'weight'))
```

Zdroj 18: Vlastní kód viz Příloha 4

Konstrukce minimální kostry

Další úlohou byla konstrukce minimální kostry grafu pomocí Kruskalova algoritmu, a to pomocí funkce `nx.minimum_spanning_tree()`. Postup výpočtu spočívá v seřazení všech hran podle jejich vah a jejich postupném zařazení do kostry za podmínky, že nevznikne cyklus. Pro lepší pochopení rozhodovacího procesu a následné vyhodnocení byl přidán cyklus, který pro každou hranu zaznamenává, zda byla přidána nebo vynechána.

Ukázka kódu 10: Postup Kruskalova algoritmu

```
vsechny_hrany = sorted(G.edges(data=True), key=lambda x: x[2]['weight'])
mst = nx.minimum_spanning_tree(G, algorithm='kruskal')
postup_list = []
for u1, v1, d in vsechny_hrany:
    stav = "PŘIDÁNA" if mst.has_edge(u1, v1) else "VYNECHÁNA"
    postup_list.append(f"Hrana ({u1}-{v1}), váha {d['weight']}: {stav}")
```

Zdroj 19: Vlastní kód viz Příloha 4

Vrcholové barevní grafu

Závěrečnou úlohou bylo nalezení vlastního vrcholového barevní grafu pomocí funkce `nx.coloring.greedy_color(G)`. Vzhledem k tomu, že určení chromatického čísla je klasifikováno jako NP-těžký problém, byl zvolen hladový algoritmu (Greedy Coloring). Byl použit parametr `strategy='largest_first'`, kdy algoritmus nejprve obarví vrcholy s nejvyšším stupněm, což často vede k nalezení obarvení s menším počtem barev než u náhodného pořadí vrcholů.

Ukázka kódu 11: Hladový algoritmus barvení grafu se strategií Largest First

```
coloring = nx.coloring.greedy_color(G, strategy='largest_first')
chromatic_number = max(coloring.values()) + 1
```

Zdroj 20: Vlastní kód viz Příloha 4

Veškeré výstupy jsou zpracovány pomocí funkcí `vykresli_tabulku` pro generová strukturovaných protokolů výpočtů a `vykresli_ulohy_komplet` pro vizuální srovnání topologie zadání s nalezeným řešením (Příloha 5).

3.5.3. Analýza pomocí GNN

V závěrečném experimentu E2 (Příloha 6) je testována schopnost GNN identifikovat strukturální vlastnosti bez jejich přímého programování. Na rozdíl od předchozích experimentů E1 a E2, které využívaly deterministické postupy, se model učí rozpoznávat topologické vzorce přímo z maticových reprezentací dat.

Inicializace prostředí

Pro realizaci výpočtů byly použita knihovna PyTorch a její specializovaná nadstavba PyToerch Geometric (PyG). Poté byly využity knihovny NetworkX, Matplotliba modul random, které jsou více rozepsané v kapitole 3.2. Použité technologie. Dále pro vizualizaci byl využit vlastní modul `datove_rozhrani` (Příloha 7).

Architektura modelu GNN

Byl využit navržený model třídy `model_GNN` (Příloha 6), jehož podrobný teoretický návrh a parametry jsou popsány v kapitole 3.4.1 Architektura modelu. Praktická realizace využívá 3 konvoluční vrstvy `GCNConv`, což definuje receptivní pole uzlů do vzdáleností tří hran. Během zkoušení sítě bylo nutné ošetřit riziko tzv. over-smoothingu (přílišného vyhlazování), kdy model ztrácel schopnost rozlišovat strukturálně podobné grafy, protože vnoření (embeddingy) se postupně stávala téměř nerozlišitelnými. Zafixování tří konvolučních vrstev a přidání techniky dropout s parametrem $p = 0{,}2$ pomohlo omezit přeučení a umožnilo síti se soustředit na podstatné topologické vzorce.

Generování datasetů

Pro potřeby trénovaní byly sestrojeny čtyři tematické datasety pospané v kapitole 3.3.2. Generování datasetů pro GNN. Původní model využíval statické vektory a dosahoval úspěšnosti kolem 50 %, neboť postrádal konkrétní informaci o charakterech vrcholů.

Problém byl odstraněn nahrazením statických dat pomocí funkce `nx_to_pyg` do tenzorové podoby, což je podrobněji vysvětleno v kapitole 3.4.2. Transformace dat a vstupní příznaky. Datasety byly poté rozděleny na trénovací a testovací sady přibližně v poměru 80:20 (Kapitola 3.3.4. Rozdělení dat).

Konfigurace trénovacího prostředí

Proces učení byl realizován pomocí optimalizéru Adam s učící rychlostí $lr = 0{,}01$, speciálně pro úlohu Rozpoznávání izomorfismu byla tato hodnota nastavena na $lr = 0{,}001$. Jak je uvedeno v kapitole 3.4.3 Konfigurace trénovacího procesu, model byl trénován po dobu 30 epoch, což se ukázalo jako dostatečné pro ustálení ztrátové funkce.

Realizace výpočetních úloh

Realizace experimentu byla rozdělena podle specifického zaměření úloh, přičemž do každé z nich byly promítnuty technické úpravy reagující na průběžná zjištění.

1. Klasifikace typů grafů

Model kategorizoval 1000 vzorků do pěti typů grafů (strom, cyklus, bipartitní graf, úplný a cesty). Hlavním problémem u této úlohy byla záměna cesty za strom, kvůli jejich strukturální podobnosti. Zatímco cesta má stupně uzlů omezeny hodnotami 1 a 2, strom může obsahovat uzly s vyššími hodnotami. Aby model tyto rozdíly rozlišil vyžadovalo zpřesnit vstupní příznaky pomocí dynamického výpočtu (`G.degree()`) vrcholů místo statických hodnot.

Ukázka kódu 12: Výpočet stupňů jako vstupních příznaků

```
x = torch.tensor([[deg] for _, deg in G.degree()], dtype=torch.float)
```

Zdroj 21: Vlastní kód viz Příloha 6

2. Detekce souvislosti

Binární úloha byla zaměřena na schopnost modelu rozpoznat, zda je graf souvislý a identifikovat přítomnost oddělených komponent. Jako pozitivní vzorky byly využity souvislé stromy, zatímco negativní vzorky byly tvořeny sjednocením dvou úplných grafů (`nx.complete_graph()`) pomocí funkce `nx.disjoint_union`, kde byla zaručena existence nesouvislých grafů.

Ukázka kódu 13: Generování nesouvislých grafů pro dataset

```
G1, G2 = nx.complete_graph(n//2), nx.complete_graph(n-n//2)
dataset.append(nx_to_pyg(nx.disjoint_union(G1, G2), 0))
```

Zdroj 22: Vlastní kód viz Příloha 6

3. Rozpoznávání izomorfismu

Tato úloha se zaměřila na srovnání strukturální identity 600 párů grafů. Polovina datasetu tvořily izomorfní dvojce, u nichž byly náhodně přeznačeny permutací indexy uzlů pomocí funkce `nx.relabel_nodes`, čím vznikly struktury, které jsou topologicky identické, ale mají odlišný maticový zápis. Pro neizomorfní dvojice byly vytvořeny různé dvojice ze stromu, cyklu, cesty nebo kompletního grafu.

Model zde narážel na neschopnost rozpoznat stejné grafy při náhodném permutování indexů. Problém byl vyřešen úpravou trénovací smyčky snížením učící rychlosti na $lr = 0{,}00$ a minimalizaci euklidovské vzdálenosti mezi vnořeními (embeddingy) grafů.

Ukázka kódu 14: Výpočet euklidovské vzdálenosti mezi vnořeními grafů

```
emb1 = model(g1.x, g1.edge_index, torch.zeros(g1.x.size(0),
dtype=torch.long))
emb2 = model(g2.x, g2.edge_index, torch.zeros(g2.x.size(0),
dtype=torch.long))
dist = torch.norm(emb1 - emb2, p=2)
```

Zdroj 23: Vlastní kód viz Příloha 6

Funkce `torch.zeros()` vrací tenzor, čímž umožňuje mechanismu globálního poolingu vytvořit vnoření (embedding) grafu. Podobnost těchto vnoření je následně vyhodnocována pomocí euklidovské vzdálenosti `torch.norm()`, jejíž minimalizace během trénování nutí model ignorovat pořadí uzlů a soustředit se výhradně na jejich identickou topologii.

4. Hamiltonovská kružnice

V poslední úloze model hledal přibližné řešení NP-úplného problému identifikace hamiltonovské kružnice v grafu. Aby byla zajištěna kvalita dat (ground truth), byl dataset sestaven z vzorků, které matematicky zaručí hamiltonovskou kružnici, to jsou cykly a úplné grafy, a vzorků u nichž z definice nelze najít kružnici, a to stromy a cesty.

Ukázka kódu 15: Konstrukce datasetu pro úlohu hledání hamiltonvské kružnice

```
dataset.append(nx_to_pyg(random.choice([nx.cycle_graph(n),          nx.com-
plete_graph(n)]), 1))
dataset.append(nx_to_pyg(random.choice([nx.random_labeled_tree(n),
nx.path_graph(n)]), 0))
```

Zdroj 24: Vlastní kód viz Příloha 6

Tím byla odstraněna nejistota z dat, která se objevovala v počátečních iteracích u zcela náhodných grafů, kde nebylo možné bez složitého algoritmu správnost predikce ověřit.

Monitorování a vizualizace výsledků

Trénovací proces byl sledován měřením ztrátové funkce a přesnosti. Pro přehlednou interpretaci byly výsledky zpracovány v modulu `datove_rozhrani` (Příloha 7), která vytváří tabulky pro srovnání predikcí s realitou.

3.6. Metriky vyhodnocení

Pro objektivní posouzení úspěšnosti navržených postupů a ověření stanovených hypotéz bylo nezbytné definovat metriky, které umožňují kvantifikovat výkonnost jak klasických algoritmů, tak modelů založených na hlubokém učení.

Přesnost (accuracy) představuje primární ukazatel úspěšnosti u úloh v experimentu E2, konkrétně Klasifikace typů grafů, Detekce souvislosti a Hamiltonovská kružnice. Je vyjadřována jako procentuální poměr správně klasifikovaných vzorků vůči celkovému počtu testovaných grafů v testovací sadě.

Ztrátovou funkcí (Cross-Entropy Loss) je měřen matematický rozdíl mezi pravděpodobnostním odhadem modelu a skutečnou hodnotou (label) během trénování. Pokles hodnot v čase značí, že se vnitřní váhy GNN úspěšně přizpůsobují strukturním datům a model se blíží k optimálnímu nastavení. Pro úlohu Rozpoznávání izomorfismu bylo použito měření euklidovské vzdálenosti mezi vnořeními (embeddingy) dvou grafů.

U úloh v experimentech E0 a E1 slouží vizuální kontrola k ověření shody grafických výstupů a maticového zápisu s teoretickými předpoklady diskrétní matematiky. V experimentu E2 je tato metoda doplněna o barevné kódování (zelená pro shodu, červená pro chybu), které poskytuje okamžitou zpětnou vazbu o spolehlivosti odhadů v testovací sadě.

4. Výsledky, jejich interpretace a diskuse

4.1. Strukturální analýza a reprezentace

První experiment E0 se zaměřil na ověření přesnosti generování grafů a jejich následnou přeměnu do algebraické podoby. Tato část potvrzuje, že zvolené prostředí (knihovna NetworkX) správně interpretuje teoretické koncepty diskrétní matematiky a poskytuje validní data pro další analýzu.

4.1.1. Ověření vlastností vybraných typů grafů

Výsledky strukturální analýzy čtyř vybraných grafů jsou shrnuty v tabulce (Obrázek 9).

Obrázek 9: Vygenerovaná tabulka vlastností grafů

Typ grafu	Souvislý	Komponenty	Skóre	Průměr (diam)	Poloměr (rad)	Excentricita (v0)
Strom	Ano	1	[3, 2, 2, 2, 2, 1, 1, 1]	5	3	5
Kružnice C_8	Ano	1	[2, 2, 2, 2, 2, 2, 2, 2]	4	4	4
Bipartitní $K_{4,2}$	Ano	1	[4, 4, 2, 2, 2, 2]	2	2	2
Nesouvislý graf	Ne	2	[2, 2, 2, 2, 2, 1, 1]	∞	N/A	N/A

Zdroj 25: Vlastní zpracování

V případě stromu byla potvrzena souvislost a naměřené skóre [3, 2, 2, 2, 2, 1, 1, 1] ukazuje přítomnost tří koncových vrcholů (listů) a jednoho centrálního vrcholu se stupněm 3. Excentricita vrcholu v_1 dosahuje hodnoty 5 a je totožná s průměrem, který se podle definice počítá jako maximální excentricita. To znamená, že z počátečního vrcholu k nejvzdálenějšímu vrcholu (v_7) vede cesta o délce 5 hran, což představuje maximální možnou vzdálenost.

Protiklad stromu představuje kružnice C_8, u níž lze pozorovat absolutní symetrii. Hodnoty průměru, poloměru a excentricity jsou zde rovny 4 a skóre je pro všechny vrcholy rovné 2.

Úplný bipartitní graf $K_{4,2}$ vykazuje při celkovém počtu šesti vrcholů průměr pouze 2. To vyplývá z definice úplného bipartitního grafu, kdy každý vrchol z jedné partity je spojen se všemi vrcholy partity druhé. K dosažení libovolného vrcholu v rámci celého

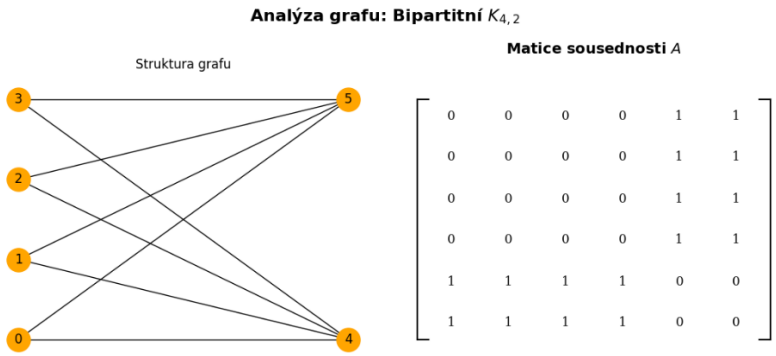
grafu jsou tedy zapotřebí maximálně dvě hrany. O tom vypovídají i hodnota 2 pro průměr a excentricitu.

U nesouvislého grafu byla pro průměr vypočítána hodnota ∞, což může naznačovat chybu, ale v diskrétní matematice je to standardní způsob zápisu, že mezi některými vrcholy nevede žádná cesta. Algoritmus správně odhalil že graf není souvislý a obsahuje dvě komponenty.

4.1.2. Analýza maticových zápisů

Pro každý graf byla vypsána matice sousednosti, incidence a vzdálenosti. Celkové výsledky jsou součástí příloh (Přílohy 9,10,11,12), ale v kapitole se místo popisu všech matic pro jednotlivé grafy zaměřením na specifické úkazy.

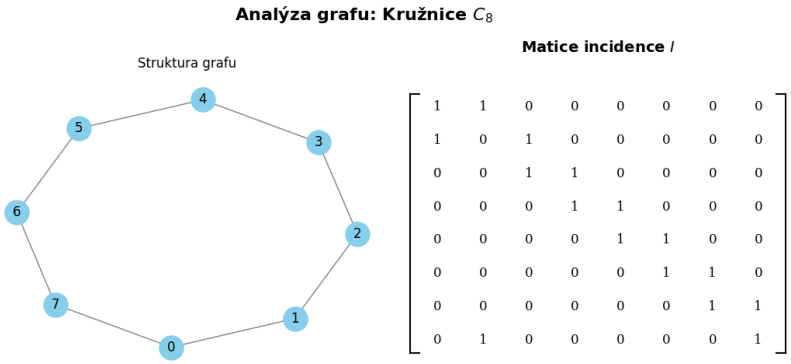
Obrázek 10: Bipartitní graf a matice sousednosti



Zdroj 26: Vlastní zpracování viz Příloha 11

U bipartitního grafu jsou v matici sousednosti vidět dva bloky nul. Protože vrcholy 0,1,2 a 3 tvoří první partitu a nejsou spojeny hranou mezi sebou, a to samé pro vrcholy 4,5.

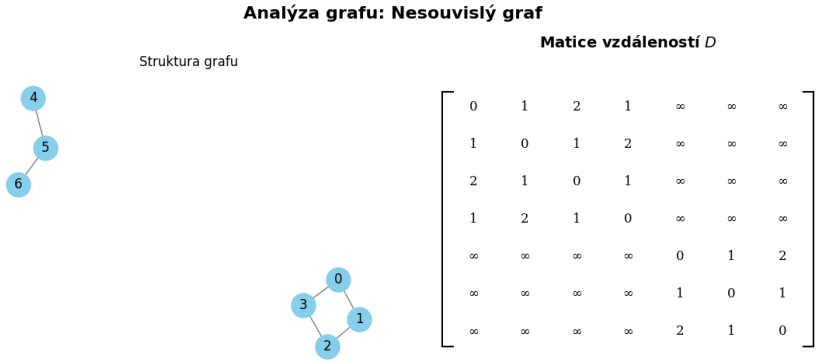
Obrázek 11: Kružnice a matice incidence



Zdroj 27: Vlastní zpracování viz Příloha 10

U kružnice je v každém řádku matice incidence právě jedna jednička. Toto odpovídá skutečnosti, že každý vrchol je incidentní právě s dvěma hranami.

Obrázek 12: Nesouvislý graf a matice vzdáleností



Zdroj 28: Vlastní zpracování viz Příloha 12

U nesouvislého grafu se matice vzdálenosti rozpadla na několik oblastí. Pro vrcholy 0 až 3 (první čtyři řádky a sloupce) tvoří souvislý blok hodnot, stejně tak vrcholy 4, 5 a 6 (zbylé řádky a sloupce). Zbylé části mezi těmito skupinami značí oddělené komponenty, což je v matici reprezentováno hodnotou ∞. Matice je symetrická podle hlavní diagonály.

4.2. Porovnání klasických algoritmů

V této kapitole jsou prezentovány výsledky experimentu E1, kde byly použity deterministické algoritmy na grafech. Pro srovnání byl zvolen stejný graf o sedmi vrcholech pro všechny čtyři algoritmy. Použité algoritmy v praxi potvrdily teoretické předpoklady diskrétní matematiky a úspěšně nalezly optimální řešení pro všechny definované úlohy.

Pro úlohy bylo vygenerováno zadání (Příloha 13) a následně jeho konkrétní řešení (Obrázek 13), které je analyzováno v níže uvedených podkapitolách.

Obrázek 13: Algoritmy a jejich řešení

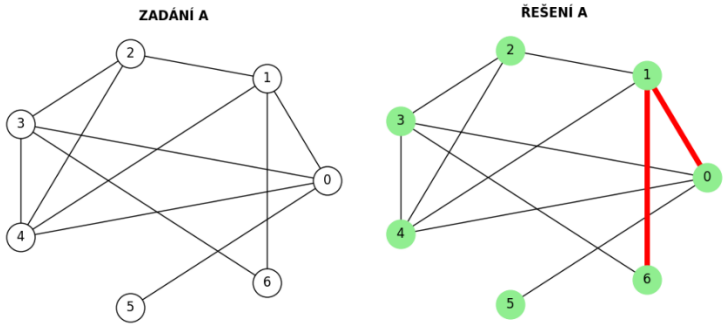
ID	Algoritmus	Klíč řešení
A	Nejkratší cesta (BFS)	0 -> 1 -> 6 (Délka: 2 hran)
B	Dijkstrův algoritmus	0 -> 3 -> 6 (Celková váha: 26)
C	Minimální kostra (MST)	Váha kostry: 63
D	Barvení grafu	Chromatické číslo: 3

Zdroj 29: Vlastní zpracování viz Příloha 13

4.2.1. Hledání nejkratších cest

První úloha srovnává dva odlišné přístupy k hledání nejkratší cesty mezi vrcholy v_0 a v_6.

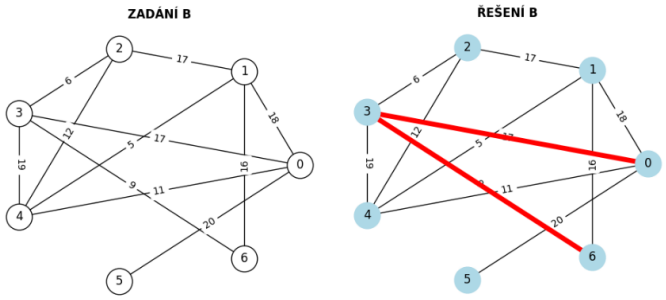
Obrázek 14: Algoritmus BFS pro hledání nejkratší cesty



Zdroj 30: Vlastní zpracování viz Příloha 14

Algoritmus BFS (Breadth-First Search) označil nejkratší cestu $0 \rightarrow 1 \rightarrow 6$. Zde je kritérium definováno minimálním počtem hran, který zde činí 2 hrany. BFS ignoruje váhy hran, což potvrzuje jeho vhodnost pro neválené grafy.

Obrázek 15: Dijkstrův algoritmus pro hledání nejkratší cesty



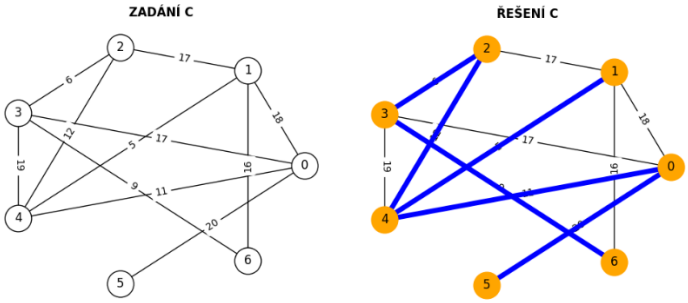
Zdroj 31: Vlastní zpracování viz Příloha 14

Naproti tomu Dijkstrův algoritmus na stejné struktuře nalezl cestu 0 → 3 → 6. Ačkoliv má cesta shodný počet hran jako v případě BFS, vzhledem zohlednění vah hran vede tato cesta přes jiný vrchol. Celková váha nejkratší cesty je 26 (17 + 9). Tento výsledek ukazuje schopnost algoritmu najít nejlevnější a nejkratší cestu v rámci váženého grafu.

4.2.2. Konstrukce minimální kostry

Další úloha byla zaměřena na nalezení podgrafu, který propojuje všechny vrcholy s minimálním součtem vah hran při zachování podmínky acykličnosti. Výsledná minimální kostra dosáhla celkové váhy 63.

Obrázek 16: Kruskalův algoritmus pro konstrukce minimální kostry



Zdroj 32: Vlastní zpracování viz Příloha 14

Důkazem správnosti algoritmu je vygenerovaná tabulky rozhodovacího procesu (Obrázek 17). Algoritmus uplatnil hladový přístup, kdy si nejdříve seřadil hrany od nejmenší váhy, a poté vyhodnocuje jejich přidání a kontroluje situace, ve kterých by přidání vedlo ke vzniku cyklu.

Obrázek 17: Vygenerovaná tabulka postupu rozhodování v Kruskalově algoritmu

Hrana (Váha)	Rozhodnutí algoritmu
Hrana (1-4), váha 5	PŘIDÁNA
Hrana (2-3), váha 6	PŘIDÁNA
Hrana (3-6), váha 9	PŘIDÁNA
Hrana (0-4), váha 11	PŘIDÁNA
Hrana (2-4), váha 12	PŘIDÁNA
Hrana (1-6), váha 16	VYNECHÁNA
Hrana (0-3), váha 17	VYNECHÁNA
Hrana (1-2), váha 17	VYNECHÁNA
Hrana (0-1), váha 18	VYNECHÁNA
Hrana (3-4), váha 19	VYNECHÁNA
Hrana (0-5), váha 20	PŘIDÁNA

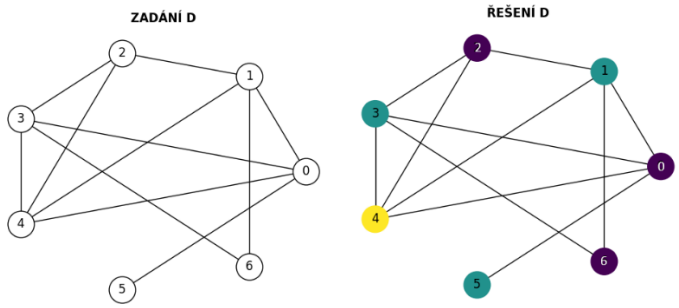
Zdroj 33: Vlastní zpracování

Z tabulky o postupu algoritmu vyplývá, že prvních pět hran bylo přidáno. Hrana (1,6) byla vynechána, přestože s váhou 16 byla zamítnuta, ačkoliv měla nižší váhu než následná přijatá hrana (0,5) s váhou 20. Toto rozhodnutí bylo nezbytné, protože po přidání hrany (1,6) by vznikl cyklus. Algoritmus tak potvrdil schopnost upřednostnit strukturální integritu (acykličnost) před lokální výhodností.

4.2.3. Barvení grafu

Poslední úloha se věnovala problému vrcholového barvení grafu, bez shody barev u sousedních vrcholů. S využitím Larfgest First (viz Kapitola 3.5.2. Implementace algoritmů), která prioritně barví vrcholy s nejvyšším stupněm, bylo dosaženo chromatického čísla 3.

Obrázek 18: Vrcholové barvení grafu



Zdroj 34: Vlastní zpracování viz Příloha 14

Vizuální kontrola potvrdila, že obarvení proběhlo podle očekávání. Například vrchol v_0 je obarven fialovou barvou, zatímco jeho sousedé jsou rozlišení žlutou a tyrkysovou. Vzhledem k hustotě hran testovaného grafu se zvolený postup ukázal jako vysoce efektivní pro dosažení přípustného obarvení, čímž byla eliminována potřeba náročného exaktního výpočtu.

4.3. Schopnost a limity GNN

Experiment E2 je poslední fází praktické části práce, v níž byly testovány schopnosti GNN při řešení čtyř odlišných úloh. Model byl trénován po dobu 30 epoch, což se ukázalo jako optimální pro ustálení parametrů.

4.3.1. Klasifikace typů grafů

Cílem první úlohy bylo naučit model rozpoznat pět základních grafových struktur (cesty, kružnice, stromy, úplné a bipartitní grafy).

Obrázek 19: Výsledky trénovacího procesu a úspěšnosti Klasifikace typů grafů

	Epocha	Ztráta	Acc %
0	5	0.7724	61.67
1	10	0.6242	72.17
2	15	0.5833	73.67
3	20	0.5349	73.00
4	25	0.6313	68.00
5	30	0.4865	75.50

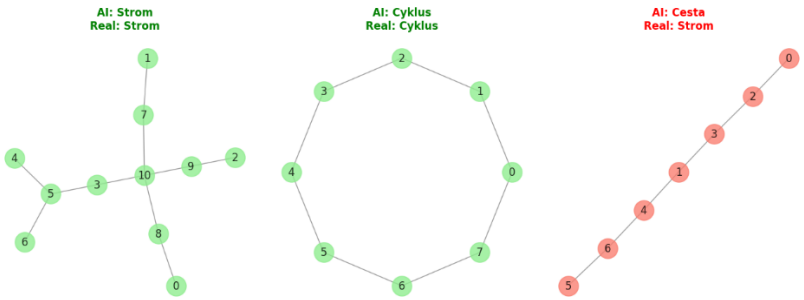
	Kategorie	Úspěšnost (%)
0	Strom	64.91
1	Cyklus	100.00
2	Bipartitní	90.98
3	Úplný	83.33
4	Cesta	75.00

Zdroj 35: Vlastní zpracování

Z výše uvedené vizualizace trénovacího procesu vyplývá, že přesnost modelu dosáhla po 30 epochách hodnoty 75,5 %. Ztrátová funkce vykazuje plynulý pokles již od počátečních fází trénování, což naznačuje, že při prodloužení trénovacího procesu by model pravděpodobně dosáhl ještě větší úspěšnosti. Současná úspěšnost však plně postačuje k potvrzení hypotézy, že GNN dokáže identifikovat základní geometrické vzorce zakódované v maticových reprezentacích.

Analýza konkrétních predikcí (Obrázek 19) ukazuje, že ne všechny grafy jsou pro model čitelné.

Obrázek 20: Ukázky úspěšné a chybné klasifikace typů grafů modelem GNN



Zdroj 36: Vlastní zpracování viz Příloha 15

Záměna cesty za strom je teoreticky opodstatněná. Tomu odpovídá i míra úspěšnosti v tabulce úspěšnosti pro každý graf (Obrázek 19), kde pro strom a cestu dosahuje nejmenších hodnot. Oba grafy sdílejí identické invarianty, jako je acykličnost a nízká hustota hran. Model vnímá tuto podobu primárně přes lokální distribuci stupňů vrcholů, což při omezeném natrénování vede k velmi podobným vnořením obou struktur.

4.3.2. Detekce souvislosti

V rámci této úlohy byla testována schopnost modelu detekovat souvislost grafu. Model vykazoval velmi přesvědčivé výsledky již na začátku učícího procesu.

Obrázek 21: Výsledky trénovacího procesu a úspěšnosti Detekce souvislosti

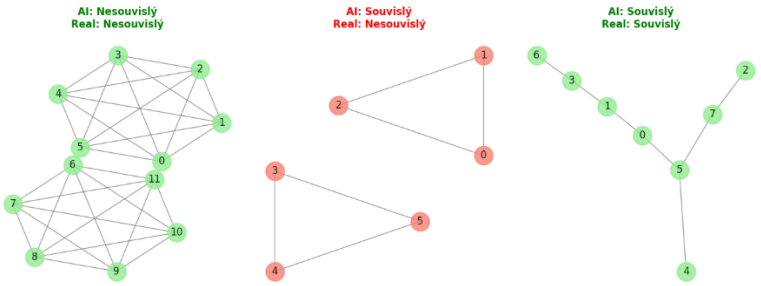
	Epocha	Ztráta	Acc %
0	5	0.2367	90.33
1	10	0.2010	93.00
2	15	0.1789	93.00
3	20	0.1286	93.00
4	25	0.1360	93.00
5	30	0.1392	93.00

	Kategorie	Úspěšnost (%)
0	Nesouvislý	85.52
1	Souvislý	100.00

Zdroj 37: Vlastní zpracování

Hodnota přesnosti se stabilizovala velmi rychle. Bylo tak potvrzeno, že existence izolovaných komponent představuje pro GNN jeden z nejlépe identifikovatelných příznaků. Vysoká úspěšnost detekce souvislosti vyplývá z mechanismu message passing, u něhož topologické bariéry mezi komponentami brání šíření informací.

Obrázek 22: Ukázky úspěšné a chybné detekce souvislosti modelem GNN



Zdroj 38: Vlastní zpracování viz Příloha 16

4.3.3. Rozpoznávání izomorfismu

Úloha rozpoznávání izomorfismu, v níž model identifikuje strukturní identity dvou různě indexovaných grafů, odhalila schopnost modelu nerozhodovat podle zápisu dat ale vnímat topologii grafu.

Obrázek 23: Výsledky trénovacího procesu a úspěšnosti Rozpoznávání izomorfismu

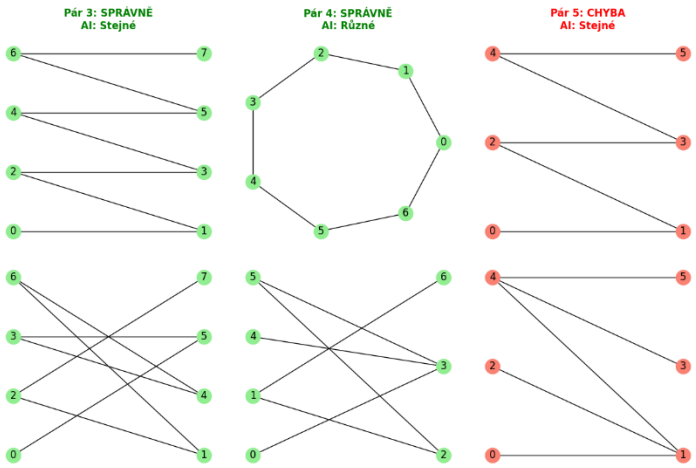
	Epocha	Ztráta	Acc %
0	5	0.2953	73.00
1	10	0.2856	72.25
2	15	0.2785	72.50
3	20	0.2641	72.25
4	25	0.2371	82.50
5	30	0.2001	85.50

	Kategorie	Úspěšnost (%)
0	Stejné	100.00
1	Různé	47.92

Zdroj 39: Vlastní zpracování

Přesnost se v průběhu 30 epoch postupně zvyšovala, což dokládá, že se model naučil identifikovat shodu nezávisle na číslování vrcholů.

Obrázek 24: Ukázky úspěšného a chybného rozpoznávání izomorfismu modelem GNN



Zdroj 40: Vlastní zpracování viz Příloha 17

Na základě zobrazených výsledků (Obrázek 24) model prokázal u Páru 3 schopnost modelu identifikovat izomorfní shodu i při náhodném přeznačení vrcholů. Naopak u Páru 5 chybné určení odhalilo limity modelu, kdy pravděpodobně u nevýrazných struktur bez cyklů se rozhoduje pouze na základě počtu prvků, nikoliv jejich přesného uspořádání.

4.3.4. Hamiltonovská kružnice

V poslední a nejnáročnější úloze hledání hamiltonovských kružnic, které spadá do NP-úplných problémů, bylo cílem ověřit, zda model dokáže identifikovat globální vlastnosti grafu.

Obrázek 25: Výsledky trénovacího procesu a úspěšnosti Hamiltonovská kružnice

	Epocha	Ztráta	Acc %
0	5	0.4469	80.94
1	10	0.4685	72.19
2	15	0.1974	96.56
3	20	0.0374	99.69
4	25	0.0097	100.00
5	30	0.0047	100.00

	Kategorie	Úspěšnost (%)
0	Bez Hamiltona	100.0
1	Má Hamiltona	100.0

Zdroj 41: Vlastní zpracování

Navzdory složitosti problému byla během trénovaní identifikována schopnost modelu rozlišit základní rozdíly mezi garantovanými hamiltonovskými a nehamiltonovskými strukturami. Kolem desáté epochy vykazoval model určitou míru nejistoty, ale pak došlo k zlomu, kdy se model zlepšoval až dosáhl přesnosti 100 %.

Obrázek 26: Ukázky úspěšného a chybného hledání hamiltonovské kružnice modelem GNN

Zdroj 42: Vlastní zpracování viz Příloha 18

Jak je patrné z Obrázku 26, model vykazuje stoprocentní shodu s realitou u všech prezentovaných grafů. Model správně určil přítomnost kružnice u všech variant cyklů a hustých grafů, a to samé i pro stromy a cesty, které kružnici nemají. Tuto záležitost naznačuje i tabulka úspěšnosti (Obrázek 25), kde obě varianty mají 100 %.

4.4. Diskuse

Výsledky tří po sobě jdoucích experimentů poskytují celkový pohled na problematiku reprezentace a algoritmického zpracování grafových struktur. Zatímco úvodní experimenty E0 a E1 potvrdily matematickou přesnost klasických metod, experiment E2 s modelem GNN odhalil hranice mezi exaktním výpočtem a neurální heuristikou, tedy mezi algoritmickou garancí správnosti a statisticky naučeným odhadem. Tato část shrnuje získaná data a hodnotí možnosti využití AI v diskrétní matematice.

4.4.1. Klasické algoritmy a GNN

Srovnání experimentů E1 a E2 ukazuje odlišný přístup k řešení grafových úloh. Deterministické algoritmy, jako je Dijkstra nebo Kruskal, poskytují jednoznačné výsledky, ale u NP-úplných problémů by narážely na časovou složitost. GNN naopak při natrénování a správném nastavení parametrů dokážou rozpoznat klíčové vlastnosti grafu. Naznačuje to, že GNN umějí zachytit globální invarianty, které jsou pro klasické algoritmy bez

úplného prohledávání stavového prostoru nepostřehnutelné. GNN tedy nepředstavují přímou náhradu exaktní matematiky, nýbrž její doplněk v podobě rychlé heuristiky.

4.4.2. Chybovost GNN a lokální kontext

Analýza chyb v experimentu E2, zejména záměna cesty za strom, odhaluje limity tzv. lokálního kontextu. Mechanismus message passing seskupují informace z bezprostředního okolí uzlů, což komplikuje rozlišení grafů se shodným rozdělením stupňů, ale odlišnou globální topologií. Tato skutečnost se projevila i u úlohy izomorfismu, kde model vykazoval nejistotu u struktur postrádajících výrazné lokální rysy.

Záměna cesty za strom indikuje, že zvolených 30 epoch postačuje k zachycení acykličnosti, nikoliv však k jemnému rozlišení mezi cestou a stromem. Pro zvýšení přesnosti se jeví jako nezbytné doplnit vstupní data o globální metriky, jako jsou průměr grafu nebo strukturální charakteristiky matic, které byly předmětem analýzy v části E0.

4.4.3. Stabilita a rychlost učení

Překvapivým zjištěním byla vysoká rychlost u úloh Detekce souvislosti a Hamiltonovské kružnice. Skok v přesnosti kolem patnácté epochy u hamiltonovského problému naznačuje, že síť nalezla efektivní strukturní „zkratky“, například identifikaci uzlů se stupněm 1, které existenci kružnice automaticky vylučují.

U rozpoznávání izomorfismu byl vývoj pozvolnější, což odpovídá vysoké náročnosti učení vnoření (embeddingů), které musí zůstat neměnné vůči jakékoli permutaci vrcholů. Výsledky přesto potvrzují, že GNN dokáže ignorovat náhodné přeznačení uzlů a soustředit se na strukturní podstatu grafu, což je vlastnost zásadní pro analýzu rozsáhlých informačních sítí.

4.4.4. Limity a budoucí vylepšení

Limity experimentu E2 jsou dány zejména rozsahem trénovacích dat a fixním počtem 30 epoch. U rozsáhlejších struktur se stovkami uzlů lze předpokládat pokles přesnosti, neboť se informace v rámci mechanismu message passing musí šířit na delší vzdálenosti, což vede k jejich postupnému rozmělnění.

Budoucí výzkum by se mohl orientovat na pokročilejší architektury, například Graph Attention Networks (GAT). Tyto modely umožňují upřednostňovat váhu jednotlivých

hran a zvyšovat tak tak výsledné predikce. Další potenciál spočívá v rozšíření funkčnosti celého prostředí. Zatím pracuje s předem připravenými sadami grafů, ale jeho návrh by bylo možné dále rozšířit směrem k plně interaktivnímu prostředí.

Získané poznatky zároveň naznačují přínos pro vzdělávání v diskrétní matematice. Vytvořené interaktivní prostředí a vizualizace mohou sloužit jako výuková pomůcka, díky níž je studentům umožněno okamžitě vidět dopady strukturních změn na maticové zápisy či výsledky algoritmů. Tento přístup dává studentům možnost v reálném čase zkoumat vlastnosti grafů a pozorovat, jak umělá inteligence reaguje na změny v topologii, což jim výrazně usnadňuje přechod od abstraktních definic k praktickému porozumění.

5. Závěr

Předložená bakalářská práce se věnovala problematice použití nástrojů umělé inteligence v oblasti diskrétní matematiky se specifickým zaměřením na teorii grafů. Hlavním cílem bylo posoudit, nakolik mohou tyto technologie nahradit či doplnit klasické algoritmy, a jaký potenciál tyto nástroje nabízejí pro výuku diskrétní matematiky.

Stanovené dílčí cíle byly v průběhu práce postupně naplněny. Nejprve došlo k sestavení spolehlivých referenčních datasetů s využitím knihovny NetworkX. Tato fáze byla nezbytná pro následnou objektivní validaci modelu, neboť poskytla přesnou matematickou jistotu o vlastnostech generovaných grafů. Dále byly implementovány klasické grafové algoritmy, jako jsou hledání nejkratší cesty nebo konstrukce minimální kostry, jejichž vizualizace umožnila srovnání a pochopení postupů algoritmu. Závěrečným dílčím krokem byl návrh, trénink a ověření grafové neuronové sítě, která prokázala schopnost identifikovat topologické vzorce bez přímého programování logických pravidel.

Výsledky provedených experimentů umožnily shrnout získaná zjištění a vyhodnotit platnost stanovených hypotéz. První hypotéza předpokládající, že klasické algoritmy budovu vykazovat chování s teoretickými předpoklady, se plně potvrdila. Tyto metody zůstávají nezbytným standardem pro úlohy vyžadující absolutní přesnost, zejména při výpočtech minimálních koster a nejkratších cest v ohodnocených grafech.

Druhá hypotéza, která předvídala vysokou úspěšnost modelu GNN u jednoduchých klasifikačních úloh a detekce souvislosti, byla rovněž potvrzena. Ukázalo se, že existence izolovaných komponent a základní strukturální rysy (např. úplnost grafu) jsou pro mechanismus message passing snadno rozpoznatelné. Právě v těchto oblastech se model GNN jeví jako vysoce efektivní nástroj pro rychlou heuristickou analýzu rozsáhlých dat.

Třetí hypotéza, zaměřená na limity GNN u složitějších problémů, byla potvrzena i s doplňujícím upřesněním. U rozpoznávání izomorfismu model narážel na teoretická omezení při identifikaci strukturálně identických dvojic s odlišným indexováním vrcholů, což odhalilo limity neuronových sítí při zachycování globální topologie bez dodatečných příznaků. V případě hamiltonovských kružnic sice model po optimalizaci dosáhl vysoké přesnosti, nicméně experimenty potvrdily, že úspěšnost zůstává silně závislá na kvalitě

trénovací sady a hloubce sítě, což u NP-úplných úloh stále nedosahuje spolehlivosti exaktních metod.

Přínos práce lze spatřit v metodickém propojení teoretických definic diskrétní matematiky zaměřených na teorii grafů s praktickou implementací v prostředí PyTorch Geometric a dalších použitých knihoven. Vytvořené skripty a sady optimalizovaných promptů tvoří základ prostředí, které může být použito při výuce teorie grafů. Přístup vizualizace může studentům napomoci k lepšímu porozumění abstraktním pojmům maticového zápisu a vizuální struktury grafu.

6. Summary

This bachelor's thesis addressed the use of artificial intelligence tools in the field of discrete mathematics, with a specific focus on graph theory. The main objective was to assess the extent to which these technologies can replace or complement classical algorithms, and to evaluate the potential these tools offer for teaching discrete mathematics.

The set sub-goals were gradually achieved during the course of the work. First, reliable reference datasets were compiled using the NetworkX library. This phase was essential for the subsequent objective validation of the model, as it provided precise mathematical certainty regarding the properties of the generated graphs. Next, classical graph algorithms, such as shortest path finding or minimum spanning tree construction, were implemented; their visualization enabled the comparison and understanding of the algorithm's procedures. The final sub-step was the design, training, and verification of a graph neural network, which demonstrated the ability to identify topological patterns without direct programming of logical rules.

The results of the experiments allowed us to summarize the findings and evaluate the validity of the established hypotheses. The first hypothesis, assuming that classical algorithms would exhibit behaviour consistent with theoretical assumptions, was fully confirmed. These methods remain an indispensable standard for tasks requiring absolute precision, particularly in the computation of minimum spanning trees and shortest paths in weighted graphs.

The second hypothesis, which predicted high performance of the GNN model in simple classification tasks and connection detection, was also confirmed. It turned out that the existence of isolated components and basic structural features (e.g., graph completeness) are easily recognizable by the message-passing mechanism. It is precisely in these areas that the GNN model appears to be a highly effective tool for rapid heuristic analysis of large-scale data.

The third hypothesis, focusing on the limitations of GNNs for more complex problems, was confirmed with additional refinements. In isomorphism recognition, the model encountered theoretical limitations when identifying structurally identical pairs with different vertex indexing, which revealed the limitations of neural networks in capturing

global topology without additional features. In the case of Hamiltonian cycles, although the model achieved high accuracy after optimization, experiments confirmed that success remains strongly dependent on the quality of the training set and the depth of the network, which for NP-complete tasks still does not reach the reliability of exact methods.

The contribution of this work lies in the methodological integration of theoretical definitions of discrete mathematics focused on graph theory with practical implementation in the PyTorch Geometric environment and other libraries used. The created scripts and sets of optimized prompts form the basis of an environment that can be used in teaching graph theory. The visualization approach can help students better understand the abstract concepts of matrix notation and the visual structure of a graph.

Keywords: graph theory, graph neural networks, discrete mathematics, graph algorithms, Python

7. Seznam použitých zdrojů

Barabási, A. -L., & Pósfai, M. (2016). *Network science*. Cambridge University Press.

Bondy, J. A., & Murty, U. S. R. (2008). *Graph theory*. Springer.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms* (Fourth edition). The MIT Press.

Demel, J. (2002). *Grafy a jejich aplikace*. Academia. https://kramerius.lib.cas.cz/uuid/uuid:80b38ce4-f825-4b3e-853d-1b3ed5244362

Fey, M., & Lenssen, J. E. (2019). Fast Graph Representation Learning with PyTorch Geometric. *ArXiv*, *2019*(1903.02428), 1-9. https://doi.org/10.48550/arXiv.1903.02428

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the Python in Science Conference* (pp. 11-15). SciPy. https://doi.org/10.25080/TCWV9851

Hamilton, W. L. (2020). *Graph representation learning*. Morgan & Claypool Publishers.

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, *9*(3), 90-95. https://doi.org/10.1109/MCSE.2007.55

Kingma, D. P., & Ba, J. (2017). Adam: A Method for Stochastic Optimization. *ArXiv*, *2017*(1412.6980), 15. https://doi.org/10.48550/arXiv.1412.6980

Matoušek, J., & Nešetřil, J. (2009). *Kapitoly z diskrétní matematiky* (4., upr. a dopl. vyd). Karolinum. http://toc.nkp.cz/NKC/201003/contents/nkc20092024910_1.pdf

Microsoft. (2025). *Visual Studio Code Documentation*. Visual Studio Code. Retrieved April 3, 2026, from https://code.visualstudio.com/docs

NumPy Developers. (2024). *NumPy Documentation*. NumPy.org. Retrieved April 4, 2026, from https://numpy.org/doc/

Project Jupyter. (2025). *Project Jupyter Documentation*. Jupyter.org. Retrieved April 3, 2026, from https://docs.jupyter.org/en/latest/index.html

PyG Team. (2025). *PyG Documentation*. Read the Docs. Retrieved April 4, 2026, from https://pytorch-geometric.readthedocs.io/en/latest/

Python Software Foundation. (2025). *Welcome to Python.org*. Python.org. Retrieved April 3, 2026, from https://www.python.org/

PyTorch Foundation. (2025). *PyTorch Documentation*. PyTorch. Retrieved April 4, 2026, from https://pytorch.org/

Pyvec. (2022). *Instalace a nastavení: VS Code*. Naucse.python.c. Retrieved April 3, 2026, from https://naucse.python.cz/2022/ostrava-python/beginners/vscode/

Pyvec. (2024). *Úvod do Pandas*. Naucse.python.cz. Retrieved April 4, 2026, from https://naucse.python.cz/lessons/intro/pandas/

Rosen, K. H. (2013). *Discrete mathematics and its applications* (Seventh edition). McGraw-Hill.

Russell, S. J., & Norvig, P. (2021). *Artificial intelligence: a modern approach* (Fourth edition). Pearson.

Šišma, P. (1998). Vznik a vývoj teorie grafů. *Pokroky matematiky, fyziky a astronomie*, *1998*(43), 89-99. http://hdl.handle.net/10338.dmlcz/137535

The pandas development team. (2025). *Pandas documentation*. Pandas.pydata.org. Retrieved April 4, 2026, from https://pydata.org

West, D. B. (2018). *Introduction to graph theory* (Second edition). Pearson.

8. Seznam obrázků

9. Seznam tabulek

10. Seznam příloh (jsou-li v práci přílohy)

11. Přílohy

Příloha 1: Kód experimentu E0

```
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt
from vykresleni_matice import vykresli_matici
from vykresleni_tabulky import vykresli_tabulku_vlastnosti

def analyzuj_vlastnosti(G, nazev):
    vlastnosti = {
        "Typ grafu": nazev,
        "Souvislý": "Ano" if nx.is_connected(G) else "Ne",
        "Komponenty": nx.number_connected_components(G),
        "Skóre": sorted([d for n, d in G.degree()], reverse=True)
    }

    if nx.is_connected(G):
        vlastnosti["Průměr (diam)"] = nx.diameter(G)
        vlastnosti["Poloměr (rad)"] = nx.radius(G)
        vlastnosti["Excentricita (v0)"] = nx.eccentricity(G, v=list(G.nodes())[0])
    else:
        vlastnosti["Průměr (diam)"] = "∞"
        vlastnosti["Poloměr (rad)"] = "N/A"
        vlastnosti["Excentricita (v0)"] = "N/A"

    return vlastnosti

test_grafy = [
    (nx.random_labeled_tree(8), "Strom"),
    (nx.cycle_graph(8), "Kružnice $C_8$"),
    (nx.complete_bipartite_graph(4, 2), "Bipartitní $K_{4,2}$"),
    (nx.disjoint_union(nx.cycle_graph(4), nx.path_graph(3)), "Nesouvislý graf")
]

vysledky = pd.DataFrame([analyzuj_vlastnosti(g, n) for g, n in test_grafy])
vykresli_tabulku_vlastnosti(vysledky)

for g, jmeno in test_grafy:
    adj = nx.to_numpy_array(g)
    inc = nx.incidence_matrix(g).toarray()
    dist = nx.floyd_warshall_numpy(g)

    fig, axes = plt.subplots(1, 4, figsize=(22, 5))
    fig.suptitle(f"Analýza grafu: {jmeno}", fontsize=16, fontweight='bold')

    if "Bipartitní" in jmeno:
        top = [n for n, d in g.nodes(data=True) if d.get('bipartite') == 0]
        pos = nx.bipartite_layout(g, top)
        nx.draw(g,  pos=pos,  ax=axes[0],  with_labels=True,  node_color='orange',
node_size=500)
    else:
```

```
        nx.draw(g,      ax=axes[0],      with_labels=True,      node_color='skyblue',
edge_color='gray', node_size=500) # Standardní rozložení pro ostatní grafy
    axes[0].set_title(f"Struktura grafu")

    vykresli_matici(adj, "Matice sousednosti $A$", axes[1])
    vykresli_matici(inc, "Matice incidence $I$", axes[2])
    vykresli_matici(dist, "Matice vzdáleností $D$", axes[3])

    plt.tight_layout()
    plt.subplots_adjust(top=0.80)
    plt.show()
```

Zdroj 43: Vlastní zpracování

Příloha 2: Pomocný kód vykresleni_matice

```
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt
from vykresleni_matice import vykresli_matici
from vykresleni_tabulky import vykresli_tabulku_vlastnosti

def analyzuj_vlastnosti(G, nazev):
    vlastnosti = {
        "Typ grafu": nazev,
        "Souvislý": "Ano" if nx.is_connected(G) else "Ne",
        "Komponenty": nx.number_connected_components(G),
        "Skóre": sorted([d for n, d in G.degree()], reverse=True)
    }

    if nx.is_connected(G):
        vlastnosti["Průměr (diam)"] = nx.diameter(G)
        vlastnosti["Poloměr (rad)"] = nx.radius(G)
        vlastnosti["Excentricita (v0)"] = nx.eccentricity(G, v=list(G.nodes())[0])
    else:
        vlastnosti["Průměr (diam)"] = "∞"
        vlastnosti["Poloměr (rad)"] = "N/A"
        vlastnosti["Excentricita (v0)"] = "N/A"

    return vlastnosti

test_grafy = [
    (nx.random_labeled_tree(8), "Strom"),
    (nx.cycle_graph(8), "Kružnice $C_8$"),
    (nx.complete_bipartite_graph(4, 2), "Bipartitní $K_{4,2}$"),
    (nx.disjoint_union(nx.cycle_graph(4), nx.path_graph(3)), "Nesouvislý graf")
]

vysledky = pd.DataFrame([analyzuj_vlastnosti(g, n) for g, n in test_grafy])
vykresli_tabulku_vlastnosti(vysledky)

for g, jmeno in test_grafy:
    adj = nx.to_numpy_array(g)
    inc = nx.incidence_matrix(g).toarray()
    dist = nx.floyd_warshall_numpy(g)

    fig, axes = plt.subplots(1, 4, figsize=(22, 5))
    fig.suptitle(f"Analýza grafu: {jmeno}", fontsize=16, fontweight='bold')

    if "Bipartitní" in jmeno:
        top = [n for n, d in g.nodes(data=True) if d.get('bipartite') == 0]
        pos = nx.bipartite_layout(g, top)
        nx.draw(g,  pos=pos,  ax=axes[0],  with_labels=True,  node_color='orange',
node_size=500)
    else:
        nx.draw(g,     ax=axes[0],     with_labels=True,     node_color='skyblue',
edge_color='gray', node_size=500) # Standardní rozložení pro ostatní grafy
    axes[0].set_title(f"Struktura grafu")
```

```
    vykresli_matici(adj, "Matice sousednosti $A$", axes[1])
    vykresli_matici(inc, "Matice incidence $I$", axes[2])
    vykresli_matici(dist, "Matice vzdáleností $D$", axes[3])

    plt.tight_layout()
    plt.subplots_adjust(top=0.80)
    plt.show()
```

Zdroj 44: Vlastní zpracování

Příloha 3: Pomocný kód vykresleni_tabulky

```
import matplotlib.pyplot as plt

def vykresli_tabulku_vlastnosti(df, titul="Vlastnosti grafů"):
    plot_df = df.copy()
    if 'Skóre' in plot_df.columns:
        plot_df['Skóre'] = plot_df['Skóre'].apply(lambda x: str(x))

    fig, ax = plt.subplots(figsize=(12, 4))
    ax.axis('off')

    tabulka = ax.table(
        cellText=plot_df.values,
        colLabels=plot_df.columns,
        cellLoc='center',
        loc='center'
    )

    tabulka.auto_set_font_size(False)
    tabulka.set_fontsize(11)
    tabulka.scale(1.2, 2.5)

    for (row, col), cell in tabulka.get_celld().items():
        if row == 0:
            cell.set_text_props(weight='bold', color='white')
            cell.set_facecolor('#2c3e50')
        elif row > 0:
            if row % 2 == 0:
                cell.set_facecolor('#f2f2f2')
            else:
                cell.set_facecolor('white')

    ax.set_title(titul, fontsize=16, fontweight='bold', pad=30)

    plt.tight_layout()
    plt.show()
    plt.show()
```

Zdroj 45: Vlastní zpracování

Příloha 4: Kód experimentu E1

```
import networkx as nx
import random
import pandas as pd
from vykresleni_ulohy import vykresli_ulohy_komplet, vykresli_tabulku

def spustit_experiment():
    n_uzlu = 7
    G = nx.erdos_renyi_graph(n=n_uzlu, p=0.5)

    while not nx.is_connected(G):
        G = nx.erdos_renyi_graph(n=n_uzlu, p=0.4)

    for (u, v) in G.edges():
        G[u][v]['weight'] = random.randint(5, 20)

    pos = nx.circular_layout(G)
    ulohy = []

    nodes = list(G.nodes())
    start_node, end_node = nodes[0], nodes[-1]

    path_bfs = nx.shortest_path(G, source=start_node, target=end_node)
    ulohy.append({
        'id': 'A', 'titul': 'Nejkratší cesta (BFS)', 'typ': 'bfs', 'G': G,
        'vysledek_text': f"{' -> '.join(map(str, path_bfs))} (Délka: {len(path_bfs)-
1} hran)",
        'cesta_hrany': list(zip(path_bfs, path_bfs[1:])),
        'barva_uzlu': 'lightgreen', 'start': start_node, 'end': end_node
    })

    path_dijkstra = nx.shortest_path(G, source=start_node, target=end_node,
weight='weight')
    total_weight = int(nx.path_weight(G, path_dijkstra, 'weight'))
    ulohy.append({
        'id': 'B', 'titul': 'Dijkstrův algoritmus', 'typ': 'vahy', 'G': G,
        'vysledek_text': f"{' -> '.join(map(str, path_dijkstra))} (Celková váha: {to-
tal_weight})",
        'cesta_hrany': list(zip(path_dijkstra, path_dijkstra[1:])),
        'barva_uzlu': 'lightblue', 'start': start_node, 'end': end_node
    })

    mst = nx.minimum_spanning_tree(G, algorithm='kruskal')

    edges_sorted = sorted(G.edges(data=True), key=lambda x: x[2]['weight'])
    postup_list = [f"Hrana ({u}-{v}), váha {d['weight']}: {'PŘIDÁNA' if
mst.has_edge(u, v) else 'VYNECHÁNA'}"
                   for u, v, d in edges_sorted]

    ulohy.append({
        'id': 'C', 'titul': 'Minimální kostra (MST)', 'typ': 'kostra', 'G': G,
        'vysledek_text': "\n".join(postup_list),
        'kostra_hrany': list(mst.edges()), 'barva_uzlu': 'orange'
    })
```

```
    coloring = nx.coloring.greedy_color(G, strategy='largest_first')
    chromatic_number = max(coloring.values()) + 1
    ulohy.append({
        'id': 'D', 'titul': 'Barvení grafu', 'typ': 'barvy', 'G': G,
        'vysledek_text': f"Chromatické číslo: {chromatic_number}",
        'barva_uzlu': [coloring[n] for n in G.nodes()]
    })

    vykresli_tabulku(ulohy)
    vykresli_ulohy_komplet(ulohy, pos)

if __name__ == "__main__":
    spustit_experiment()
```

Zdroj 46: Vlastní zpracování

Příloha 5: Pomocný kód vykresleni_ulohy

```
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd

def vykresli_ulohy_komplet(ulohy_data, pos):
    """Vykreslí grafy (váhy jen u Dijkstry a Kruskala)."""
    fig, axes = plt.subplots(2, 4, figsize=(20, 10))
    fig.suptitle("VIZUALIZACE GRAFOVÝCH ÚLOH", fontsize=16, fontweight='bold')

    for i, u in enumerate(ulohy_data):
        G = u['G']
         nx.draw(G, pos, ax=axes[0, i], with_labels=True, node_color='white', edge-
colors='black', node_size=700)
        if u['typ'] in ['vahy', 'kostra']:
               nx.draw_networkx_edge_labels(G, pos, edge_labels=nx.get_edge_attri-
butes(G, 'weight'), ax=axes[0, i])
        axes[0, i].set_title(f"ZADÁNÍ {u['id']}", fontweight='bold')

        nx.draw(G, pos, ax=axes[1, i], with_labels=True, node_color=u.get('barva_uz-
lu', 'lightgray'), node_size=700)
        if u['typ'] in ['vahy', 'kostra']:
               nx.draw_networkx_edge_labels(G, pos, edge_labels=nx.get_edge_attri-
butes(G, 'weight'), ax=axes[1, i])

        if 'cesta_hrany' in u:
                      nx.draw_networkx_edges(G, pos, edgelist=u['cesta_hrany'],
edge_color='red', width=5, ax=axes[1, i])
        if 'kostra_hrany' in u:
                     nx.draw_networkx_edges(G, pos, edgelist=u['kostra_hrany'],
edge_color='blue', width=5, ax=axes[1, i])
        axes[1, i].set_title(f"ŘEŠENÍ {u['id']}", fontweight='bold')

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()

def vytvor_tabulku(ax, df, titul):
    ax.axis('off')
    ax.set_title(titul, fontsize=12, fontweight='bold', loc='left',y=1.05)

    tab = ax.table(cellText=df.values,
                   colLabels=df.columns,
                   cellLoc='left',
                   loc='center',
                   colColours=["#f2f2f2"] * len(df.columns))

    tab.auto_set_font_size(False)
    tab.set_fontsize(10)
    tab.scale(1, 1.8)
    return tab

def vykresli_tabulku(ulohy_data):
    zadani_data = []
    for u in ulohy_data:
```

```
        text = f"Najděte nejkratší cestu z {u['start']} do {u['end']}" if 'start' in
u else ("Najděte min. kostru" if u['typ']=='kostra' else "Určete chrom. číslo")
        zadani_data.append([u['id'], u['titul'], text])
    df_zadani = pd.DataFrame(zadani_data, columns=["ID", "Algoritmus", "Instrukce k
úloze"])

    vysledky_data = []
    kruskal_rows = []
    for u in ulohy_data:
        vysl = u['vysledek_text'].split('\n')[0]
        if u['typ'] == 'kostra':
            vaha_mst = sum(d['weight'] for u1, v1, d in u['G'].edges(data=True) if
(u1, v1) in u['kostra_hrany'] or (v1, u1) in u['kostra_hrany'])
            vysl = f"Váha kostry: {vaha_mst}"
            for line in u['vysledek_text'].split('\n'):
               kruskal_rows.append([line.split(':')[0], line.split(':')[1].strip()])
        vysledky_data.append([u['id'], u['titul'], vysl])
    df_vysledky = pd.DataFrame(vysledky_data, columns=["ID", "Algoritmus", "Klíč ře-
šení"])
    df_kruskal = pd.DataFrame(kruskal_rows, columns=["Hrana (Váha)", "Rozhodnutí al-
goritmu"])

    fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(12, 16))

    vytvor_tabulku(ax1, df_zadani, "1. MATICE ZADÁNÍ ÚLOH")
    vytvor_tabulku(ax2, df_vysledky, "2. MATICE SOUHRNNÝCH VÝSLEDKŮ")
    vytvor_tabulku(ax3, df_kruskal, "3. MATICE PRŮBĚHU KRUSKALOVA ALGORITMU")

    plt.subplots_adjust(hspace=0.4, top=0.92, bottom=0.05)

    plt.show()
```

Zdroj 47: Vlastní zpracován

Příloha 6: Kód experimentu E2

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import networkx as nx
import random
import matplotlib.pyplot as plt
from torch_geometric.data import Data
from torch_geometric.loader import DataLoader
from torch_geometric.nn import GCNConv, global_mean_pool
from datove_rozhrani import (vypis_statistiky, vytvor_tabulku_epoch, vyhodnot_mo-
del_pandas, vyhodnot_iso_pandas,
                         univerzalni_vykresleni, vykresli_dvojice_isomorfismu)class
model_GNN(nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(model_GNN, self).__init__()
        torch.manual_seed(42)
        self.conv1 = GCNConv(in_channels, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, hidden_channels)
        self.conv3 = GCNConv(hidden_channels, hidden_channels)
        self.lin = nn.Linear(hidden_channels, out_channels)

    def forward(self, x, edge_index, batch):
        x = self.conv1(x, edge_index).relu()
        x = self.conv2(x, edge_index).relu()
        x = self.conv3(x, edge_index).relu()
        x = global_mean_pool(x, batch)
        x = F.dropout(x, p=0.2, training=self.training)
        return self.lin(x)

def nx_to_pyg(G, label):
    edges = list(G.edges())
    if len(edges) == 0:
        edge_index = torch.empty((2, 0), dtype=torch.long)
    else:
        edge_index = torch.tensor(edges, dtype=torch.long).t().contiguous()
        edge_index = torch.cat([edge_index, edge_index[[1, 0]]], dim=1)
    x = torch.tensor([[deg] for _, deg in G.degree()], dtype=torch.float)
    y = torch.tensor([label], dtype=torch.long)
    return Data(x=x, edge_index=edge_index, y=y)

def vytvor_univerzalni_dataset(konfigurace, pocet_na_typ=100):
    dataset = []
    for nx_funkce, label in konfigurace:
        for _ in range(pocet_na_typ):
            n = random.randint(6, 12)
            dataset.append(nx_to_pyg(nx_funkce(n), label))
    random.shuffle(dataset)
    return dataset

label_to_name_typy = {0: "Strom", 1: "Cyklus", 2: "Bipartitní", 3: "Úplný", 4:
"Cesta"}
```

```
konfigurace = [(nx.random_labeled_tree, 0), (nx.cycle_graph, 1),
                (lambda n: nx.complete_bipartite_graph(n//2, n-n//2), 2),
                (nx.complete_graph, 3), (nx.path_graph, 4)]

dataset_typy = vytvor_univerzalni_dataset(konfigurace, pocet_na_typ=150)
train_loader_typy = DataLoader(dataset_typy[:600], batch_size=32, shuffle=True)
history_typy = []

model_typy = model_GNN(1, 32, 5)
optimizer = torch.optim.Adam(model_typy.parameters(), lr=0.01)
criterion = nn.CrossEntropyLoss()

for epoch in range(1, 31):
    model_typy.train()
    loss_sum, correct, total = 0, 0, 0
    for batch in train_loader_typy:
        optimizer.zero_grad()
        out = model_typy(batch.x, batch.edge_index, batch.batch)
        loss = criterion(out, batch.y)
        loss.backward(); optimizer.step()
        loss_sum += loss.item()
        correct += (out.argmax(dim=1) == batch.y).sum().item()
        total += batch.y.size(0)
    if epoch % 5 == 0:
        history_typy.append({"Epocha": epoch, "Ztráta": round(loss_sum/len(train_lo-
ader_typy), 4), "Acc %": round(100*correct/total, 2)})

display(vytvor_tabulku_epoch(history_typy))
display(vyhodnot_model_pandas(model_typy, train_loader_typy, la-bel_to_name_typy))
univerzalni_vykresleni(model_typy, dataset_typy[600:], label_to_name_typy, pocet=10)

def generuj_data_souvislost(pocet=400):
    dataset = []
    for _ in range(pocet // 2):
        n = random.randint(6, 12)
        dataset.append(nx_to_pyg(nx.random_labeled_tree(n), 1))
        G1, G2 = nx.complete_graph(n//2), nx.complete_graph(n-n//2)
        dataset.append(nx_to_pyg(nx.disjoint_union(G1, G2), 0))
    random.shuffle(dataset)
    return dataset

label_to_name_conn = {0: "Nesouvislý", 1: "Souvislý"}
dataset_conn = generuj_data_souvislost(400)
train_loader_conn = DataLoader(dataset_conn[:300], batch_size=32, shuffle=True)
history_conn = []

model_conn = model_GNN(1, 32, 2)
optimizer = torch.optim.Adam(model_conn.parameters(), lr=0.01)
criterion = nn.CrossEntropyLoss()

for epoch in range(1, 31):
    model_conn.train()
    loss_sum, correct, total = 0, 0, 0
    for batch in train_loader_conn:
        optimizer.zero_grad()
```

```
        out = model_conn(batch.x, batch.edge_index, batch.batch)
        loss = criterion(out, batch.y)
        loss.backward(); optimizer.step()
        loss_sum += loss.item()
        correct += (out.argmax(dim=1) == batch.y).sum().item()
        total += batch.y.size(0)
    if epoch % 5 == 0:
        history_conn.append({"Epocha": epoch, "Ztráta": round(loss_sum/len(train_lo-
ader_conn), 4), "Acc %": round(100*correct/total, 2)})

display(vytvor_tabulku_epoch(history_conn))
display(vyhodnot_model_pandas(model_conn, train_loader_conn, la-bel_to_name_conn))
univerzalni_vykresleni(model_conn, dataset_conn[300:], label_to_name_conn, pocet=10)

def nx_to_pyg_iso(G, label):
    x = torch.tensor([[deg] for _, deg in G.degree()], dtype=torch.float)
    edge_index = torch.tensor(list(G.edges)).t().contiguous()
    if edge_index.numel() > 0:
        edge_index = torch.cat([edge_index, edge_index[[1, 0]]], dim=1)
    return Data(x=x, edge_index=edge_index, y=torch.tensor([label]))

def generuj_isomorfismus_v2(pocet=400):
    dataset = []
    typy = [nx.random_labeled_tree, nx.cycle_graph, nx.path_graph, nx.complete_graph]

    for _ in range(pocet // 2):
        n = random.randint(6, 9)
        vybrany_typ = random.choice(typy)
        G1 = vybrany_typ(n)
        mapping = list(range(len(G1)))
        random.shuffle(mapping)
        G2 = nx.relabel_nodes(G1, {i: mapping[i] for i in range(len(G1))})
        dataset.append((nx_to_pyg_iso(G1, 1), nx_to_pyg_iso(G2, 1), 1))

        t1, t2 = random.sample(typy, 2)
        G3 = t1(n)
        G4 = t2(n)
        dataset.append((nx_to_pyg_iso(G3, 0), nx_to_pyg_iso(G4, 0), 0))

    random.shuffle(dataset)
    return dataset


vlak_iso = generuj_isomorfismus_v2(400)
history_iso = []
model_iso = model_GNN(1, 32, 2)
optimizer = torch.optim.Adam(model_iso.parameters(), lr=0.001)

for epoch in range(1, 31):
    model_iso.train()
    loss_sum, correct = 0, 0
    for i in range(0, len(vlak_iso), 10):
        batch = vlak_iso[i:i+10]
        optimizer.zero_grad()
        batch_l = 0
```

```
        for g1, g2, label in batch:
                emb1 = model_iso(g1.x, g1.edge_index, torch.zeros(g1.x.size(0),
dtype=torch.long))
                emb2 = model_iso(g2.x, g2.edge_index, torch.zeros(g2.x.size(0),
dtype=torch.long))
            dist = torch.norm(emb1 - emb2, p=2)
            loss = dist if label == 1 else F.relu(1.0 - dist)
            loss.backward(); batch_l += loss.item()
            if (label == 1 and dist < 0.5) or (label == 0 and dist >= 0.5): correct
+= 1
        optimizer.step(); loss_sum += batch_l

    if epoch % 5 == 0:
        history_iso.append({"Epocha": epoch, "Ztráta": round(loss_sum/400, 4), "Acc
%": round(100*correct/400, 2)})

display(vytvor_tabulku_epoch(history_iso))
test_iso = vlak_iso[300:]
df_iso_vysledky = vyhodnot_iso_pandas(model_iso, test_iso)
display(df_iso_vysledky)
vykresli_dvojice_isomorfismu(model_iso, test_iso, pocet=5)

def generuj_data_hamilton(pocet=400):
    dataset = []
    for _ in range(pocet // 2):
        n = random.randint(8, 12)
                dataset.append(nx_to_pyg(random.choice([nx.cycle_graph(n),   nx.com-
plete_graph(n)]), 1))
                 dataset.append(nx_to_pyg(random.choice([nx.random_labeled_tree(n),
nx.path_graph(n)]), 0))
    random.shuffle(dataset)
    return dataset

label_to_name_ham = {0: "Bez Hamiltona", 1: "Má Hamiltona"}
dataset_ham = generuj_data_hamilton(400)
train_loader_ham = DataLoader(dataset_ham[:320], batch_size=32, shuffle=True)
history_ham = []

model_ham = model_GNN(1, 32, 2)
optimizer = torch.optim.Adam(model_ham.parameters(), lr=0.01)
criterion = nn.CrossEntropyLoss()

for epoch in range(1, 31):
    model_ham.train()
    loss_sum, correct, total = 0, 0, 0
    for batch in train_loader_ham:
        optimizer.zero_grad()
        out = model_ham(batch.x, batch.edge_index, batch.batch)
        loss = criterion(out, batch.y)
        loss.backward(); optimizer.step()
        loss_sum += loss.item()
        correct += (out.argmax(dim=1) == batch.y).sum().item()
        total += batch.y.size(0)
    if epoch % 5 == 0:
```

```
        history_ham.append({"Epocha": epoch, "Ztráta": round(loss_sum/len(train_loa-
der_ham), 4), "Acc %": round(100*correct/total, 2)})

display(vytvor_tabulku_epoch(history_ham))
display(vyhodnot_model_pandas(model_ham, train_loader_ham, label_to_name_ham))
univerzalni_vykresleni(model_ham, dataset_ham[320:], label_to_name_ham, po-cet=10)
```

Zdroj 48: Vlastní zpracování

Příloha 7: Pomocný kód datove_rozhrani

```
import torch
import torch.nn.functional as F
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd
import random
from torch_geometric.utils import to_networkx

def vypis_statistiky(dataset, label_mapping):
    from collections import Counter
    labels = [d.y.item() for d in dataset]
    cetnost = Counter(labels)

    data_pro_df = []
    for label_id, count in cetnost.items():
        data_pro_df.append({
            'Kategorie': label_mapping.get(label_id, f"ID {label_id}"),
            'Počet vzorků': count
        })

    return pd.DataFrame(data_pro_df)

def vytvor_tabulku_epoch(historie):
    if not historie:
        return "Historie je prázdná."
    return pd.DataFrame(historie)

def vyhodnot_model_pandas(model, loader, label_mapping):
    model.eval()
    stats = {name: {'correct': 0, 'total': 0} for name in label_mapping.values()}

    with torch.no_grad():
        for batch in loader:
            out = model(batch.x, batch.edge_index, batch.batch)
            pred = out.argmax(dim=1)
            for i in range(len(batch.y)):
                l_name = label_mapping[batch.y[i].item()]
                stats[l_name]['total'] += 1
                if pred[i] == batch.y[i]:
                    stats[l_name]['correct'] += 1

    data_vysledky = []
    for kat, s in stats.items():
        acc = (s['correct'] / s['total'] * 100) if s['total'] > 0 else 0
        data_vysledky.append({'Kategorie': kat, 'Úspěšnost (%)': round(acc, 2)})

    return pd.DataFrame(data_vysledky)

def vyhodnot_iso_pandas(model_objekt, test_data):
    model_objekt.eval()
```

```
    statistiky_iso = {"Stejné": [0, 0], "Různé": [0, 0]}

    with torch.no_grad():
        for g1, g2, label in test_data:
               emb1 = model_objekt(g1.x, g1.edge_index, torch.zeros(g1.x.size(0),
dtype=torch.long))
               emb2 = model_objekt(g2.x, g2.edge_index, torch.zeros(g2.x.size(0),
dtype=torch.long))

            sim = F.cosine_similarity(emb1, emb2).item()
            pred = 1 if sim > 0.5 else 0
            kat = "Stejné" if label == 1 else "Různé"

            if pred == label:
                statistiky_iso[kat][0] += 1
            statistiky_iso[kat][1] += 1

    data_iso = []
    for kat, counts in statistiky_iso.items():
        acc = (counts[0] / counts[1] * 100) if counts[1] > 0 else 0
        data_iso.append({'Kategorie': kat, 'Úspěšnost (%)': round(acc, 2)})

    return pd.DataFrame(data_iso)

def univerzalni_vykresleni(model, test_dataset, label_mapping, pocet=10):
    model.eval()
    fig, axes = plt.subplots(2, 5, figsize=(20, 8))
    axes = axes.flatten()

    vzorky = random.sample(test_dataset, min(len(test_dataset), pocet))

    for i, data in enumerate(vzorky):
        with torch.no_grad():
                out = model(data.x, data.edge_index, torch.zeros(data.x.size(0),
dtype=torch.long))
            pred = out.argmax(dim=1).item()

        skutecnost = data.y.item()
        G = to_networkx(data, to_undirected=True)
        trefa = (pred == skutecnost)

        if nx.is_bipartite(G) and (skutecnost == 2 or pred == 2): # 2 je index pro
Bipartitní
            strany = nx.bipartite.sets(G)
            pos = nx.bipartite_layout(G, list(strany[0]))
        else:
            pos = nx.kamada_kawai_layout(G)

        plt.sca(axes[i])
        nx.draw(G, pos, with_labels=True,
                node_color=('lightgreen' if trefa else 'salmon'),
                edge_color='gray', node_size=400)

        axes[i].set_title(f"AI: {label_mapping[pred]}\nReal: {label_mapping[skutec-
nost]}",
```

```
                            color=('green' if trefa else 'red'), fontweight='bold')
    plt.tight_layout()
    plt.show()

def vykresli_dvojice_isomorfismu(model_iso, dataset_iso, pocet=5):
    model_iso.eval()
    fig, axes = plt.subplots(2, pocet, figsize=(4 * pocet, 8))

    vzorky = random.sample(dataset_iso, min(len(dataset_iso), pocet))

    for i, (g1_data, g2_data, label) in enumerate(vzorky):
        with torch.no_grad():
                    emb1  =  model_iso(g1_data.x,  g1_data.edge_index,  torch.ze-
ros(g1_data.x.size(0), dtype=torch.long))
                    emb2  =  model_iso(g2_data.x,  g2_data.edge_index,  torch.ze-
ros(g2_data.x.size(0), dtype=torch.long))
            dist = torch.norm(emb1 - emb2, p=2).item()
            pred_id = 1 if dist < 0.5 else 0

        trefa = (pred_id == label)

        for row, g_data in enumerate([g1_data, g2_data]):
            ax = axes[row][i]
            plt.sca(ax)
            G = to_networkx(g_data, to_undirected=True)

            if nx.is_bipartite(G):
                strany = nx.bipartite.sets(G)
                pos = nx.bipartite_layout(G, list(strany[0]))
            else:
                pos = nx.kamada_kawai_layout(G)

            nx.draw(G, pos, with_labels=True,
                    node_color=('lightgreen' if trefa else 'salmon'), node_size=300)

            if row == 0:
                  ax.set_title(f"Pár {i+1}: {'SPRÁVNĚ' if trefa else 'CHYBA'}\nAI:
{'Stejné' if pred_id==1 else 'Různé'}",
                               color=('green' if trefa else 'red'), fontweight='bold')
    plt.tight_layout()
    plt.show()
```

Zdroj 49: Vlastní zpracování

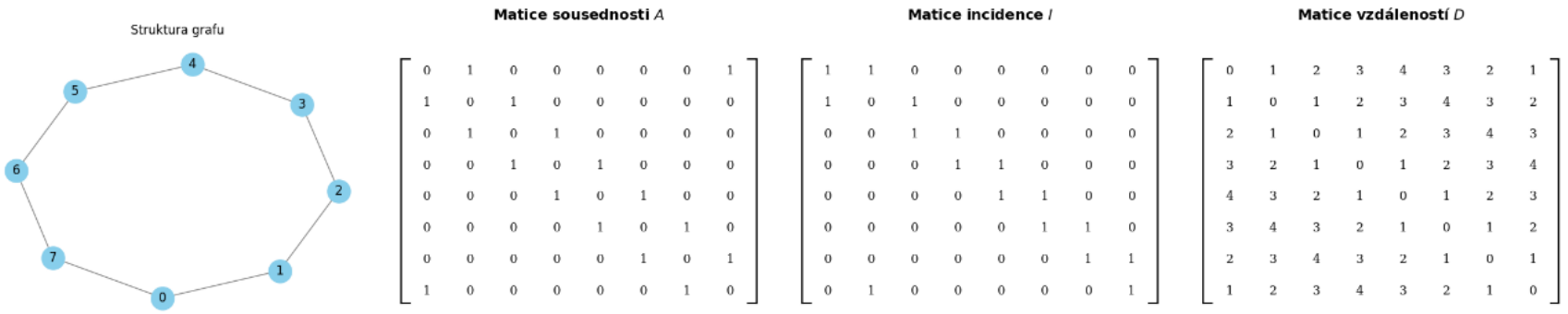
Příloha 8: Tabulka vlastností z experimentu E0

Vlastnosti grafů

Typ grafu	Souvislý	Komponenty	Skóre	Průměr (diam)	Poloměr (rad)	Excentricita (v0)
Strom	Ano	1	[3, 2, 2, 2, 2, 1, 1, 1]	5	3	5
Kružnice C_8	Ano	1	[2, 2, 2, 2, 2, 2, 2, 2]	4	4	4
Bipartitní $K_{4,2}$	Ano	1	[4, 4, 2, 2, 2, 2]	2	2	2
Nesouvislý graf	Ne	2	[2, 2, 2, 2, 2, 1, 1]	∞	N/A	N/A

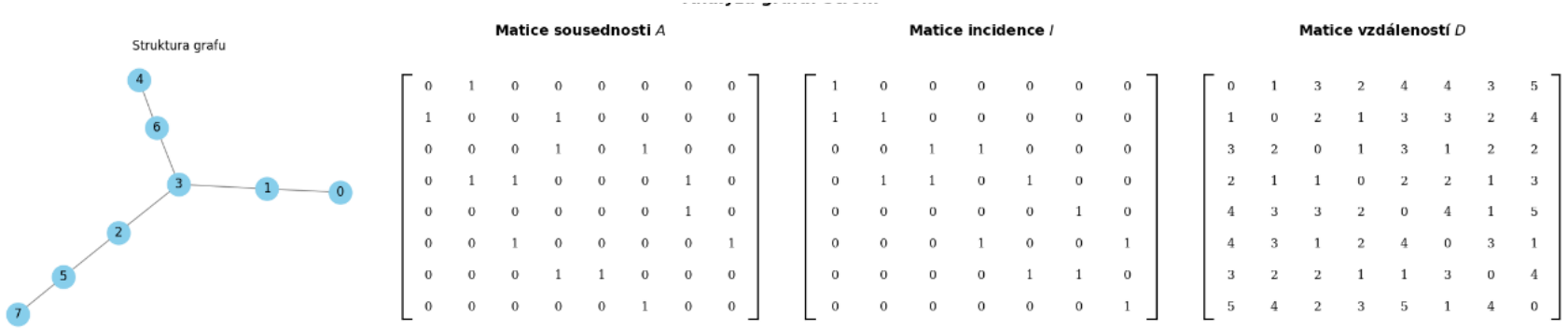
Zdroj 50: Vlastní zpracování

Příloha 10: Vygenerovaný graf typ kružnice z experimentu E0



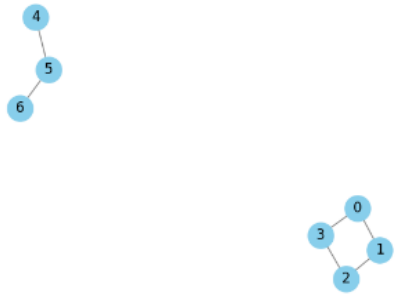
Zdroj 52: Vlastní zpracování

Příloha 9: Vygenerovaný graf typu strom z experimentu E0



Zdroj 51: Vlastní zpracování

Příloha 12: Vygenerovaný graf typu nesouvislý graf z experimentu E0



Matice sousednosti *A*

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Matice incidence *I*

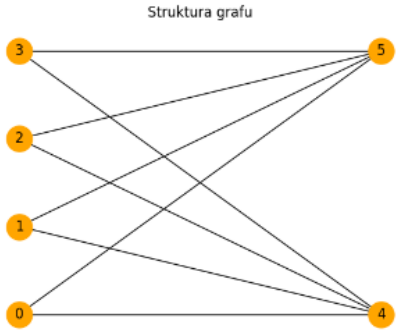
$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Matice vzdáleností *D*

$$\begin{bmatrix} 0 & 1 & 2 & 1 & \infty & \infty & \infty \\ 1 & 0 & 1 & 2 & \infty & \infty & \infty \\ 2 & 1 & 0 & 1 & \infty & \infty & \infty \\ 1 & 2 & 1 & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & 1 & 2 \\ \infty & \infty & \infty & \infty & 1 & 0 & 1 \\ \infty & \infty & \infty & \infty & 2 & 1 & 0 \end{bmatrix}$$

Zdroj 54: Vlastní zpracování

Příloha 11: Vygenerovaný graf typu bipartitní graf z experimentu E0



Matice sousednosti *A*

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Matice incidence *I*

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Matice vzdáleností *D*

$$\begin{bmatrix} 0 & 2 & 2 & 2 & 1 & 1 \\ 2 & 0 & 2 & 2 & 1 & 1 \\ 2 & 2 & 0 & 2 & 1 & 1 \\ 2 & 2 & 2 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 2 \\ 1 & 1 & 1 & 1 & 2 & 0 \end{bmatrix}$$

Zdroj 53: Vlastní zpracování

Příloha 13: Vygenerované tabulky k experimentu E1

1. MATICE ZADÁNÍ ÚLOH

ID	Algoritmus	Instrukce k úloze
A	Nejkratší cesta (BFS)	Najděte nejkratší cestu z 0 do 6
B	Dijkstrův algoritmus	Najděte nejkratší cestu z 0 do 6
C	Minimální kostra (MST)	Najděte min. kostru
D	Barvení grafu	Určete chrom. číslo

2. MATICE SOUHRNNÝCH VÝSLEDKŮ

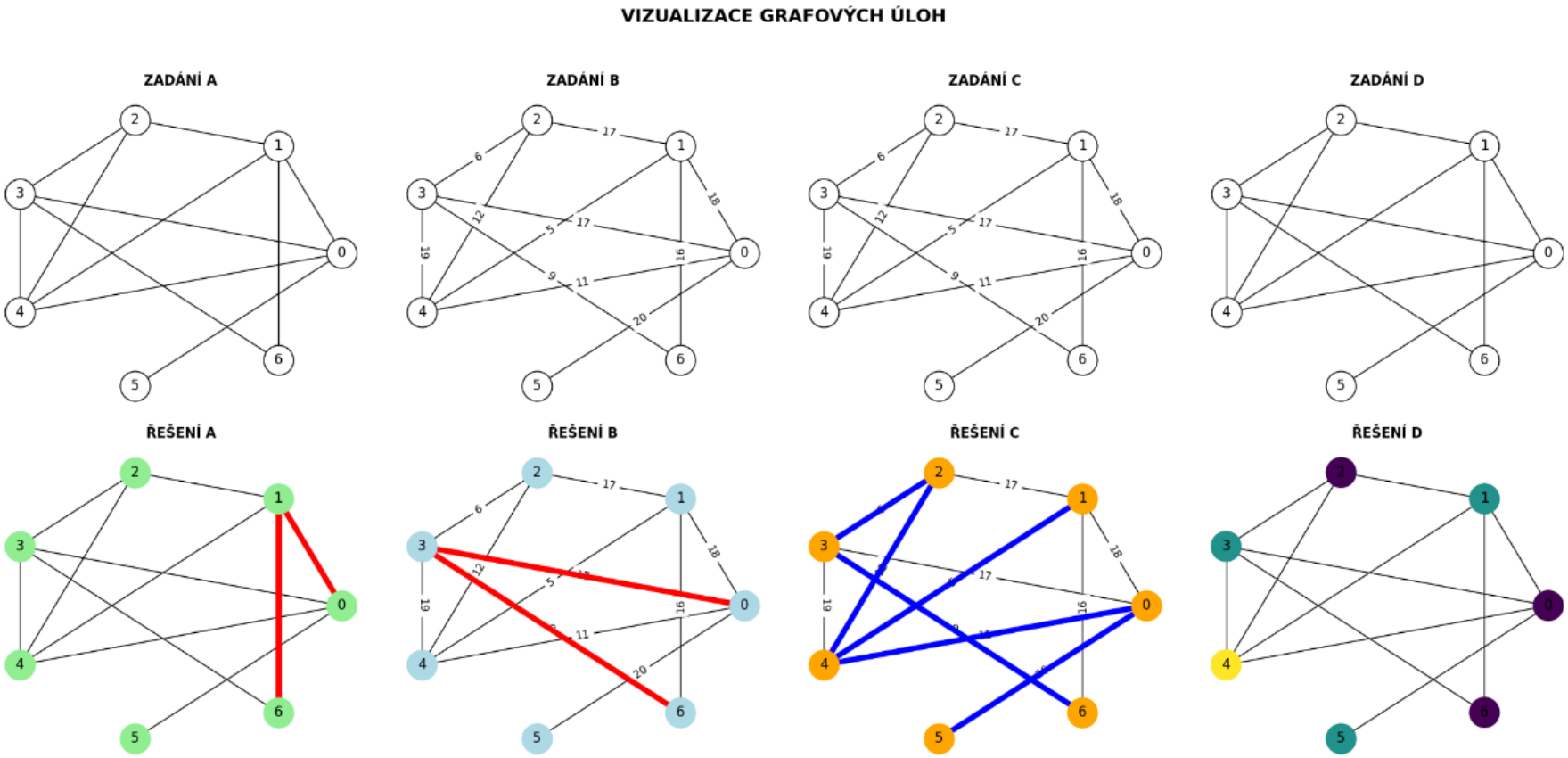
ID	Algoritmus	Klíč řešení
A	Nejkratší cesta (BFS)	0 -> 1 -> 6 (Délka: 2 hran)
B	Dijkstrův algoritmus	0 -> 3 -> 6 (Celková váha: 26)
C	Minimální kostra (MST)	Váha kostry: 63
D	Barvení grafu	Chromatické číslo: 3

3. MATICE PRŮBĚHU KRUSKALOVA ALGORITMU

Hrana (Váha)	Rozhodnutí algoritmu
Hrana (1-4), váha 5	PŘIDÁNA
Hrana (2-3), váha 6	PŘIDÁNA
Hrana (3-6), váha 9	PŘIDÁNA
Hrana (0-4), váha 11	PŘIDÁNA
Hrana (2-4), váha 12	PŘIDÁNA
Hrana (1-6), váha 16	VYNECHÁNA
Hrana (0-3), váha 17	VYNECHÁNA
Hrana (1-2), váha 17	VYNECHÁNA
Hrana (0-1), váha 18	VYNECHÁNA
Hrana (3-4), váha 19	VYNECHÁNA
Hrana (0-5), váha 20	PŘIDÁNA

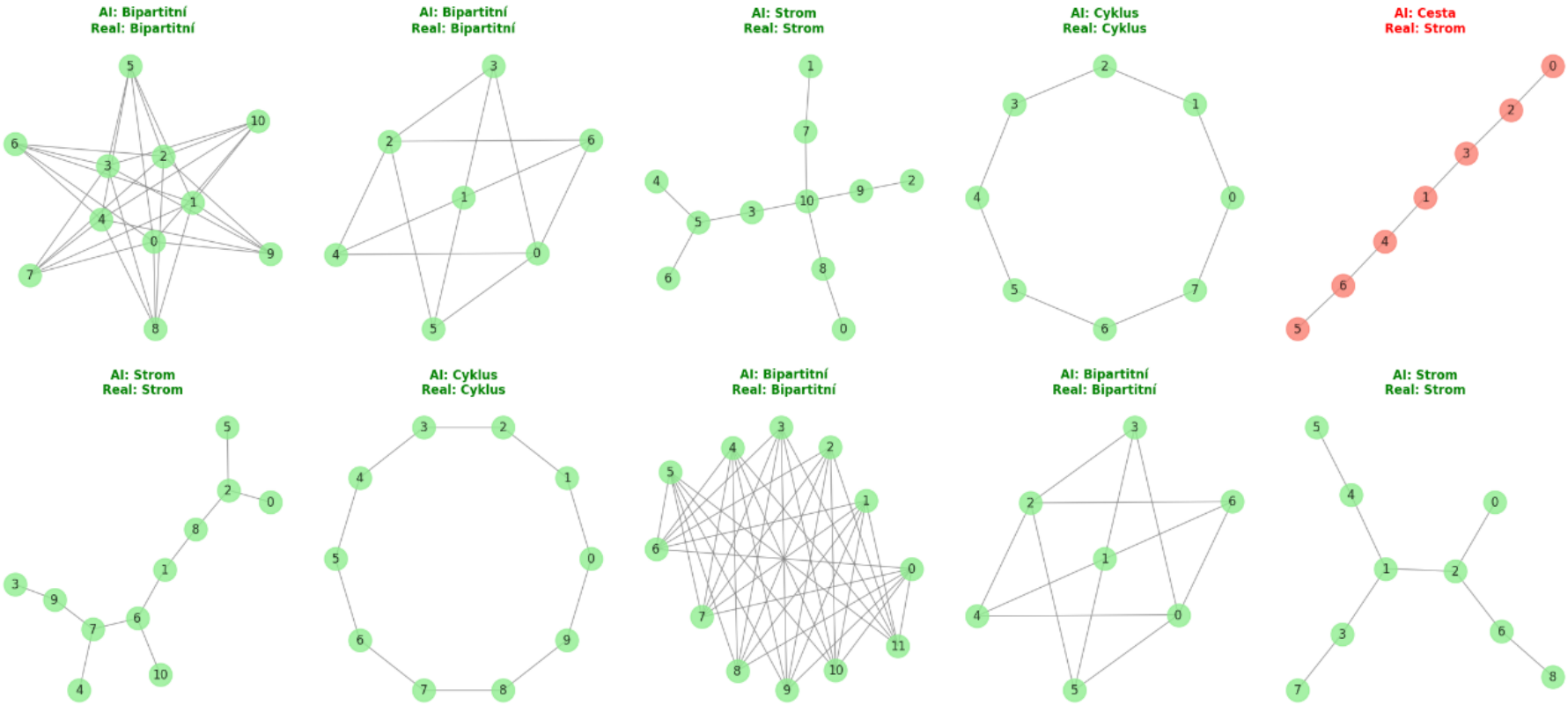
Zdroj 55: Vlastní zpracování

Příloha 14: Vygenerované grafy k experimentu E1



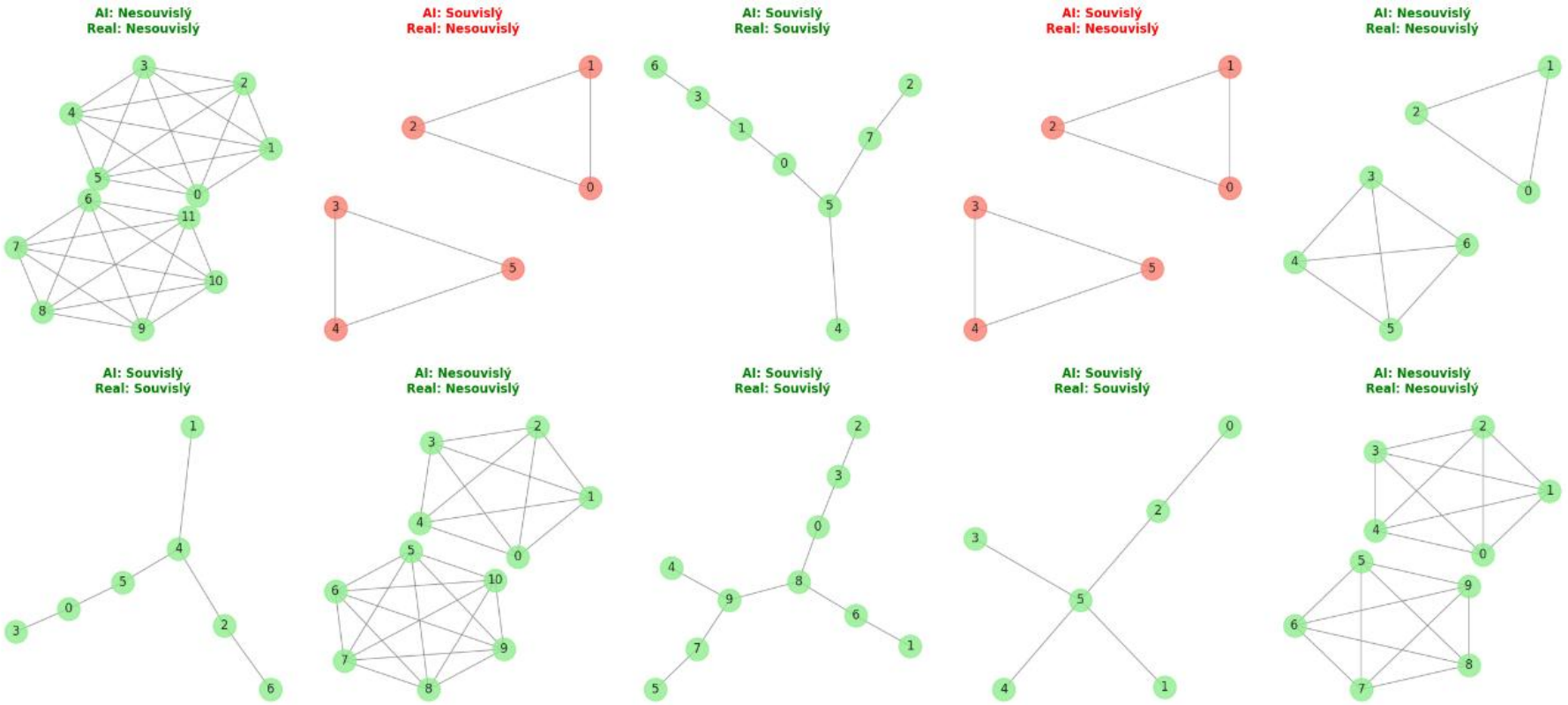
Zdroj 56: Vlastní zpracování

Příloha 15: Vygenerované grafy úlohy Klasifikace typů grafů k experimentu E2



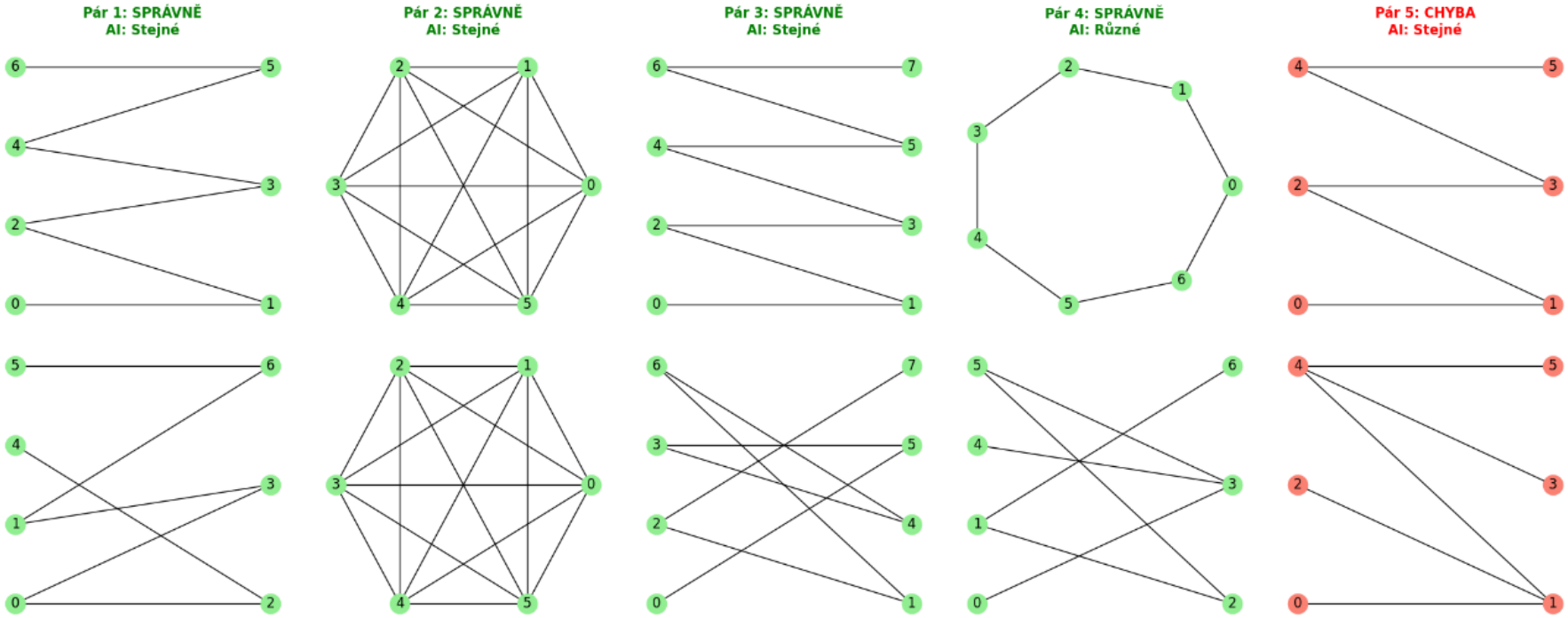
Zdroj 57: Vlastní zpracování

Příloha 16: Vygenerované grafy úlohy Detekce souvislosti k experimentu E2



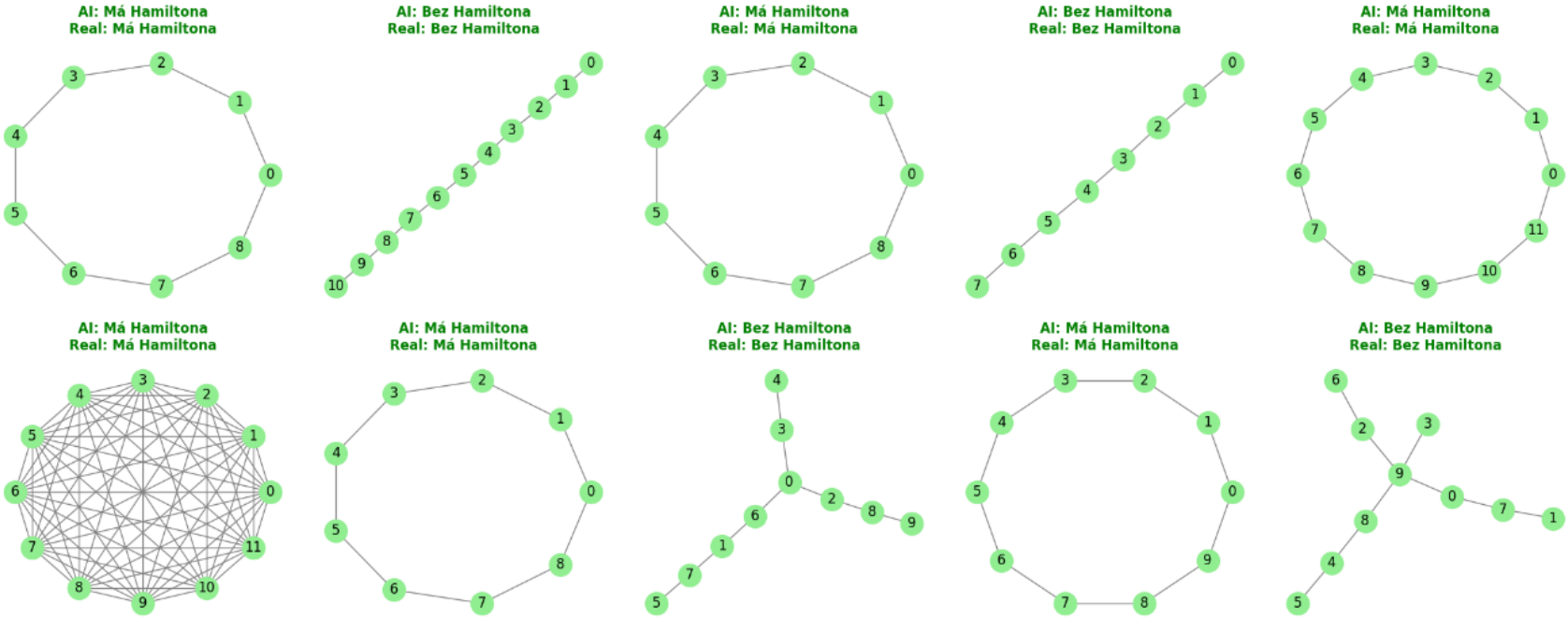
Zdroj 58: Vlastní zpracování

Příloha 17: Vygenerované grafy úlohy Rozpoznávání izomorfismu k experimentu E2



Zdroj 59: Vlastní zpracování

Příloha 18: *Vygenerované grafy úlohy Hamiltonovská kružnice k experimentu E2*



Zdroj 60: Vlastní zpracování