Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Problém nejdelší cesty a jeho aplikace v železniční dopravě

Bc. Matěj Pátek

Diplomová práce
2025

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2024/2025

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Matěj Pátek**
Osobní číslo:	**I23280**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Problém nejdelší cesty a její aplikace v železniční dopravě**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Práce se bude zabývat problémem nalezení jednoduché cesty maximální délky v daném grafu. Cesta se nazývá jednoduchá, pokud nemá žádné opakované vrcholy; Délka cesty může být buď měřena jejím počtem hran, nebo (ve vážených grafech) součtem vah jejích hran. Na rozdíl od problému s nejkratší cestou, kterou lze vyřešit v polynomiálním čase v grafech bez cyklů se zápornou váhou, je problém s nejdelší cestou NP-těžký. Tento problém lze aplikovat např v dopravních problémech.
Cílem teoretické části bude rešerše metod pro nalezení nejdelší cesty v grafu a použitelnosti tohoto problému v dopravních úlohách.
Cílem praktické části bude vytvoření aplikace pro nalezení nejdelší cesty v grafu na dopravním problému a visualizace výsledků

Rozsah pracovní zprávy:
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

HOCHBAUM, Dorit S. Approximation algorithms for NP-hard problems. Boston: PWS Pub. Co., c1997. ISBN 978-0534949686

Vedoucí diplomové práce: **RNDr. Josef Rak, Ph.D.**
Katedra automatizace a matematiky

Datum zadání diplomové práce: **31. října 2024**
Termín odevzdání diplomové práce: **23. května 2025**

L.S.

prof. Ing. Petr Doležel, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 29. listopadu 2024

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 20. 5. 2025

Bc. Matěj Pátek

PODĚKOVÁNÍ

Na tomto místě bych rád vyjádřil své upřímné poděkování panu RNDr. Josefu Rakovi, Ph.D., za jeho odborné vedení, trpělivost a cenné rady, které mi byly v průběhu zpracování této diplomové práce velkou pomocí.

Mé poděkování patří také mé rodině, která mě po celou dobu studia podporovala, motivovala a vytvářela mi zázemí potřebné pro soustředěnou práci. Bez jejich pochopení a podpory by bylo dosažení tohoto cíle podstatně náročnější.

ANOTACE

Práce se bude zabývat problémem nalezení jednoduché cesty maximální délky v daném grafu. Cesta se nazývá jednoduchá, pokud nemá žádné opakované vrcholy. Délka cesty může být buď měřena jejím počtem hran, nebo (ve vážených grafech) součtem vah jejích hran. Na rozdíl od problému s nejkratší cestou, kterou lze vyřešit v polynomiálním čase v grafech bez cyklů se zápornou váhou, je problém s nejdelší cestou NP-těžký. Tento problém lze aplikovat např. v dopravních problémech.

KLÍČOVÁ SLOVA

Graf, vrchol, hrana, nejdelší cesta, NP-těžký, heuristika, Java

TITLE

The longest path problem and its application in railway transport

ANNOTATION

This thesis will deal with the problem of finding a simple path of maximum length in each graph. A path is called simple if it has no repeated vertices. The length of a path can either be measured by its number of edges or (in weighted graphs) by the sum of the weights of its edges. In contrast to the shortest path problem, which can be solved in polynomial time in cycle-free graphs with negative weights, the longest path problem is NP-hard. This problem can be applied e.g. in traffic problems.

KEYWORDS

Graph, vertex, edge, longest path, NP-hard, heuristic, Java

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

NP	Nondeterministic Polynomial-time
DAG	Directed Acyclic Graph
DP	Dynamic Programming
BF	Brute Force
GA	Genetic Algorithm
SA	Simulated Annealing
TS	Tabu Search
GUI	Graphical User Interface
JRE	Java Runtime Environment

ÚVOD

Vyhledávání nejdelší cesty v grafu patří mezi zásadní problémy diskrétní matematiky a teorie grafů, přičemž nachází široké uplatnění v různých oblastech, zejména v logistice, dopravních systémech, biotechnologiích či plánování výrobních procesů. Tento problém spočívá v nalezení takové jednoduché cesty v grafu, která má maximální možnou délku, přičemž jednoduchá cesta je taková, která neobsahuje žádný vrchol více než jednou. Délka cesty může být měřena různými způsoby, například počtem hran v případě neorientovaného neohodnoceného grafu nebo součtem vah hran v případě váženého grafu. [1]

Zatímco problém nejkratší cesty má efektivní řešení v polynomiálním čase například pomocí Dijkstrova algoritmu nebo Bellman-Fordova algoritmu, problém nejdelší cesty patří do třídy NP-těžkých problémů. To znamená, že pro obecné grafy neexistuje známý polynomiální algoritmus, který by jej vyřešil v rozumném čase. V případě, že graf obsahuje cykly se zápornou vahou, lze sice nejkratší cestu nalézt relativně snadno, avšak nalezení nejdelší cesty se v takových případech stává ještě složitějším a výpočetně náročnějším. [1]

Tato práce se zaměřuje na analýzu existujících přístupů k řešení problému nejdelší cesty, jejich výpočetní složitost a potenciální heuristické metody, které mohou být využity k nalezení přibližných řešení v přijatelném čase. Důraz bude kladen nejenom na aplikaci v železničním modelu, ale také v obecných grafech, kde se tento problém právě stává složitějším. Cílem práce je nejen zhodnocení různých algoritmických přístupů, ale také ověření vybraných metod na datech různého charakteru, což umožní lepší pochopení praktických dopadů tohoto teoretického problému.

1 ZÁKLADNÍ POJMY

Tato kapitola vychází z bakalářské práce autora, kde byly základní definice a pojmy již podrobně popsány. Vzhledem k neměnnosti dané problematiky a zaměření této práce na jinou část teorie grafů, se ponechávají původní formulace definic s případnými drobnými úpravami. Bakalářská práce se soustředila na rekurzivní algoritmy v teorii grafů.

1.1 Teorie grafů

Teorie grafů vznikla v 18. století díky práci Leonharda Eulera, který řešil známý problém sedmi mostů v Královci. Za zakladatele oboru je považován právě Euler, protože použil přístup, který dnes označujeme pojmem graf. V roce 1936 pak Dénes Kőnig publikoval první ucelenou učebnici věnovanou této disciplíně. Grafy slouží k přehlednému modelování vztahů mezi objekty a nacházejí široké uplatnění v mnoha oblastech. Jejich hlavní výhodou je možnost využití známých algoritmů a jednoduchá implementace v počítačových systémech. [2]

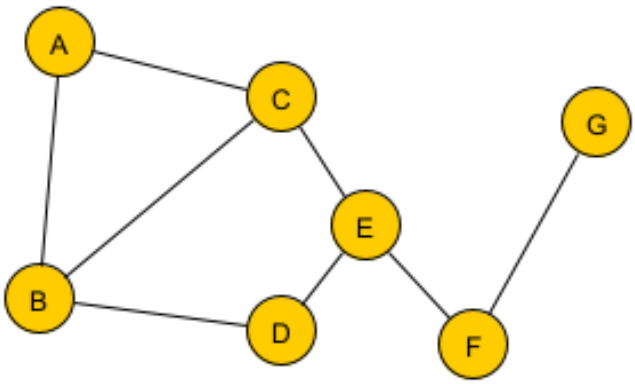
1.2 Graf

Graf je klíčovým pojmem v diskrétní matematice. Definujeme ho jako algebraickou strukturu, která slouží k popisu objektů a jejich vztahů. Graf se skládá z vrcholů a hran, které spojují tyto vrcholy. Nespornou výhodou je, že je lze snadno a přehledně zobrazit. Vrcholy se reprezentují jako body nebo symboly a hrany jako křivky, které je spojují. Tento způsob zobrazení je intuitivní a umožňuje snadno porozumět vztahům mezi vrcholy a hranami. [2] [3]

Při složitých analýzách a implementaci algoritmů pro řešení složitých úloh nestačí pouze intuitivní reprezentace grafů jako puntíků a čar v rovině. Podstatou struktury grafu jsou objekty a jejich vazby. Pokud mezi dvěma objekty existuje vazba, reprezentujeme ji jako dvojici (dvouprvkovou množinu) příslušných objektů (vrcholů). Pokud vazba mezi dvěma objekty neexistuje, tuto dvojici objektů do množiny hran nezařadíme. [2]

1.2.1 Neorientovaný graf

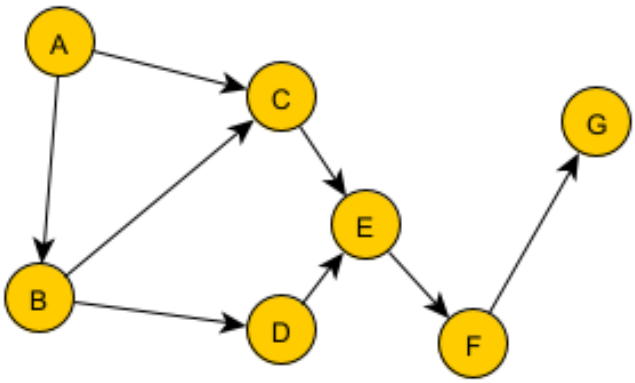
Definice 1: ***Neorientovaný graf*** *je trojice G = (V, E, ε) tvořená konečnou množinou V, jejíž prvky nazýváme uzly, konečnou množinou E, jejíž prvky nazýváme neorientovanými hranami a zobrazením ε (vztah incidence), které přiřazuje každé hraně e jedno nebo dvouprvkovou množinu uzlů*. [4]



Obrázek 1: Neorientovaný graf

1.2.2 Orientovaný graf

Definice 2: ***Orientovaný graf*** *je trojice G = (V, E, ε) tvořená konečnou množinou prvků V, jejíž prvky nazýváme uzly, konečnou množinou E, jejíž prvky nazýváme orientovanými hranami a zobrazením ε: E* $\rightarrow V^2$*, které nazýváme vztahem incidence, a které přiřazuje každé hraně e* $\in E$ *uspořádanou dvojici uzlů*. [4]



Obrázek 2: Orientovaný graf

1.2.3 Sled, tah, cesta, kružnice a dráha

Definice 3: ***Sledem*** *mezi vrcholy u a v, nazveme posloupnost vrcholů a hran:*

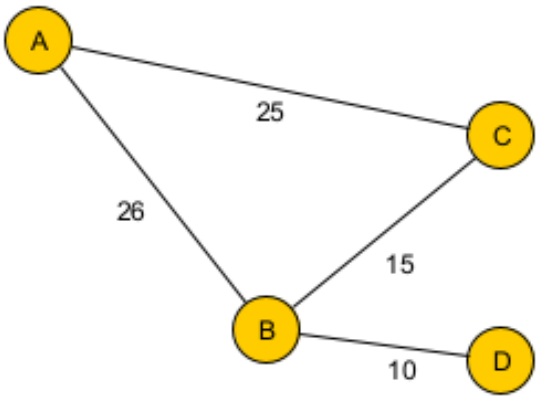
$S = \{u, h_1, u_1, h_2, \ldots, u_{n-1}, h_n, v\}$.

Vrcholy u a v budeme nazývat krajními vrcholy sledu. Pokud jsou krajní body sledu stejné, hovoříme o uzavřeném sledu. ***Sled****, ve kterém se neopakuje ani jedna hrana nazveme tahem. Tahem, ve kterém se neopakuje ani jeden vrchol nazveme* ***cestou****. Cestu mezi dvěma vrcholy budeme značit m (u, v).* ***Uzavřený sled****, ve kterém se neopakuje žádná hrana a žádný vrchol (kromě okrajového) nazýváme* ***kružnicí****. V případě orientovaného grafu orientovaného grafu se místo cesty používá termín* ***dráha****.* [5]

1.3 Ohodnocení grafu

Definice 4: ***Ohodnocení grafu*** *G je funkce* $w: E(G) \rightarrow R$*, která každé hraně* $e \in E(G)$ *přiřadí reálné číslo* $w(e)$*, kterému říkáme váha hrany.* ***Ohodnocený graf*** *je graf G spolu s ohodnocením hran reálnými čísly.* ***Kladně ohodnocený*** *(říkáme také vážený) graf G má takové ohodnocení w, že pro každou hranu* $e \in E(G)$ *je její váha* $w(e)$ *kladná*. [2]

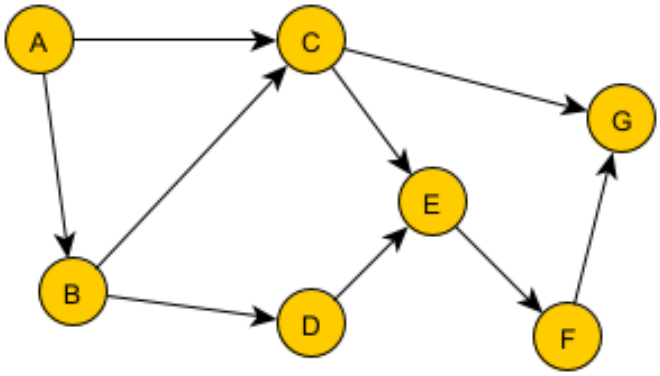
Hrany v grafu bývají nejčastěji ohodnoceny kladnými čísly, ale obecně mohou nést libovolné konečné reálné hodnoty. Pokud jsou všechny hodnoty kladné, označujeme takový graf jako vážený. V praxi se často používají přirozená čísla, protože odpovídají měřitelným veličinám a zároveň minimalizují chyby při výpočtech a zaokrouhlování napříč různými systémy. [2]



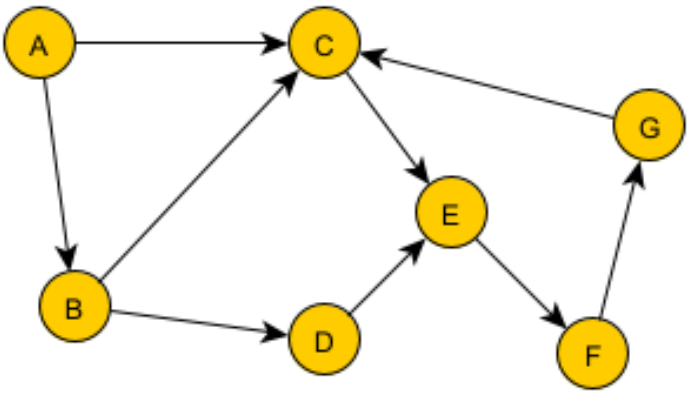
Obrázek 3: Ohodnocený graf

1.4 Cyklicita grafu

Graf, který neobsahuje žádný cyklus nazýváme acyklický. Disponuje-li alespoň jednou kružnicí, nazývá se graf cyklický. [4]



Obrázek 4: Acyklický graf



Obrázek 5: Cyklický graf

2 TŘÍDY SLOŽITOSTI

Výpočetní problémy lze klasifikovat podle jejich složitosti a časové náročnosti řešení. Tato kapitola se zaměřuje na základní rozdělení problémů na sledovatelné, nesledovatelné, rozhodovací a optimalizační, a také na nejvýznamnější třídy složitosti: P, NP, NP-těžké a NP-úplné.

2.1 Sledovatelné a nesledovatelné problémy

Sledovatelné problémy jsou takové, které lze řešit v čase, který je polynomiálně závislý na velikosti vstupu. To znamená, že existuje algoritmus, jehož doba běhu je omezena nějakým polynomem vzhledem k délce vstupu. Typickými příklady jsou problémy jako seřazení seznamu čísel, nalezení nejkratší cesty v grafu (např. Dijkstrův algoritmus) nebo vynásobení dvou matic. [7]

Zásadní rozdíl mezi sledovatelnými a nesledovatelnými problémy spočívá ve složitosti algoritmu – konkrétně v tom, zda má polynomiální nebo exponenciální časovou složitost. [7]

Polynomiální složitost (např. $O(n)$, $O(n^2)$, $O(n^3)$) znamená, že výpočetní čas roste „přijatelně“ s velikostí vstupu. Takové algoritmy jsou považovány za prakticky použitelné. [6]

Naopak exponenciální složitost (např. $O(2^n)$, $O(n!)$) znamená, že i mírné zvětšení vstupu dramaticky prodlužuje čas výpočtu. Tyto algoritmy se stávají neproveditelnými už pro střední velikosti vstupu. [6]

Typickým příkladem problému s exponenciální složitostí jsou Hanoiské věže, kde počet kroků nutných k přesunu n disků je dán vztahem $2^n - 1$. Tento růst ilustruje, jak rychle se takový problém stává výpočetně náročným. Tedy s přidáním jednoho disku se doba výpočtu zdvojnásobí. [10]

Exponenciální složitost se často objevuje u algoritmů, které využívají backtracking, tedy systematické prohledávání všech možných kombinací. Ačkoli backtracking může být optimalizován pomocí využití heuristik, v nejhorším případě má stále exponenciální složitost. Tento rozdíl ve složitosti má zásadní dopad na klasifikaci problémů a určení, zda je možné je efektivně řešit. [10]

Na základě této časové náročnosti jsou nesledovatelné problémy definovány jako takové, pro které žádný známý algoritmus nemá polynomiální časovou složitost. Jejich řešení je výpočetně náročné a často roste exponenciálně nebo rychleji se zvětšující se velikostí vstupu. To činí tyto problémy prakticky neřešitelnými pro větší instance. [7]

2.2 Rozhodovací a optimalizační problémy

Rozhodovací problémy jsou takové, na které lze odpovědět jednoduchým „ano“ nebo „ne“. Například: „Existuje v daném grafu cesta délky nejvýše 5 mezi vrcholy A a B?“ Rozhodovací problémy jsou klíčové zejména v teorii složitosti, protože mnoho tříd složitosti (např. třída NP) je definováno právě pro rozhodovací problémy. [7]

Optimalizační problémy naproti tomu hledají nejlepší možné řešení podle daného kritéria. Například: „Jaká je nejkratší cesta mezi vrcholy A a B?“ Každý optimalizační problém může být převeden na rozhodovací problém tím, že se přidá prahová hodnota a ověřuje se, zda existuje řešení lepší než tato hodnota. [7]

2.3 Třída P

Třída P obsahuje všechny rozhodovací problémy, které lze vyřešit deterministickým algoritmem v polynomiálním čase. To znamená, že doba výpočtu řešení roste nejvýše jako nějaká mocnina délky vstupu. Problémy v této třídě jsou považovány za efektivně řešitelné. Příklady zahrnují testování dělitelnosti, seřazení seznamu, nebo nalezení nejkratší cesty v grafu. [8]

2.4 Třída NP

Třída NP zahrnuje všechny rozhodovací problémy, pro které lze v polynomiálním čase ověřit správnost navrženého řešení. Jinými slovy, pokud někdo poskytne navržené řešení, lze rychle ověřit, zda je správné. Není ale známo, zda všechny problémy v NP lze také v polynomiálním čase vyřešit – to je podstatou otázky P vs NP. Typickým příkladem problému v NP je problém obchodního cestujícího ve své rozhodovací variantě („Existuje cesta kratší než X?“) nebo problém splnitelnosti booleovských výrazů (SAT). [7] [8]

2.5 NP-úplné problémy

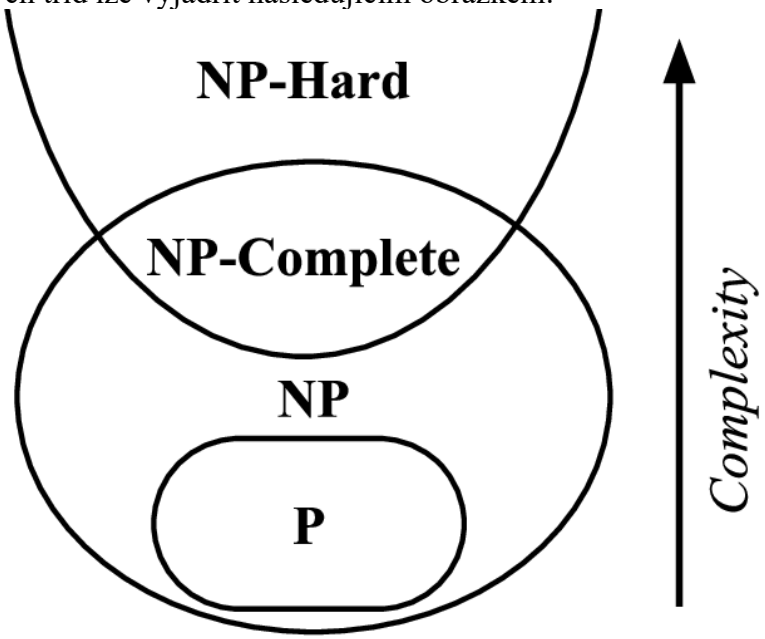
NP-úplné problémy tvoří podmnožinu třídy NP, která má zvláštní vlastnost: každý problém v NP lze převést na libovolný NP-úplný problém v polynomiálním čase. To znamená, že pokud by se našel efektivní algoritmus pro řešení jednoho NP-úplného problému, bylo by možné

efektivně řešit všechny problémy v NP. Mezi nejznámější NP-úplné problémy patří SAT, problém clique, problém pokrytí množin, nebo rozhodovací varianta problému obchodního cestujícího. [7]

2.6 NP-těžké problémy

NP-těžké problémy jsou alespoň tak obtížné jako NP-úplné problémy, ale na rozdíl od nich nemusí být rozhodovací – často jde o optimalizační problémy. To znamená, že na ně lze v polynomiálním čase převést libovolný problém z NP, ale nemusí samy patřit do NP (nemusí mít řešení ověřitelné v polynomiálním čase). Příkladem je optimalizační verze problému obchodního cestujícího („Najdi nejkratší možnou cestu“) nebo problém obecného plánování. [7]

Vztahy jednotlivých tříd lze vyjádřit následujícím obrázkem:



Obrázek 6: Diagram tříd složitosti [9]

3 PROBLEMATIKA NEJDELŠÍ CESTY

Problém nejdelší cesty patří mezi základní úlohy teorie grafů. Jde o nalezení nejdelší možné jednoduché cesty v grafu, tedy takové, která prochází maximálním počtem vrcholů, přičemž žádný z nich nenavštíví více než jednou. Tento problém je známý svou výpočetní složitostí, konkrétně NP-těžkostí, což znamená, že neexistuje známý polynomiální algoritmus, který by ho dokázal efektivně řešit pro obecné grafy. [1]

Jedním z dobře známých speciálních případů problému nejdelší cesty je Hamiltonovská cesta, která hledá cestu procházející každým vrcholem právě jednou. Rozhodovací verze tohoto problému (tj. zda taková cesta v daném grafu existuje) je NP-úplná. Z toho vyplývá, že problém nejdelší cesty je alespoň tak složitý jako Hamiltonovská cesta, a tudíž rovněž NP-těžký. [1]

Vzhledem k výpočetní složitosti problému nejdelší cesty existují pouze omezené případy, ve kterých je možné jej řešit efektivně. Polynomiální algoritmy byly nalezeny pouze pro určité speciální třídy grafů, mezi něž patří například stromy, intervalové grafy, blokové grafy či kaktusové grafy. Tento problém je tedy zajímavý nejen z teoretického hlediska, ale i pro jeho praktické využití v oblastech, jako je dopravní plánování, síťová analýza či biologické sekvencováni. [1]

3.1 Složitost Hamiltonovské cesty a její vztah k NP-těžkým problémům

Hamiltonovská cesta v grafu $G = (V, E, \varepsilon)$ je taková posloupnost vrcholů, která navštíví každý vrchol právě jednou a mezi každými sousedními vrcholy existuje hrana. Pokud taková cesta existuje mezi dvěma danými vrcholy s a t, označuje se jako rozhodovací problém D-HAM-PATH. Podobně se definuje Hamiltonovský cyklus, což je Hamiltonovská cesta, která se vrací do výchozího bodu, tedy propojuje první a poslední vrchol cesty hranou. [11]

3.1.1 NP-úplnost Hamiltonovské cesty

Hamiltonovská cesta je NP-úplný problém, což znamená:

- Patří do třídy NP, tedy pokud je dáno kandidátské řešení (například nějaká cesta), lze jeho správnost ověřit v polynomiálním čase.
- Je NP-těžký, což znamená, že k němu existuje polynomiální redukce z jiného NP-těžkého problému, konkrétně ze 3-SAT. [11]

3.1.2 Důkaz NP-úplnosti Hamiltonovské cesty

Důkaz NP-úplnosti probíhá redukcí z 3-SAT, což je klasický NP-úplný problém rozhodování o splnitelnosti Booleovské formule v konjunktivní normální formě. [11]

Ověřitelnost v polynomiálním čase

Pokud existuje nějakou posloupnost vrcholů jako kandidát na Hamiltonovskou cestu, lze ověřit, zda tato posloupnost obsahuje všechny vrcholy a každá sousední dvojice je spojena hranou. Tento proces lze provést v polynomiálním čase $O(n)$, což znamená, že problém patří do NP. [11]

Redukce z 3-SAT

Aby bylo možné dokázat, že Hamiltonovská cesta je NP-těžká, je nutné demonstrovat, že každý jiný problém z NP na ni lze redukovat v polynomiálním čase. [11]

Redukce funguje následovně:

- Pro každou proměnnou x_i v logické formuli se vytvoří podgraf, který reprezentuje dvě možné hodnoty proměnné (pravda nebo nepravda).
- Každá klauzule Cj je reprezentována speciálním vrcholem, který je propojen s odpovídajícími literály v proměnných.
- Celkový graf je navržen tak, že existuje Hamiltonovská cesta právě tehdy, když existuje splnitelné přiřazení proměnných ve vstupní logické formuli. [11]

Důležitým krokem v této konstrukci je zajištění, aby cesta „neuvízla“ v nějakém vrcholu a aby všechny vrcholy byly navštíveny právě jednou. [11]

Tento převod probíhá v polynomiálním čase $O(n \cdot m)$ (kde n je počet proměnných a m je počet klauzulí), a tím dokazuje, že pokud by bylo možné řešit Hamiltonovskou cestu v polynomiálním čase, bylo by možné také rozhodnout o splnitelnosti 3-SAT, což je ale velmi nepravděpodobné (za předpokladu, že $P \neq$ NP). [11]

3.1.3 Další důsledky NP-těžkosti

Důkaz NP-úplnosti Hamiltonovské cesty má zásadní dopad na posouzení složitosti dalších problémů v teorii grafů. Díky tomu, že Hamiltonovská cesta patří mezi základní NP-úplné problémy, lze její složitostí argumentovat i při dokazování obtížnosti jiných úloh:

- Hamiltonovský cyklus lze převést na Hamiltonovskou cestu tak, že se do grafu přidá nový speciální vrchol, který je spojen s počátečním a koncovým vrcholem cesty. Tím vznikne cyklus, který zahrnuje všechny původní vrcholy.
- Problém obchodního cestujícího (TSP) je považován za NP-těžký, protože zobecňuje Hamiltonovský cyklus o minimalizaci celkové délky cesty, čímž přidává optimalizační prvek nad rámec pouhé existence cesty.
- Problém nejdelší cesty je NP-těžký právě proto, že jako podproblém obsahuje i hledání Hamiltonovské cesty. Jinými slovy, pokud bychom dokázali efektivně nalézt nejdelší jednoduchou cestu v grafu, uměli bychom tím zároveň rozhodnout, zda v grafu existuje taková cesta, která projde každým vrcholem právě jednou. To jasně ukazuje přímou souvislost mezi těmito dvěma problémy a potvrzuje, že obecné řešení problému nejdelší cesty je alespoň stejně obtížné jako řešení Hamiltonovské cesty. [11]

3.2 Využití

Na rozdíl od nejkratší cesty má nejdelší cesta kromě dopravy specifické využití v dalších oblastech, kde je třeba zohlednit závislosti a maximální rozsah aktivit:

- Plánování projektů: Identifikace tzv. *kritické cesty*, která určuje celkovou dobu trvání projektu.
- Rozvrhování úloh: Stanovení nejzazšího možného začátku úloh bez ohrožení celkového termínu dokončení.
- Biotechnologie: Studium genomových sekvencí nebo modelování interakcí mezi proteiny.
- Kryptografie: Testování odolnosti systémů pomocí grafových reprezentací.
- Robotika: Plánování nejdelších tras např. při prohledávání prostoru (search and rescue scénáře).
- Energetika: Analýza elektrických sítí, např. nejdelší elektrická vzdálenost mezi dvěma body. [12]

3.2.1 Kritická cesta

Kritická cesta je pojem vycházející z teorie plánování projektů, kde reprezentuje nejdelší cestu závislostí mezi jednotlivými úlohami. Tato cesta určuje minimální dobu potřebnou k dokončení

celého projektu. Zpoždění kterékoliv úlohy na kritické cestě vede ke zpoždění celého projektu, a proto je důležitá pro správné rozvržení zdrojů a řízení rizik. Výpočet kritické cesty využívá algoritmus pro hledání nejdelší cesty v DAG, kde vrcholy představují úkoly a hrany závislosti mezi nimi. V prostředí s paralelizací úloh (např. více procesorů) je nutné zohlednit délky trvání úloh i jejich závislosti, přičemž kritická cesta pomáhá optimalizovat načasování celého harmonogramu. [13]

4 PŘEHLED ALGORITMŮ PRO HLEDÁNÍ NEJDELŠÍ CESTY

Tato kapitola se zabývá přehledem algoritmů pro hledání nejdelší cesty v grafu. Je rozdělena na jednotlivé podkapitoly, v nichž každá popisuje různé přístupy řešení v závislosti na typu grafu a výsledku, kterého je třeba dosáhnout.

4.1 Algoritmy pro DAG

4.1.1 Topologické třídění a dynamické programování

Hledání nejdelší cesty v orientovaném acyklickém grafu (DAG) lze efektivně řešit pomocí dynamického programování (DP), pokud je nejprve provedeno topologické třídění, což je proces, který uspořádá vrcholy tak, že všechny hrany vedou „zleva doprava“. To umožní procházet vrcholy ve správném pořadí, kde každý vrchol je zpracován až poté, co byly vyhodnoceny všechny jeho předchůdci. [15]

Cílem je najít nejdelší cestu mezi dvěma zadanými vrcholy, S a T. Tento problém lze řešit stejným způsobem jako hledání nejdelší cesty z S do všech vrcholů, ale pro výsledek je důležitá pouze hodnota $dist(T)$ a odpovídající cesta.

Algoritmus funguje následovně:

(1) Topologické setřídění vrcholů v grafu.

(2) Inicializace dynamického programování:

- Vytvoří se pole $dist(v)$, které bude obsahovat délku nejdelší cesty z vrcholu S do vrcholu v.
- Pro počáteční vrchol platí: $dist(S) = 0$.
- Pro všechny ostatní vrcholy se inicializuje $dist(v)$ na $-\infty$ (nebo minimální možnou hodnotu).

(3) Dynamické programování pro výpočet nejdelší cesty:

- Projdou se vrcholy v topologickém pořadí a pro každý vrchol u se aktualizuje vzdálenost jeho sousedů v podle vzorce:

$$dist(v) = \max\left(dist(v), dist(u) + w(u, v)\right)$$

Kde:

- $w\ (u, v)$ je váha hrany mezi u a v (pokud je graf neohodnocený, bere se váha jako 1).
- Pokud je cesta přes vrchol u delší než současná hodnota $dist(v)$, aktualizuje se a zapíše předchůdce v cestě: $previous(v) = u$.
- Tento krok se opakuje pro všechny vrcholy v topologickém pořadí.

(4) Rekonstrukce nejdelší cesty (Backtracking):

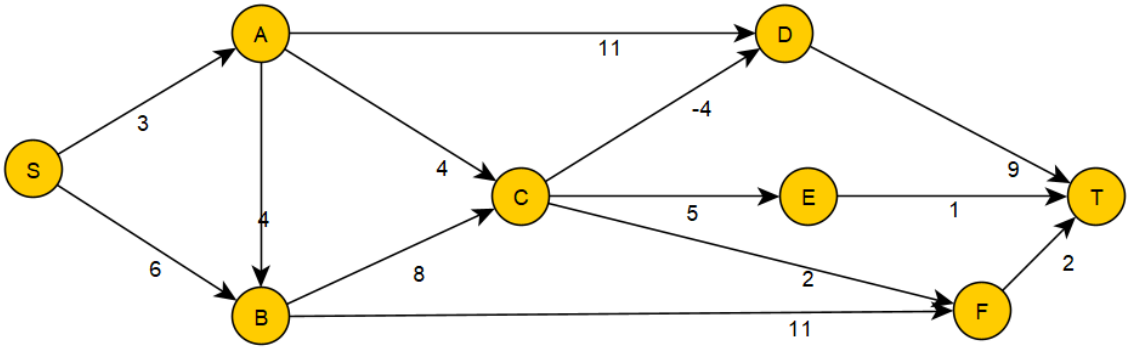
- Pokud $dist(T) = -\infty$, znamená to, že neexistuje žádná cesta z S do T.
- Pokud $dist(T) \neq -\infty$, lze rekonstruovat nejdelší cestu zpětným sledováním $previous(v)$, dokud se nenarazí na počáteční vrchol S.

[14]

Časová složitost tohoto algoritmu je lineární $O(V + E)$. Backtracking při rekonstrukci cesty lze elegantně řešit tím, že si algoritmus pamatuje předchozí vrchol, což umožňuje efektivní a přímou rekonstrukci cesty bez nutnosti zpětného prohledávání. [15]

Ukázka algoritmu

Průběh algoritmu bude demonstrován na následujícím grafu:



Obrázek 7: Ukázkový graf pro DAG algoritmus

Jelikož se jedná o směrový acyklický graf, lze algoritmus provést. Následně lze tedy provést topologické třídění, respektive uspořádání vrcholů zleva doprava.

Schéma grafu po aplikování topologického řazení je následující uspořádání: S → A → B → C → D → E → F → T

Dalším krokem je průchod všech vrcholů v tomto pořadí a aktualizace jejich sousedů podle již zmiňovaného vzorce:

$$dist(v) = \max\left(dist(v), dist(u) + w(u,v)\right)$$

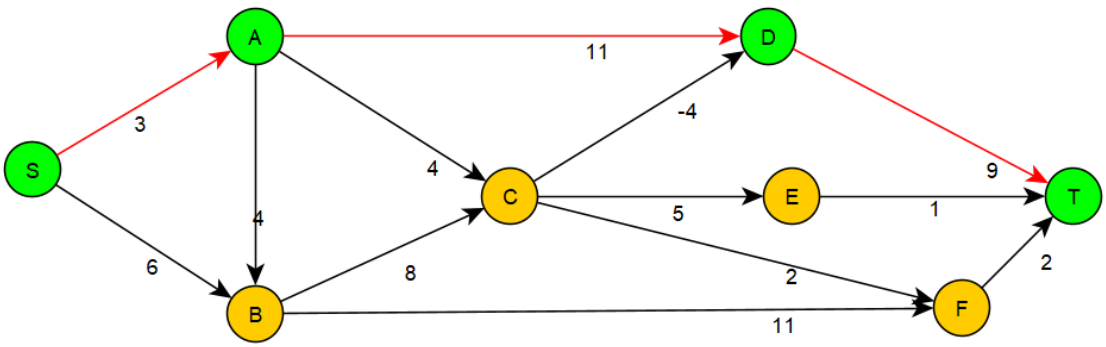
Průběh výpočtu pro jednotlivé vrcholy je následující:

- Vrchol S:
  - $dist(A) = 0 + 3 = 3,\ předchůdce(A) = S$
  - $dist(B) = 0 + 6 = 6,\ předchůdce(B) = S$
- Vrchol A:
  - $dist(B) = max\,(6, 3 + 4) = 7,\ předchůdce(B) = A$
  - $dist(C) = max\,(-\infty, 3 + 4) = 7,\ předchůdce(C) = A$
  - $dist(D) = max\,(-\infty, 3 + 11) = 14,\ předchůdce(D) = A$
- Vrchol B:
  - $dist(C) = max\,(7, 7 + 8) = 15,\ předchůdce(C) = B$

- $dist(F) = max\ (-\infty, 7 + 11) = 18,\ předchůdce(F) = B$

- Vrchol C:
  - $dist(D) = max\ (14, 15 + (-4)) = 14$, (nezmění se)
  - $dist(E) = \max(-\infty, 15 + 5) = 20,\ předchůdce(E) = C$
  - $dist(F) = max\ (18, 15 + 2) = 18,$ (nezmění se)
- Vrchol D:
  - $dist(T) = max\ (-\infty, 14 + 9) = 23,\ předchůdce(T) = D$
- Vrchol E:
  - $dist(T) = max\ (23, 20 + 1) = 23$, (nezmění se)
- Vrchol F:
  - $dist(T) = max\ (23, 18 + 2) = 23$, (nezmění se)

Po dokončení aktualizací vzdáleností mezi vrcholy je zjištěno, že nejdelší cesta má délku 23. Tato cesta je následně zpětně rekonstruována pomocí předchůdců a reverzním průchodem je zjištěno finální pořadí vrcholů (S → A → D → T).



Obrázek 8: Výsledná nejdelší cesta po provedení DAG algoritmu

4.2 Algoritmy pro obecné grafy

Co se týče obecných grafů, tak zde existuje několik možných způsobů od modifikace jiných algoritmů až po řešení, která vyžadují prohledání všech možných kombinací.

4.2.1 Bellman-Fordův algoritmus

Bellman-Fordův algoritmus je určen pro hledání nejkratší cesty z jednoho výchozího vrcholu do všech ostatních ve váženém orientovaném grafu s n vrcholy a m hranami. Na rozdíl od Dijkstrova algoritmu dokáže pracovat i s grafy obsahujícími hrany se zápornými váhami, což jej činí užitečným v širokém spektru aplikací. [16]

Jednou z klíčových vlastností tohoto algoritmu je schopnost detekovat záporné cykly, tedy cykly, jejichž součet vah je záporný. Pokud takový cyklus v grafu existuje, znamená to, že neexistuje reálná nejkratší cesta – hmotnost cesty by totiž mohla být neomezeně zmenšována opakovaným průchodem tímto cyklem. Algoritmus lze tedy upravit nejen pro nalezení nejkratších cest, ale i pro identifikaci přítomnosti takových cyklů. [16]

Tento algoritmus nese jméno dvou amerických vědců, Richarda Bellmana a Lestera Forda. Ford jej poprvé navrhl v roce 1956 při řešení jiného matematického problému, který se nakonec redukoval na hledání nejkratších cest v grafu. Bellman poté v roce 1958 publikoval samostatnou práci, v níž algoritmus popsal v podobě, v jaké jej známe dnes. [16]

Díky své schopnosti pracovat s libovolnými reálnými vahami a jednoduché implementaci je Bellman-Ford široce využíván nejen v teoretické informatice, ale i v síťových protokolech (např. RIP – Routing Information Protocol), finančním modelování nebo analýze toků v dopravních systémech. [16]

Modifikace na řešení nejdelší cesty

Přestože je Bellman-Fordův algoritmus určen pro hledání nejkratší cesty, lze jej snadno modifikovat pro nalezení nejdelší cesty. Klíčovou změnou je inverze vah hran, tedy místo minimalizace součtu vah se hledá jejich maximum. To lze provést například změnou znamének hran (např. každá váha w se nahradí $-w$), čímž se problém nejdelší cesty převede na klasickou úlohu nejkratší cesty. Po spuštění standardního Bellman-Forda vznikne nejkratší cesta v transformovaném grafu, kterou stačí přepočítat zpět na nejdelší cestu v původním grafu. Tento přístup funguje pouze v stále pouze jen v grafech, které nemají cykly se zápornou váhou. Pro

DAG je efektivnější topologické uspořádání s relaxací hran v jednosměrném průchodu, což umožní nalezení nejdelší cesty v lineárním čase. [17]

Algoritmus funguje následovně:

(1) Inicializace

- Vytvoří se pole $dist(v)$, které bude obsahovat nejkratší vzdálenost z výchozího vrcholu S do každého vrcholu v.
- Pro počáteční vrchol platí $dist(S) = 0$.
- Pro všechny ostatní vrcholy se inicializuje $dist(v) = \infty$ (nebo maximální možnou hodnotu).
- Vytvoří se pole $previous(v)$, které bude sloužit k rekonstrukci nejkratší cesty.

(2) Relaxace hran (opakuje se |V| - 1krát):

- Opakovaně se prochází všechny hrany (u, v, w) v grafu a aktualizují se vzdálenosti podle vzorce:

$$dist(v) = \min(dist(v), dist(u) + w(u, v))$$

- Pokud je cesta přes u kratší než současná hodnota $dist(v)$, aktualizuje se a zaznamená se předchůdce: $previous(v) = u$.

(3) Detekce záporných cyklů:

- Po |V| - 1 iteracích se provede ještě jeden průchod všemi hranami.
- Pokud lze ještě zkrátit nějakou vzdálenost $dist(v)$, znamená to, že graf obsahuje záporný cyklus, protože algoritmus by už neměl najít kratší cestu.
- V takovém případě algoritmus ukončí běh a nahlásí existenci záporného cyklu.

(4) Rekonstrukce nejdelší cesty (Backtracking):

- Pokud $dist(T) = \infty$, znamená to, že neexistuje žádná cesta z S do T.
- Pokud $dist(T) \neq \infty$, lze rekonstruovat nejdelší cestu zpětným sledováním $previous(v)$, dokud se nenarazí na počáteční vrchol S.

[16]

Hlavní část algoritmu (relaxace hran) běží $O(V \cdot E)$, což jej činí pomalejším než a Dijkstrův algoritmus $O((E + V) \cdot log\, V)$. [17]

4.2.2 Dijkstrův algoritmus

Dijkstrův algoritmus je známý algoritmus používaný pro hledání nejkratších cest v grafu, konkrétně v orientovaných nebo neorientovaných grafech s nezápornými váhami hran. Tento algoritmus byl vyvinut v roce 1956 nizozemským informatikem Edsgerem W. Dijkstrou a patří mezi základní metody v teorii grafů. Jeho hlavním cílem je nalézt nejkratší cestu mezi počátečním uzlem a všemi ostatními uzly v grafu, což ho činí velmi užitečným v mnoha oblastech, jako jsou síťové protokoly, navigační systémy, či optimalizace dopravních cest. [18]

Algoritmus byl poprvé publikován v roce 1959. Byl navržen k řešení problému hledání nejkratších cest v síti, kde váhy hran reprezentovaly náklady nebo vzdálenosti. Algoritmus měl významný vliv na rozvoj oblasti teorie grafů a jeho aplikace se rychle rozšířily do praxe, zejména v oblastech jako je správa sítí nebo doprava. [18]

Algoritmus funguje následovně:

(1) Inicializace

- Vytvoří se pole $dist[]$, které bude obsahovat nejkratší vzdálenosti od počátečního vrcholu S k ostatním vrcholům.
- Pro počáteční vrchol platí $dist(S) = 0$.
- Pro všechny ostatní vrcholy se inicializuje $dist(v) = \infty$ (nebo velmi vysokou hodnotu).
- Vytvoří se prázdná množina zpracovaných vrcholů, které již byly navštíveny a prozkoumány.

(2) Hlavní smyčka algoritmu

- Opakovaně se vybírá nenavštívený vrchol s nejnižší hodnotou v poli $dist[]$.
- Tento vrchol se označí jako zpracovaný.
- Pro všechny sousední vrcholy vrcholu, který je zpracován se provede relaxace hran.

(3) Relaxace hran

- Pro každou hranu (u, v, w) v grafu, kde u je aktuální vrchol a v je sousední vrchol s váhou hrany $w(u, v)$, se provádí následující kroky:
  - Pokud je vzdálenost do vrcholu v přes vrchol u kratší než dosavadní hodnota $dist(v)$, aktualizuje se podle vzorce:

$$dist(v) = \min(dist(v), dist(u) + w(u, v))$$

- Pokud došlo k aktualizaci vzdálenosti, zapíše se vrchol u jako předchůdce vrcholu v do pole $previous(v)$, aby bylo možné rekonstruovat cestu.

(4) Ukončení algoritmu

- Jakmile jsou všechny vrcholy zpracovány, algoritmus končí.
- V poli $dist[]$ se nachází nejkratší vzdálenosti od počátečního vrcholu k ostatním vrcholům v grafu.

(5) Rekonstrukce cesty (Backtracking)

- Pokud je potřeba rekonstruovat konkrétní nejkratší cestu, začne se od cílového vrcholu a zpětně se prochází pole $previous[]$ až k počátečnímu vrcholu.
- Pokud je v poli $dist[]$ pro cílový vrchol hodnota ∞, znamená to, že cesta neexistuje.

[19]

Modifikace na řešení nejdelší cesty

Dijkstrův algoritmus je navržen pro minimalizaci součtu vah cest a předpokládá se, že všechny váhy hran jsou nezáporné. To znamená, že relaxace, která by vedla ke zvýšení váhy cesty, není nikdy prováděna. Pokud by byl Dijkstrův algoritmus upraven pro hledání nejdelší cesty, narazilo by se na problém – algoritmus by stále hledal nejkratší cestu a možnost maximalizace váhy cesty by byla ignorována. [20]

V případě záporných vah by Dijkstrův algoritmus selhal, protože nelze zaručit, že váha nebude měněna zpětně. Tento problém by nastal i při inverzi vah – v takovém případě by Dijkstra místo hledání maximální váhy hledal minimální váhu, což by nevedlo k požadovanému výsledku. [20]

4.2.3 Brute force algoritmus

Brute force algoritmy (BF) jsou jednoduché, ale výkonné techniky pro řešení problémů, které spočívají v důkladném prozkoumání všech možných řešení. Tyto metody nevyužívají žádné sofistikované optimalizace nebo zkratky, ale místo toho se zaměřují na vyzkoušení každé možné možnosti, dokud není nalezen správný výsledek. BF přístup lze považovat za nejjednodušší, ale výpočetně náročný způsob řešení problémů. Pokud jde o teorii grafů, tak i zde BF algoritmy mají své uplatnění. [21]

Koncept BF v problému nejdelší cesty

V kontextu procházení grafu BF přístup znamená systematické prozkoumání všech možných cest mezi počátečním a cílovým vrcholem grafu. To zahrnuje prozkoumání každého vrcholu a hrany, vyhodnocení všech možných kombinací a výběr té s nejdelší délkou. [21]

Tento problém lze jednoduše demonstrovat na příkladu hledání nejdelší cesty mezi dvěma městy, kde vrcholy představují města a hrany představují silnice s různými vzdálenostmi mezi nimi. BF metoda by zahrnovala kontrolu každé možné kombinace měst a silnic, výpočet celkové vzdálenosti pro každou cestu a výběr té s největší vzdáleností. [21]

Brute force algoritmy jsou snadno implementovatelné a pochopitelné. Nevyžadují žádné speciální znalosti o struktuře problému, což je činí užitečnými v případech, kdy složitější metody nejsou snadno aplikovatelné, nebo když je problém dostatečně malý, aby výpočetní výkon nebyl omezením. Hlavní výhodou této metody je zaručení optimálního řešení, protože algoritmus prozkoumá každou možnost, čímž zajistí, že žádná potenciální cesta nebude opomenuta. [21]

Problém nastává při řešení problémů, které mají na vstupu větší množství dat. Počet možných kombinací roste tak rapidně, že exaktní metody jsou použitelné pouze pro omezené instance problémů. Za určitou kritickou hranicí velikosti vstupních dat se tyto metody stávají prakticky nepoužitelnými v reálném čase. [21]

Výpočetní složitost BF algoritmů při procházení grafu, jako je hledání nejdelší cesty, je obecně exponenciální. Konkrétně, pokud graf má n vrcholů a m hran, počet možných cest by mohl být v řádu $O(2^n)$, protože každá cesta v grafu může obsahovat nebo neobsahovat určitou hranu. To

činí algoritmus nepraktickým pro velké grafy, protože počet výpočtů rychle roste s velikostí grafu. [21]

4.3 Heuristické a metaheuristické přístupy

Heuristiky a metaheuristiky představují důležitou skupinu metod pro řešení složitých optimalizačních úloh, kde klasické algoritmy selhávají nebo jsou výpočetně neúnosné. Tyto přístupy nezaručují nalezení optimálního řešení, ale místo toho nabízejí efektivní způsoby hledání dostatečně kvalitního řešení v rozumném čase.

Mezi nejznámější metaheuristické metody patří genetické algoritmy, simulované žíhání, částicové roje, mravenčí kolonie, tabu search a další. Jejich společným rysem je inspirace přírodními nebo sociálními jevy a schopnost adaptivně prohledávat prostor řešení pomocí stochastických prvků, jako je náhodná mutace, selekce nebo řízené prohledávání. [22]

4.3.1 Genetický algoritmus

Genetické algoritmy představují jednu z nejznámějších a nejpoužívanějších skupin evolučních algoritmů, které jsou inspirovány principy biologické evoluce a přirozeného výběru. Jejich historie sahá do 60. a 70. let 20. století, kdy byly poprvé systematicky zkoumány zejména díky pracím Johna Hollanda na Michiganské univerzitě. Holland ve své průlomové knize *Adaptation in Natural and Artificial Systems* (1975) představil teoretický rámec, na jehož základě lze modelovat evoluční procesy v počítačovém prostředí. Jeho cílem bylo pochopit principy adaptace a navrhnout algoritmy, které by byly schopny samostatně hledat řešení složitých úloh bez explicitního programování každého kroku. [23] [25]

V následujících desetiletích došlo k rozvoji a rozšíření genetických algoritmů do celé řady oblastí, a to zejména díky jejich schopnosti efektivně prohledávat rozsáhlé a složité prostory řešení. GA se ukázaly jako velmi účinné při řešení úloh, kde tradiční deterministické algoritmy selhávají nebo jsou příliš nákladné výpočetně, například u optimalizačních problémů s mnoha lokálními extrémy, v kombinatorických úlohách nebo při návrhu složitých systémů. [23]

Základní myšlenkou genetického algoritmu je existence populace potenciálních řešení, která v každé generaci podléhá selekci, křížení (crossover) a mutaci. Vhodnost jednotlivých řešení je vyhodnocována pomocí tzv. fitness funkce, která udává, jak dobře dané řešení odpovídá požadovanému cíli. Na základě těchto hodnot jsou vybírány jedinci pro tvorbu nové generace.

V průběhu opakovaného aplikování těchto operátorů dochází k postupnému zlepšování kvality řešení a k přibližování se optimálnímu nebo dostatečně kvalitnímu řešení problému. [24]

Velkou výhodou genetických algoritmů je jejich flexibilita – nevyžadují znalost derivací nebo spojitosti funkce, a navíc umožňují paralelní zpracování více kandidátních řešení. Díky tomu jsou vhodné i pro úlohy, jejichž řešení není možné jednoduše popsat analyticky. Na druhé straně je třeba mít na paměti i jejich nevýhody, jako je relativně vysoká výpočetní náročnost nebo riziko předčasné konvergence na suboptimální řešení, pokud není dostatečně zachována diverzita populace. [24]

V současnosti nacházejí genetické algoritmy široké uplatnění nejen v informatice, ale i v oblastech jako je strojírenství, biotechnologie, finanční analýza, robotika či umělá inteligence. [24]

Algoritmus funguje následovně:

(1) Inicializace populace

- Na začátku algoritmu je vytvořena počáteční populace jedinců. Tato populace je zpravidla generována náhodně, přičemž každý jedinec reprezentuje jedno možné řešení dané úlohy. Velikost populace je důležitým parametrem ovlivňujícím rychlost a kvalitu konvergence.

(2) Vyhodnocení jedinců (fitness funkce)

- Každému jedinci je přiřazena hodnota tzv. *fitness*, která udává, jak dobré je jeho řešení ve vztahu k cíli optimalizace. Tato funkce je specifická pro daný problém – v této práci např. délka nalezené cesty v grafu.

(3) Výběr rodičů (selekce)

- Z aktuální populace jsou na základě jejich fitness vybráni jedinci, kteří budou použiti k vytvoření nové generace. Časté metody výběru jsou např. ruletový výběr, turnajová selekce nebo výběr podle pořadí.

(4) Křížení (crossover)

- Vybraní rodiče jsou kombinováni pomocí operátoru křížení za účelem vytvoření nových potomků. Cílem je zkombinovat vlastnosti obou rodičů

a vytvořit jedince s potenciálně lepšími vlastnostmi. Existuje více variant křížení – jednobodové, dvoubodové, uniformní aj.

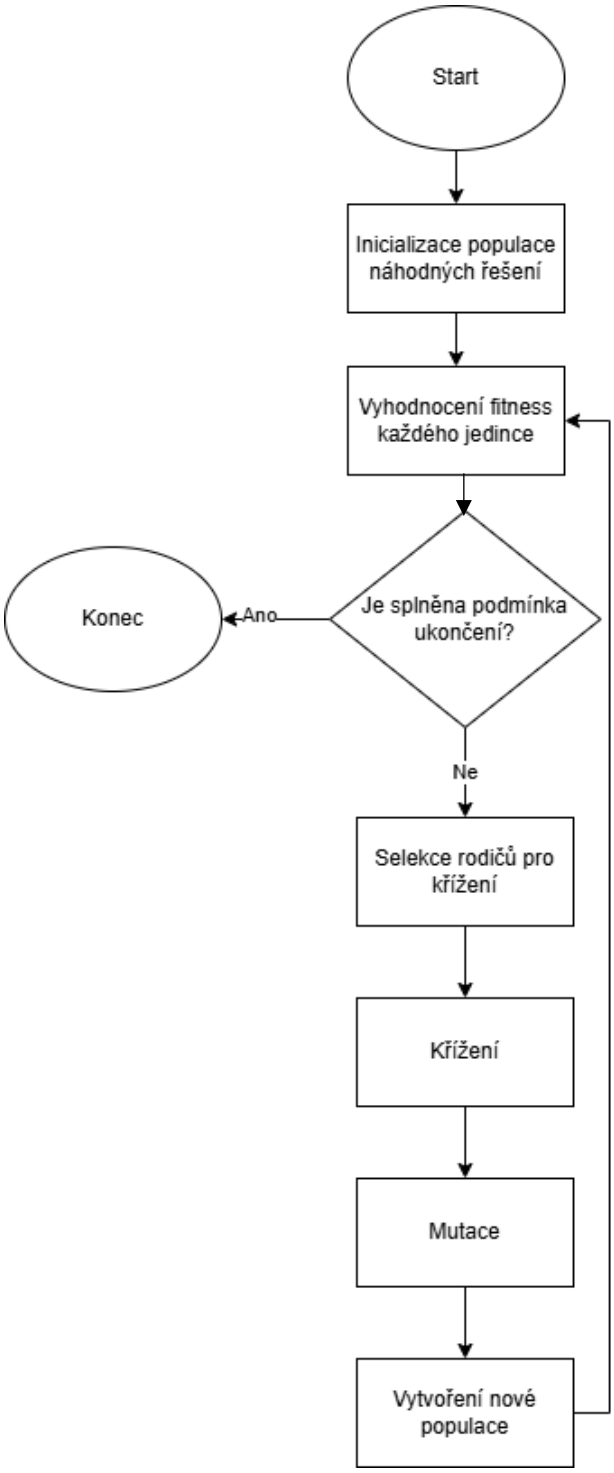
(5) Mutace

- Na nově vzniklé potomky je aplikována mutace – náhodná změna jednoho nebo více genů v chromozomu. Tento krok zajišťuje genetickou diverzitu v populaci a zabraňuje předčasné konvergenci.

(6) Vytvoření nové generace

- Potomci (někdy v kombinaci s nejlepšími jedinci předchozí generace – tzv. *elitismus*) tvoří novou generaci populace. Následně se celý proces (od bodu 2) opakuje, dokud není dokončen předem stanovený počet generací nebo není dosaženo jiné podmínky ukončení.

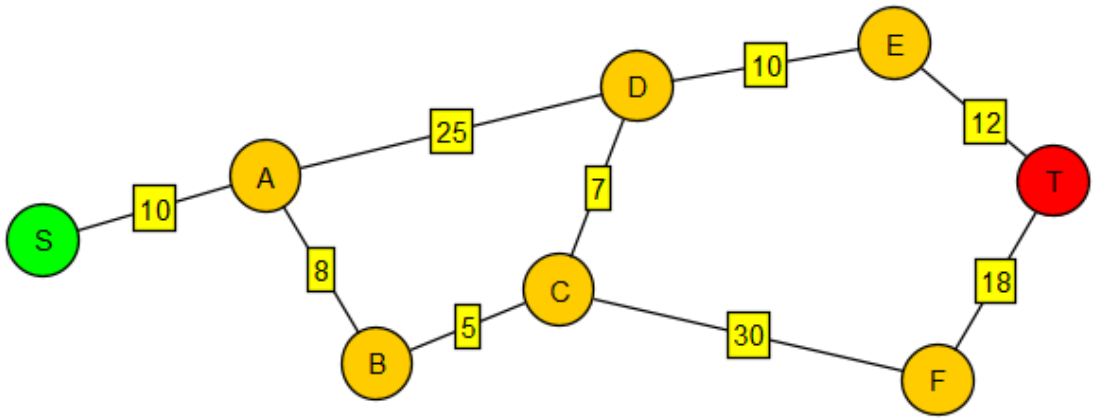
[24]



Obrázek 9: Vývojový diagram GA

Ukázka algoritmu

Nastínění průběhu algoritmu bude demonstrováno na následujícím ohodnoceném neorientovaném grafu, přičemž se budeme snažit najít nejdelší cestu mezi vrcholy *S* a *T*.



Obrázek 10: Graf pro ukázku GA

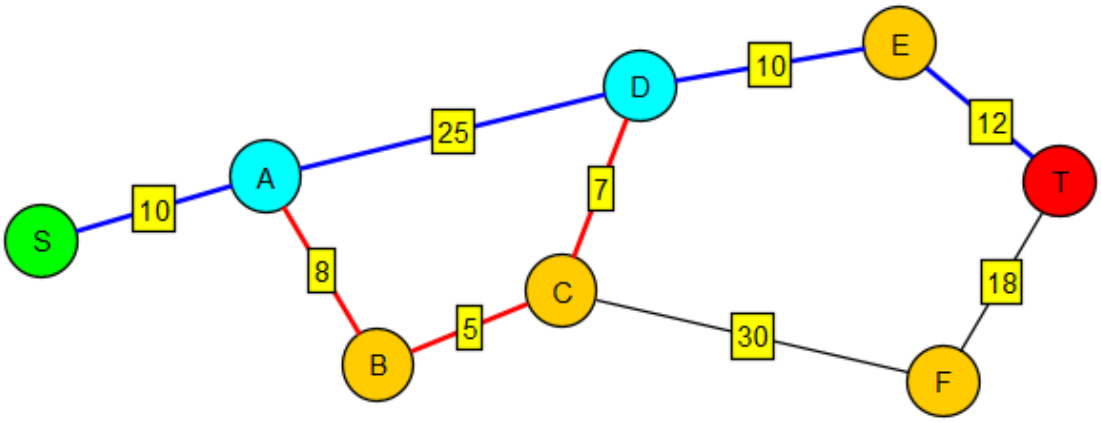
Prvním krokem je vytvoření výchozí populace, zde záleží, jaká je nastavena velikost, podle toho vznikne takový počet cest, příkladem mohou být tyto cesty:

1. S → A → D → E → T (délka 57)
2. S → A → B → C → D → E → T (délka 52)
3. S → A → D → C → F → T (délka 90)

Následně se pro každou cestu vypočítá hodnota fitness funkce. V tomto případě se může jednat buď o počet hran v cestě, nebo o součet vah jednotlivých hran, které danou cestu tvoří.

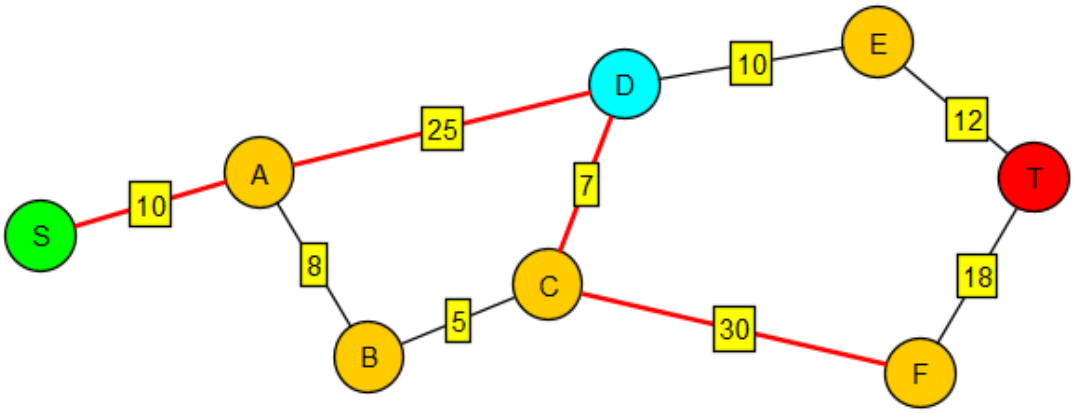
V dalším kroku se vyberou pomocí turnajového výběru rodiče (cesty), kteří se mezi sebou kříží pro vytvoření nového potomka (nové cesty).

Mechanismus křížení je navržen tak, že u vybraných rodičovských cest jsou nejprve identifikovány společné (průnikové) vrcholy, z nichž je následně jeden náhodně vybrán. Nově vzniklá cesta je pak složena z úvodního segmentu první rodičovské cesty až po zvolený průnikový bod a z navazující části druhé rodičovské cesty od tohoto bodu až do cíle. V případě křížení dvou cest z našeho ukázkového grafu lze například nalézt průsečíky ve vrcholech *A* a *D*. Nicméně u takto malého grafu by ani jedno z těchto míst nevedlo ke vzniku nové unikátní cesty. V případě, že křížení nelze provést, zachová se jeden z rodičů. U větších a složitějších grafů však tento způsob křížení výrazně přispívá k variabilitě generovaných řešení a zvyšuje šanci na nalezení optimální nebo alespoň suboptimální cesty.



Obrázek 11: Ukázka průniků dvou cest při křížení v GA

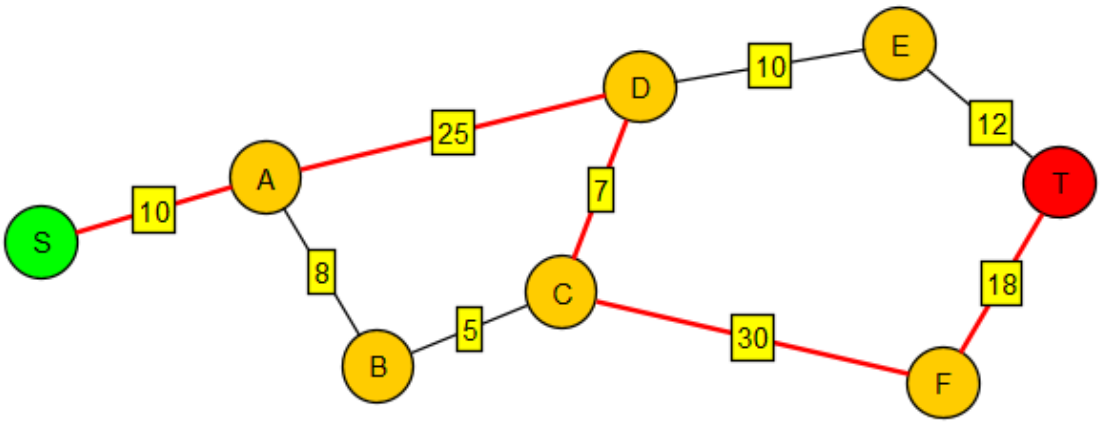
Po každém křížení je s předem definovanou pravděpodobností provedena mutace. Ta spočívá v přerušení cesty v náhodně zvoleném bodě, od kterého je následně snaha sestavit novou část cesty směřující k cílovému vrcholu. Tento mechanismus zajišťuje vyšší variabilitu v populaci a přispívá k prozkoumávání širšího prostoru možných řešení, čímž se také zvyšuje pravděpodobnost nalezení kvalitnější cesty. Pokud by tedy byla první cesta z příkladu přerušena například ve vrcholu D, mutace by mohla vytvořit novou, a dokonce delší cestu: S → A → D → C → F → T.



Obrázek 12: Ukázka mutace cesty v GA

Posledním krokem každé generace je sestavení nové populace o stejné velikosti. Ta je tvořena jednak elitními jedinci, kteří jsou do nové generace převedeni beze změny, a dále nově vzniklými jedinci vytvořenými pomocí křížení a případné mutace. Tento postup zajišťuje rovnováhu mezi zachováním kvalitních řešení a průběžným objevováním nových variant.

Tento proces se opakuje po předem stanovený počet generací. Konečný výsledek závisí na kvalitě zvolených parametrů algoritmu (např. velikost populace, počet generací, míra mutace či způsob výběru rodičů). V závislosti na těchto nastaveních může výsledná cesta představovat jak podprůměrné řešení, tak i optimální – tedy skutečně nejdelší nalezenou cestu v grafu.



Obrázek 13: Výsledná cesta po dokončení GA

4.3.2 Simulované žíhání

Algoritmus simulovaného žíhání (SA) je inspirován fyzikálním procesem žíhání pevných látek, při kterém je materiál zahříván na vysokou teplotu a poté pomalu ochlazován. Během tohoto procesu se částice mohou uspořádat do stabilní, nízkoenergetické struktury. Pokud však ochlazování probíhá příliš rychle (tzv. *quenching*), může se systém „zaseknout“ v méně výhodném, nestabilním stavu. [26]

Tento fyzikální proces inspiroval Metropolisův algoritmus, který simuluje pohyb atomů pomocí náhodného výběru změn (perturbací). Každá nová konfigurace (stav) je přijata buď:

- pokud vede ke snížení energie (lepší stav),
- nebo s určitou pravděpodobností, která závisí na teplotě a rozdílu energie (tzv. *Metropolisovo kritérium*). [25]

Pravděpodobnost přijetí horšího stavu je dána vztahem:

$$prob(\Delta E, t) = exp\left(-\frac{\Delta E}{k_B t}\right)$$

kde ΔE je nárůst energie, t je teplota a k_B je Boltzmannova konstanta. [25]

Tento princip byl později využit k řešení kombinatorických optimalizačních úloh. Myšlenkou je přirovnat:

- stav systému ↔ možné řešení problému,
- energie systému ↔ hodnota cílové funkce. [26]

Algoritmus začíná na vysoké teplotě, kdy často přijímá i horší řešení (kvůli větší pravděpodobnosti přijetí). Postupně, jak teplota klesá, se přechází od náhodného prohledávání k jemnému ladění v okolí již nalezených dobrých řešení. Klíčové je, aby tzv. *cooling schedule* (plán ochlazování) byl dostatečně pozvolný – jen tehdy může algoritmus dosáhnout rovnováhy na každé teplotní úrovni a vyhnout se uvíznutí v lokálním optimu. [26]

V každé iteraci je z aktuálního stavu s vygenerován sousední stav s', který je:

- přijat vždy, pokud $f(s') < f(s)$(lepší řešení),
- přijat s pravděpodobností závislou na rozdílu $\Delta f = f(s') - f(s)$ a aktuální teplotě t, analogicky podle vzorce:

$$prob(\Delta f, t) = exp\left(\frac{-\Delta f}{t}\right)$$

Nejlepší nalezené řešení se průběžně uchovává a algoritmus se zastavuje, pokud při konstantní teplotě není přijat žádný nový stav. To značí, že se systém „zafixoval“ – tedy „zamrzl“ v daném řešení. [25]

Teoreticky bylo dokázáno, že pokud teplota klesá podle určitých podmínek (např. $t_k \geq \frac{c}{log(k+1)}$, algoritmus konverguje k optimálnímu řešení. Tato podmínka je ale v praxi výpočetně náročná. Proto se používají jednodušší (ale méně přesné) postupy. [25]

Simulované žíhání nachází široké uplatnění v praxi – např. v plánování výroby, rozvrhování, návrhu sítí či dopravních systémů. [25]

Algoritmus funguje následovně:

(1) Inicializace

- Nastaví se počáteční teplota $t0$ (vysoká hodnota) a vybere počáteční řešení s.
- Definuje se plán ochlazování, který určuje, jak rychle teplota klesá.

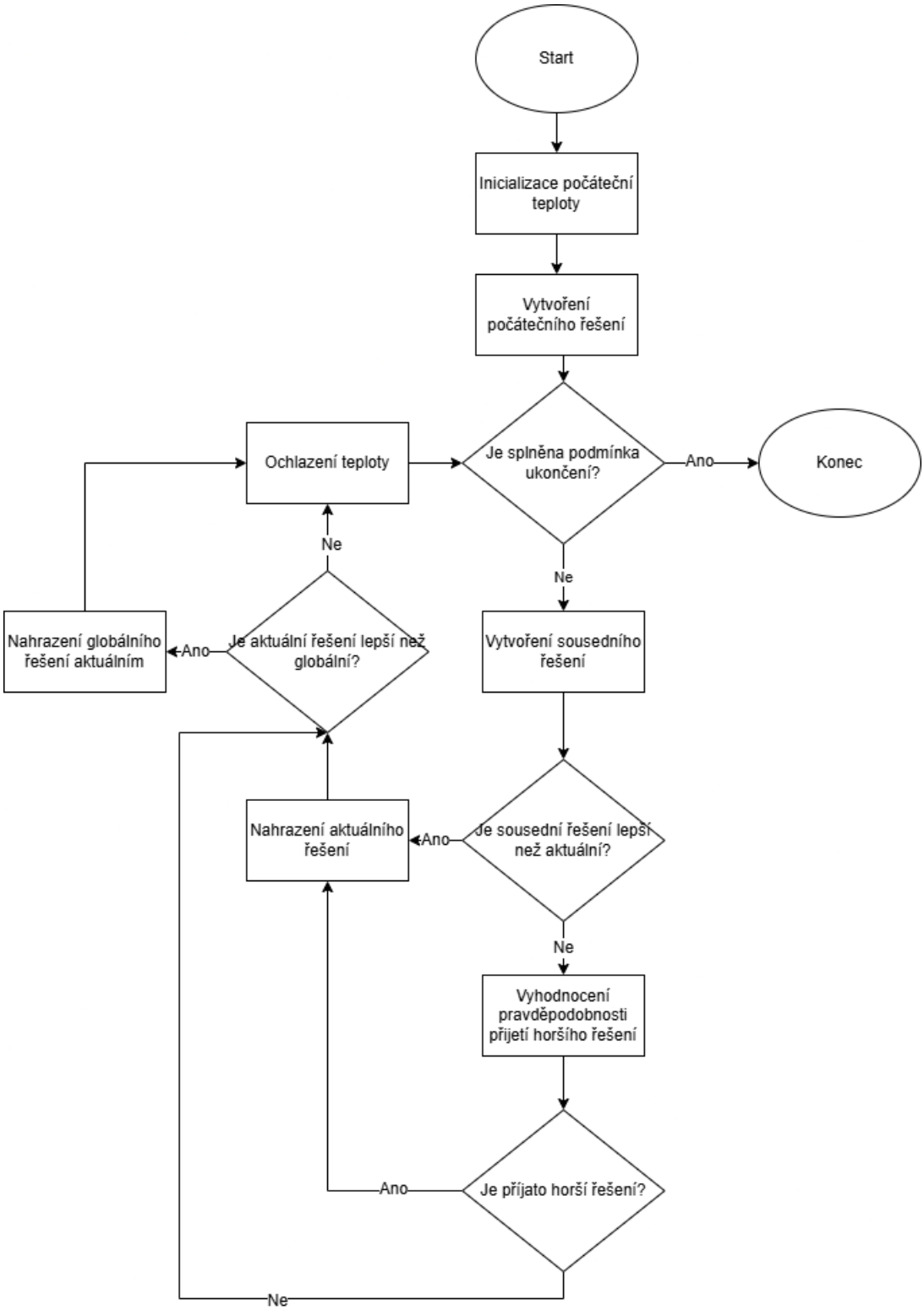
(2) Hlavní cyklus

- Generování sousedního řešení
  - Z aktuálního řešení s se náhodně vygeneruje „sousední“ stav s' (např. drobná modifikace cesty v grafu).
  - Vypočte se změna cílové funkce $\Delta f = f(s') - f(s)$.
- Přijetí nového řešení
  - Pokud $\Delta f \leq 0$ (lepší řešení), přijme se s'.
  - Pokud $\Delta f \geq 0$ (horší řešení), přijme se s' s pravděpodobností, která již byla uvedena v úvodním textu. (Čím vyšší teplota nebo menší Δf, tím vyšší pravděpodobnost přijetí.)
- Aktualizace nejlepšího řešení
  - Pokud s' je lepší než dosavadní nejlepší řešení s^*, uloží se $s^* = s'$.

(3) Ochlazování

- Teplota se sníží podle plánu ochlazování.
- Algoritmus pokračuje, dokud není dosaženo kritéria ukončení (např. teplota klesne pod prahovou hodnotu nebo počet iterací bez zlepšení překročí limit).

[25]



Obrázek 14: Vývojový diagram SA

4.3.3 Tabu search

Tabu Search (TS) je pokročilá metaheuristická metoda určená pro řešení kombinatorických optimalizačních problémů. Tento algoritmus byl vyvinut Fredem Gloverem v 80. letech a jeho hlavní přínos spočívá v rozšíření klasického lokálního prohledávání o mechanismus paměti. Díky tomu dokáže TS efektivně překonávat lokální optima a vyhýbat se opakovanému prohledávání již navštívených oblastí prostoru řešení. [25]

Na rozdíl od standardních lokálních metod, které obvykle akceptují pouze lepší nebo mírně horší řešení, TS systematicky eviduje nedávno provedené kroky nebo navštívená řešení a zakazuje jejich opětovné použití po určitý čas – tyto záznamy jsou uloženy na tzv. *tabu seznamu*. Existence tohoto seznamu pomáhá algoritmu opustit lokální extrémy a směřovat do nových oblastí vyhledávacího prostoru. [25]

Algoritmus v každé iteraci generuje množinu sousedních řešení k aktuálnímu stavu a vybírá z nich nejlepšího kandidáta – i tehdy, pokud se jedná o horší řešení než současné. Pokud je tento kandidát na tabu seznamu, je jeho výběr zakázán. Výjimku tvoří situace, kdy dané řešení splňuje tzv. *aspirační kritérium* – například pokud by vedlo k dosud nejlepšímu známému výsledku. V takovém případě může být zákaz ignorován. [25]

Tabu seznam má omezenou délku a funguje na principu omezené fronty – po každém kroku se do něj přidá nový záznam a nejstarší je odstraněn. Díky tomu algoritmus nezůstává „uvězněn“ v cyklu mezi několika řešeními, ale naopak prozkoumává širší část prostoru, čímž zvyšuje pravděpodobnost nalezení kvalitního globálního optima. [25]

Zvláštní pozornost si zaslouží výběr sousedů – u některých složitých úloh totiž může být množina všech sousedních řešení příliš rozsáhlá. Proto se často omezuje pouze na náhodně nebo heuristicky zvolenou podmnožinu, čímž se šetří výpočetní náročnost. [27]

Algoritmus obvykle běží buď po předem daný počet iterací, nebo dokud není nalezeno dostatečně kvalitní řešení. Alternativně může skončit i v případě, že po určitý počet kroků nedojde k žádnému zlepšení nejlepšího řešení. [27]

Oproti jiným metaheuristikám (např. již zmiňovaných GA nebo SA) vyniká důmyslným řízením paměti a systematickým obcházením neproduktivních částí prostoru řešení.

Algoritmus funguje následovně:

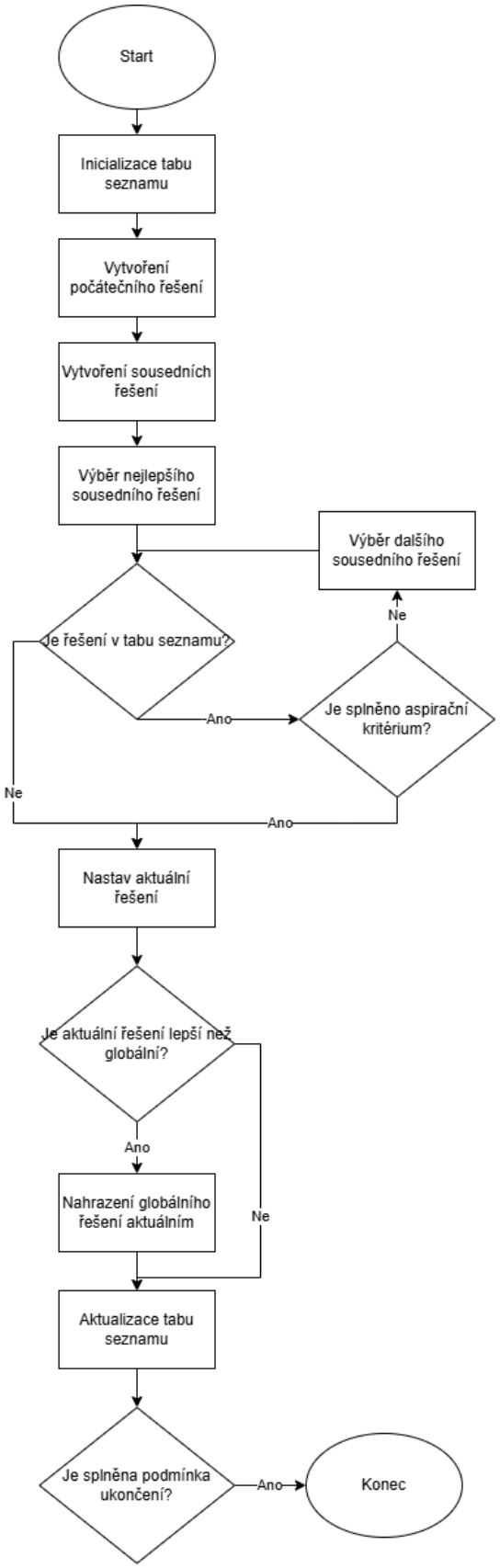
(1) Inicializace

- Zvolí se počáteční řešení $s \in X$.
- Nastaví se nejlepší nalezené řešení $s^* = s$.
- Inicializuje se tabu seznam T

(2) Hlavní cyklus

- Vygeneruje se podmnožina sousedních řešení $N' \subseteq N(s)$
- Výběr nejlepšího souseda s', který buď není na tabu seznamu, nebo splňuje aspirační kritérium.
- Aktualizace seznamu tabu (přidání současného *s*, odstranění nejstaršího).
- Přechod na nové řešení $s = s'$.
- Pokud s' zlepšilo dosavadní nejlepší řešení, nastaví se: $s^* = s'$.
- Cyklus běží, dokud není splněna podmínka ukončení (např. maximální počet iterací).

[25]



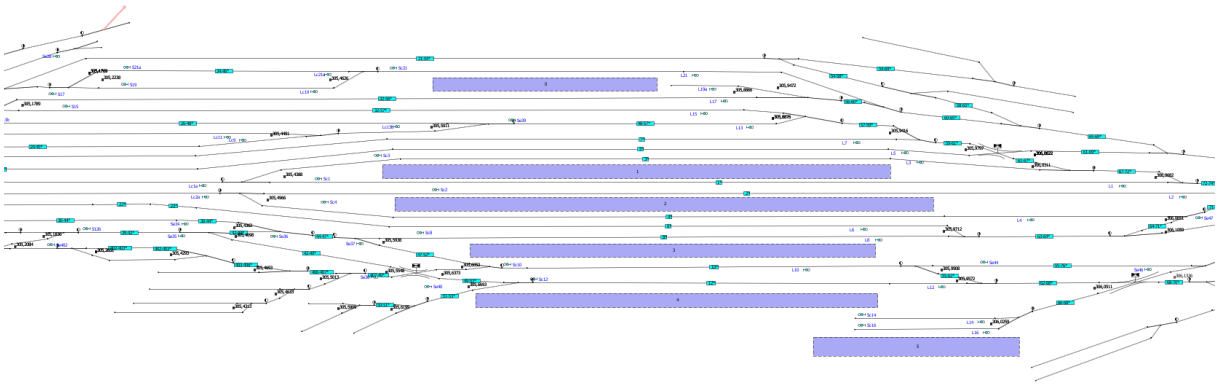
Obrázek 15: Vývojový diagram TS

5 EXPERIMENTY A VÝSLEDKY

Tato kapitola prezentuje experimentální ověření chování algoritmů pro hledání nejdelších jízdních cest. Cílem je porovnat chování jednotlivých metaheuristických metod na železničním modelu a také na různě velkých obecných grafech, přičemž budou obměňovány některé vstupní parametry. Výsledky budou porovnány jak na časové, tak i kvalitativní úrovni.

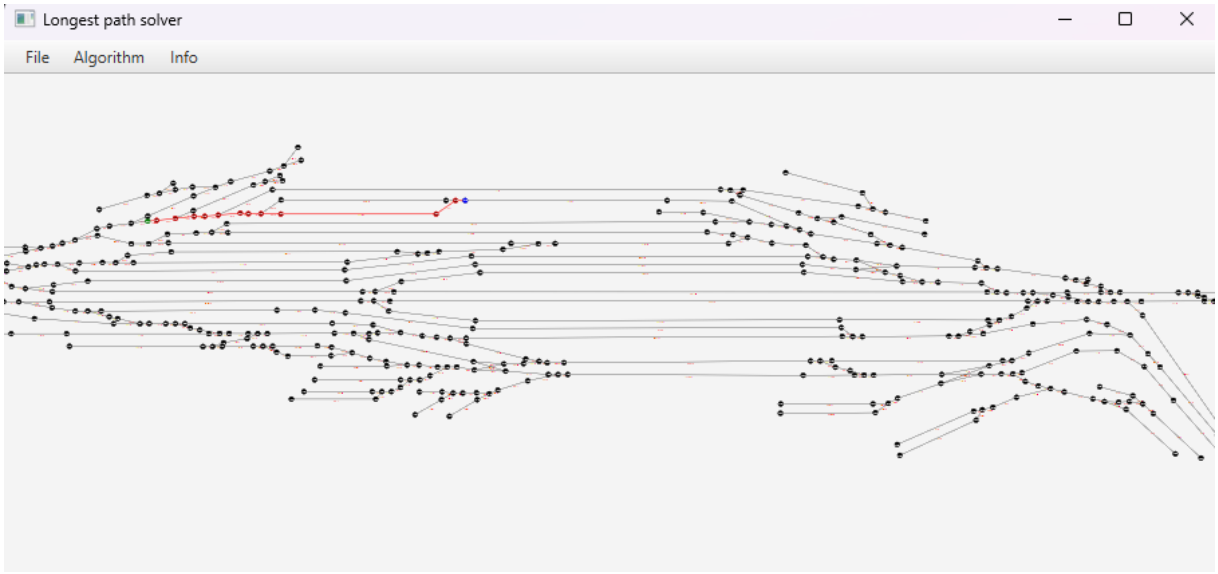
5.1 Železniční model Pardubického hlavního nádraží

Experimentální část práce využívá model Pardubického hlavního nádraží, který byl vytvořen na základě reálných dat poskytnutých ve formátu XML a ověřen pomocí softwaru MesoRail pro kontrolu správnosti topologie. Model této aplikace disponuje řadou detailnějších informací, které však pro účely této práce nejsou důležité.



Obrázek 16: Model z aplikace MesoRail

Výsledný model je reprezentován jako neorientovaný ohodnocený graf, kde vrcholy odpovídají klíčovým bodům infrastruktury a hrany znázorňují kolejové úseky s váhami udávajícími jejich skutečnou délku v metrech. Celkově graf obsahuje 677 vrcholů a 710 hran.



Obrázek 17: Ukázka části modelu Pardubice hlavní nádraží

5.1.1 Kolejové objekty a pohybová omezení

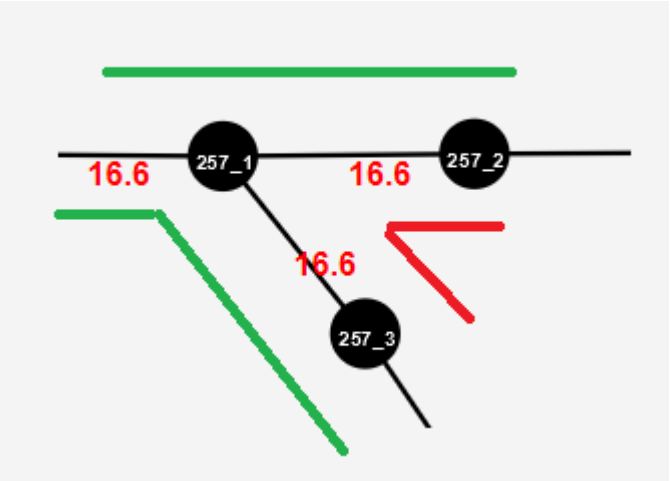
Model zahrnuje tři typy kolejových objektů s odlišnými pravidly průjezdu, která byla nutná pro realistické simulace pohybu vlaků:

- Segment (jednoduchá kolej) – standardní hrana grafu bez omezení.
- Single Cross (výhybka) – umožňuje pouze specifické průjezdy, které odráží chování vlaků v reálném světě.
- Double Cross (křižovatka) – obdobně jako u výhybky povoluje jen některé průchody.

V aplikaci jsou tyto průchody ošetřeny podmínkami, které explicitně zakazují neplatné kombinace průjezdu křižovatkami. Konkrétně pro křižovatku typu SINGLE_CROSS platí následující omezení:

```
if ("SINGLE_CROSS".equals(currentEdge.getPrototypeType())
    && "SINGLE_CROSS".equals(previousEdge.getPrototypeType())) {
    if (previousId.endsWith("_12") && currentId.endsWith("_13"))
        return false;
    if (previousId.endsWith("_13") && currentId.endsWith("_12"))
        return false;
}
```

Kód 1: Omezení průchodu pro SINGLE_CROSS



Obrázek 18: Omezení průchodu pro SINGLE_CROSS

Tato omezení vedou ke vzniku řídkého grafu, kde je počet možných cest výrazně redukován. Z hlediska hledání nejdelší jízdní dráhy to zjednodušuje problém, protože vlaky nemohou libovolně kličkovat po síti.

5.1.2 Experimentální ověření algoritmů

Pro účely experimentu byly definovány dva testovací scénáře, ve kterých byly zvoleny dva od sebe co nejvíce vzdálené vrcholy v rámci modelu:

- Scénář 1: nejlevější bod a nejnižší bod v modelu (pohledově vlevo a dole),
- Scénář 2: nejlevější bod a nejpravější bod v modelu (pohledově vlevo a vpravo).

Tyto kombinace zajišťují výrazně odlišnou strukturu hledaných tras a umožňují posoudit chování algoritmů i z hlediska různých topologických charakteristik sítě.

V každém scénáři byla pomocí BF nalezena všechna platná řešení, jejichž počet slouží jako referenční údaj pro ostatní algoritmy:

- Scénář 1: 26 různých cest,
- Scénář 2: 40 různých cest.

Pro dosažení relevantních výsledků byl každý uvedený algoritmus s příslušnou konfigurací proveden ve 100 iteracích. Údaj o úspěšnosti ve výsledných tabulkách udává, kolikrát byla heuristikou s danými parametry nalezena nejdelší cesta v procentech. Tento přístup umožňuje objektivně vyhodnotit výkon jednotlivých algoritmů.

Tabulka 1: Výsledky experimentu – Scénář 1 (nejlevější a nejnižší bod)

Algoritmus	Úspěšnost (%)	Průměrná délka cesty (m)	Průměrný čas (s)
BF	100	12512.48	0.0288
GA(1)	59	12512.45	0.0486
GA(2)	97	12512.48	0.2251
SA(1)	100	12512.48	0.0671
SA(2)	100	12512.48	0.6227
TS(1)	100	12512.48	0.6560

Tabulka 2: Výsledky experimentu – Scénář 2 (nejlevější a nejpravější bod)

Algoritmus	Úspěšnost (%)	Průměrná délka cesty (m)	Průměrný čas (s)
BF	100	23301.27	0.0325
GA(1)	25	23301.19	0.0592
GA(2)	76	23301.26	0.3347
SA(1)	64	23301.26	0.0289
SA(2)	100	23301.27	0.5228
TS(1)	100	23301.27	0.3632

Poznámky k parametrům algoritmů:

- GA(1): populace = 10, elita = 3, mutace = 20 %, počet generací = 100, turnaj = 10
- GA(2): populace = 50, elita = 10, mutace = 20 %, počet generací = 100, turnaj = 10
- SA(1): počáteční teplota = 100, maximální počet iterací = 1000, ochlazení = 0.90
- SA(2): počáteční teplota = 100, maximální počet iterací = 1000, ochlazení = 0.99
- TS(1): velikost tabu seznamu = 5, počet iterací = 10

Experimenty na železničním modelu ukázaly, že BF zaručuje vždy správné řešení, v nekompromisně rychlém čase oproti ostatním metodám. Genetické algoritmy vyžadují větší populaci pro dobré výsledky, simulované žíhání fungovalo nejlépe při pomalém ochlazování (např. 0.99) a tabu search dosáhl vždy optimálního řešení, ale za cenu nejdelšího výpočetního času.

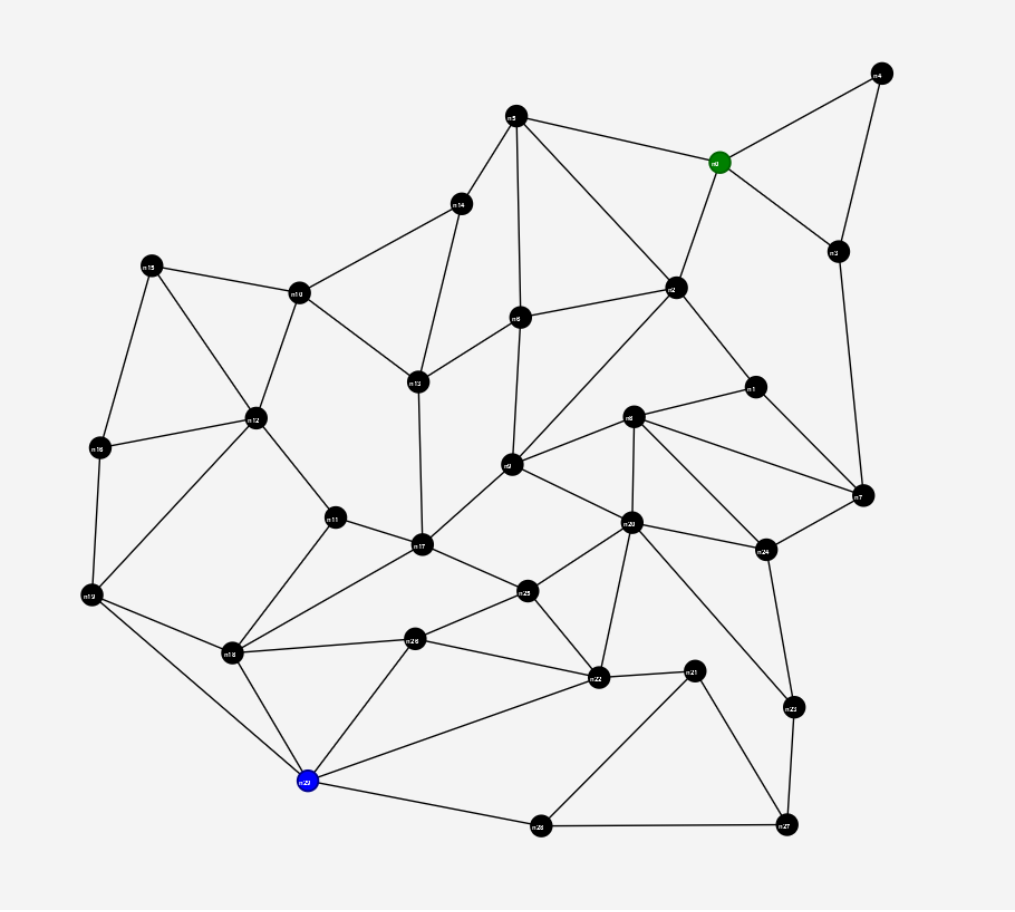
Průměrné délky cest u metaheuristik přitom ukazují, že algoritmy obvykle nalezly velmi dlouhé trasy – často jen o pár metrů kratší než ta nejdelší možná. To naznačuje, že výsledek pravděpodobně uvízl v některém z lokálních extrémů, například druhé nejdelší cesty, a tedy poskytl stále velmi kvalitní řešení.

5.2 Obecné grafy

Experimenty budou prováděny na třech různých neohodnocených neorientovaných grafech různé velikosti, konkrétně o 30, 50 a 100 vrcholech. Cílem je otestovat chování jednotlivých algoritmů na grafech s rostoucí složitostí a ověřit jejich škálovatelnost a schopnost nalézt dostatečně kvalitní řešení i při větším počtu uzlů. Pro všechny algoritmy budou zvoleny stejné počáteční a koncové body. Každý algoritmus byl spuštěn alespoň v 10 iteracích, aby byly výsledky relevantní a statisticky významné.

5.2.1 Graf s 30 vrcholy

Testovací graf pro tuto fázi experimentu obsahuje celkem 30 vrcholů a 59 hran. Jedná se o neohodnocený neorientovaný graf střední hustoty.



Obrázek 19: Neorientovaný neohodnocený graf s 30 vrcholy

Tabulka 3: Výsledky experimentu – Graf s 30 vrcholy

Algoritmus	Úspěšnost (%)	Průměrný počet hran v cestě	Průměrný čas (s)
BF	100	29	2.6077
GA(1)	30	27.86	0.0565
GA(2)	51	28.33	0.4172
GA(3)	89	28.88	2.2797
SA(1)	4	25.72	0.0015
SA(2)	14	27.64	0.0068
SA(3)	52	28.14	0.0185
TS(1)	3	26.28	0.0021
TS(2)	40	28.12	0.0124
TS(3)	100	29	0.154

Poznámky k parametrům algoritmů:

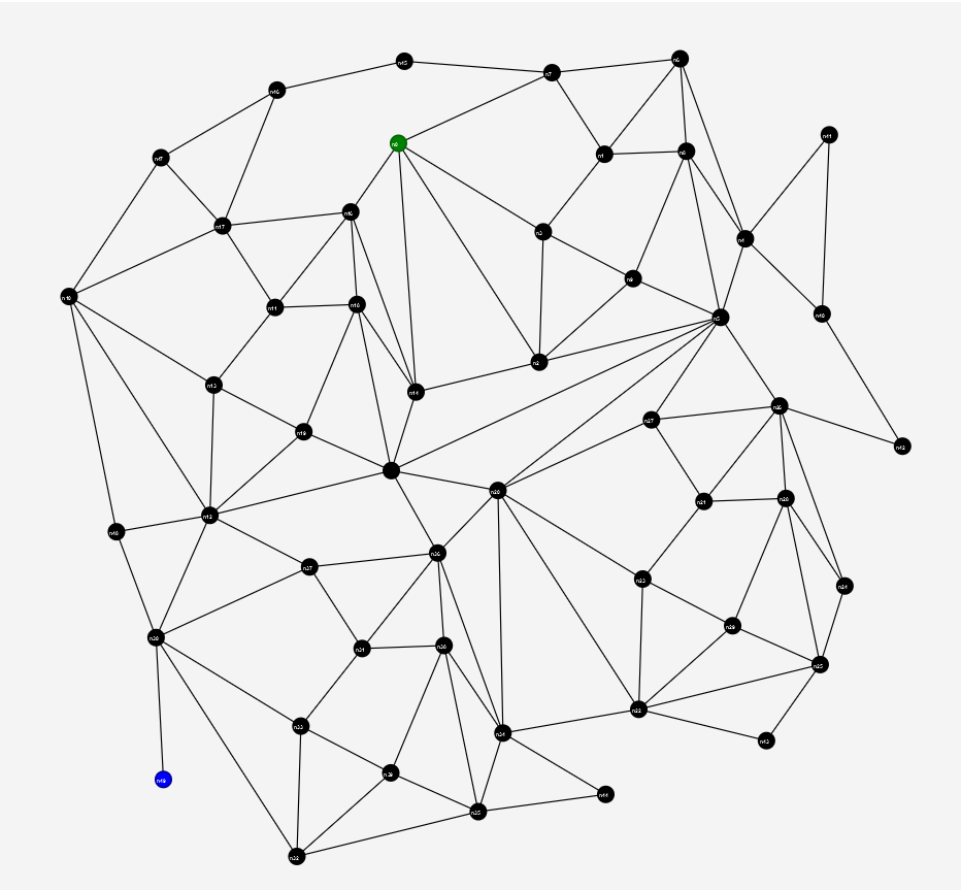
- GA(1): populace = 50, elita = 10, mutace = 20 %, počet generací = 100, turnaj = 10
- GA(2): populace = 100, elita = 15, mutace = 20 %, počet generací = 200, turnaj = 20
- GA(3): populace = 100, elita = 20, mutace = 40 %, počet generací = 1000, turnaj = 20
- SA(1): počáteční teplota = 100, maximální počet iterací = 1000, ochlazení = 0.90
- SA(2): počáteční teplota = 100, maximální počet iterací = 1000, ochlazení = 0.99
- SA(3): počáteční teplota = 10000, maximální počet iterací = 10000, ochlazení = 0.99
- TS(1): velikost tabu seznamu = 5, počet iterací = 10
- TS(2): velikost tabu seznamu = 50, počet iterací = 100
- TS(3): velikost tabu seznamu = 200, počet iterací = 1000

Z výsledků testu na grafu s 30 vrcholy vyplývá, že exaktní Brute Force algoritmus dokáže najít optimální řešení za nízký čas. Heuristické metody jako genetické algoritmy a tabu search při vhodném nastavení parametrů dosahují výsledků blízkých optimu, přičemž výrazně snižují čas potřebný k výpočtu. Simulované žíhání je v této konfiguraci méně úspěšné, avšak jeho rychlost je velmi dobrá.

5.2.2 Graf s 50 vrcholy

Druhý testovaný graf obsahuje 50 vrcholů a 109 hran. Tento graf již představuje výrazně komplexnější strukturu než graf o 30 vrcholech, a tudíž umožňuje lépe otestovat schopnost jednotlivých algoritmů nalézt dostatečně dlouhé cesty ve větším prostoru. Díky vyšší hustotě spojení narůstá počet možných cest mezi počátečním a koncovým bodem, což klade větší nároky především na exaktní metody.

Na tomto grafu již Brute force algoritmus nebyl prakticky použitelný. Jeho časová náročnost rychle roste s počtem vrcholů, a při testování na podobném grafu o 35 vrcholech byl čas výpočtu přibližně 15 minut.



Obrázek 20: Neorientovaný neohodnocený graf s 50 vrcholy

Tabulka 4: Výsledky experimentu – Graf s 50 vrcholy

Algoritmus	Průměrný počet hran v cestě	Průměrný čas (s)
GA(1)	44.64	42.94
GA(2)	46.1	199.36
GA(3)	46.4	180.00
SA(1)	42.6	13.05
SA(2)	44.1	67.47
SA(3)	46.3	201.04
TS(1)	25.5	13.40
TS(2)	42.5	166.08
TS(3)	46.2	228.54

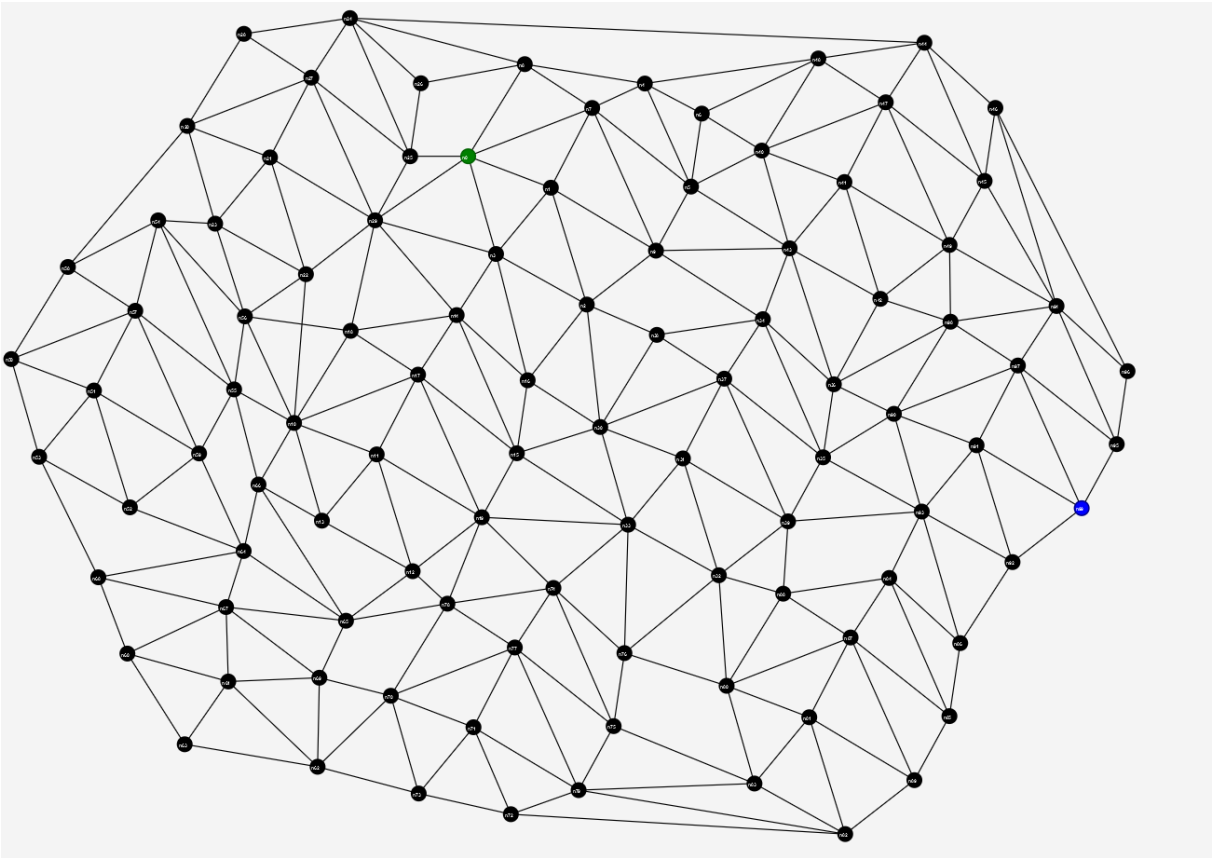
- GA(1): populace = 50, elita = 10, mutace = 20 %, počet generací = 100, turnaj = 10
- GA(2): populace = 100, elita = 15, mutace = 20 %, počet generací = 200, turnaj = 20
- GA(3): populace = 30, elita = 15, mutace = 20 %, počet generací = 1000, turnaj = 20
- SA(1): počáteční teplota = 100, maximální počet iterací = 1000, ochlazení = 0.90
- SA(2): počáteční teplota = 100, maximální počet iterací = 1000, ochlazení = 0.99
- SA(3): počáteční teplota = 10000, maximální počet iterací = 10000, ochlazení = 0.99
- TS(1): velikost tabu seznamu = 20, počet iterací = 20
- TS(2): velikost tabu seznamu = 50, počet iterací = 100
- TS(3): velikost tabu seznamu = 200, počet iterací = 150

Při zvětšení grafu na 50 vrcholů již exaktní metody nejsou prakticky použitelné vzhledem k časové náročnosti. Heuristiky však stále dokážou nalézt kvalitní řešení, i když za cenu delšího výpočetního času. Zlepšení parametrů (větší populace, více generací, delší tabu seznam) vede

k lepším výsledkům, avšak také k vyšší výpočetní náročnosti. Tento test ukazuje potřebu kompromisu mezi kvalitou řešení a výpočetním časem.

5.2.3 Graf s 100 vrcholy

Posledním a zároveň nejrozsáhlejším grafem je neorientovaný neohodnocený graf skládající se ze 100 vrcholů a 264 hran. Porovnávány byly opět pouze heuristické metody.



Obrázek 21: Neorientovaný neohodnocený graf se 100 vrcholy

Tabulka 5: Výsledky experimentu – Graf s 100 vrcholy

Algoritmus	Průměrný počet hran v cestě	Průměrný čas (s)
GA(1)	85.6	51.47
GA(2)	88.0	199.14
GA(3)	88.4	158.87
SA(1)	83.6	11.94
SA(2)	87.4	55.62
SA(3)	89.0	188.75
TS(1)	88.5	103.28
TS(2)	91.8	549.30
TS(3)	91.2	685.14

Poznámky k parametrům algoritmů:

- GA(1): populace = 50, elita = 10, mutace = 20 %, počet generací = 100, turnaj = 10
- GA(2): populace = 100, elita = 15, mutace = 20 %, počet generací = 200, turnaj = 20
- GA(3): populace = 30, elita = 15, mutace = 20 %, počet generací = 1000, turnaj = 20
- SA(1): počáteční teplota = 100, maximální počet iterací = 1000, ochlazení = 0.90
- SA(2): počáteční teplota = 100, maximální počet iterací = 1000, ochlazení = 0.99
- SA(3): počáteční teplota = 10000, maximální počet iterací = 10000, ochlazení = 0.99
- TS(1): velikost tabu seznamu = 20, počet iterací = 20
- TS(2): velikost tabu seznamu = 50, počet iterací = 100
- TS(3): velikost tabu seznamu = 200, počet iterací = 150

Na největším grafu s 100 vrcholy se potvrzuje, že heuristické algoritmy jsou jedinou reálně použitelnou metodou. Výsledky naznačují, že delší běhy s vhodnou konfigurací parametrů

přinášejí lepší kvalitu cest, ale za cenu významně vyšších časových nároků. Výběr vhodné konfigurace tak závisí na požadavcích na přesnost řešení a dostupném výpočetním čase.

5.3 Výpočetní prostředí

Všechny experimenty byly spuštěny na jednom fyzickém stroji s následující konfigurací:

- Procesor: AMD Ryzen 5 3600 (6 jader, 12 vláken, 3.60 GHz)
- Operační paměť: 16 GB DDR4 3200 MHz

6 NÁVOD K POUŽITÍ APLIKACE

Aplikace s názvem *Longest Path Solver* je vytvořena v prostředí Java s využitím knihovny JavaFX a slouží k načítání, vizualizaci a analýze grafových struktur s cílem hledání nejdelší cesty mezi dvěma uzly. Celá aplikace je navržena jako jedno hlavní okno s důrazem na jednoduchost a intuitivní ovládání, přičemž všechny hlavní funkce jsou dostupné skrze tři základní nabídky v horní liště: *File*, *Algorithm* a *Info*.

6.1 Prerekvizity pro spuštění aplikace

Aplikaci je možné spustit dvěma způsoby. První, jednodušší možností, je stáhnout a rozbalit archiv *app.zip*, který obsahuje všechny potřebné soubory včetně přibaleného JRE. Po rozbalení stačí aplikaci spustit pomocí souboru *app.bat*, který se nachází v podadresáři *bin*.

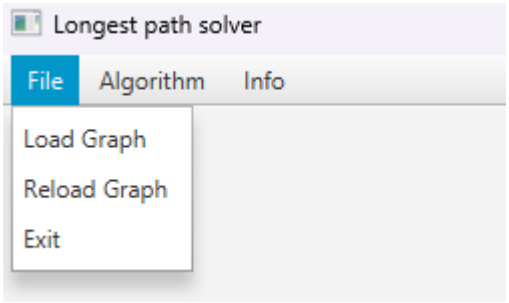
Druhý způsob spočívá ve spuštění přiloženého souboru *LongestPathSolver.jar*. Tento přístup je vhodnější pro zkušenější uživatele, kteří mají v systému nainstalovanou Javu ve verzi 23. Spuštění na jiných verzích nebylo testováno – na nižších verzích aplikace nefunguje a na vyšších verzích není funkčnost zaručena. Pokud se aplikace nespustí dvojklikem, je možné, že operační systém nemá správně přiřazené *.jar* soubory k Javě. V takovém případě lze aplikaci spustit ručně přes příkazový řádek pomocí příkazu: *java -jar LongestPathSolver.jar*

6.2 Hlavního okno a struktura menu

Aplikace je řešena jako jedno hlavní okno, které se skládá ze tří základních komponent:

- Horní menu – obsahuje tři hlavní položky: *File*, *Algorithm* a *Info*.
- Pracovní plocha – centrální část obrazovky, na které je vykreslován graf. Zde probíhá interakce s uživatelem – označování uzlů, přibližování a posun.
- Dialogová okna – slouží k výběru algoritmu, zobrazení výsledků nebo nastavení parametrů.

Každá nabídka v horní liště obsahuje konkrétní funkce důležité pro práci s aplikací, a to v logickém pořadí, jakým uživatel postupuje při analýze grafu.

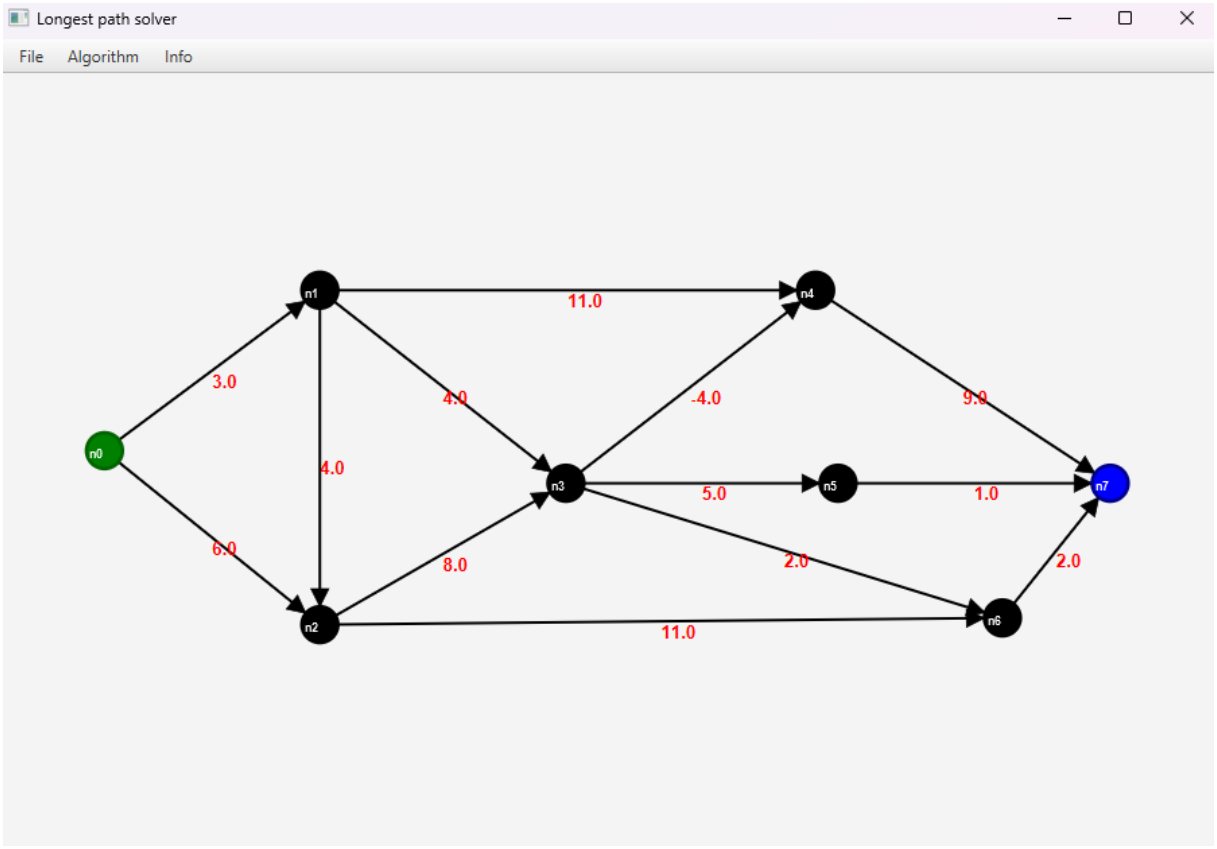


Obrázek 22: Menu aplikace

6.2.1 Ovládání grafu

Jedním z klíčových prvků aplikace je interaktivita při práci s grafem. Díky implementaci intuitivního ovládání pomocí myši je práce s grafem přirozená a připomíná práci s běžnými mapovými aplikacemi:

- Přibližování a oddalování grafu probíhá plynule pomocí kolečka myši. Uživatel si tak může snadno přiblížit konkrétní část grafu (např. stanici či uzel), nebo naopak získat celkový přehled o struktuře grafu. Přiblížení se provádí relativně ke kurzoru, což výrazně zvyšuje komfort práce.
- Posun celé pracovní plochy je realizován pomocí prostředního tlačítka myši – tedy stiskem a držením kolečka. Tato funkce je důležitá zejména při práci s rozsáhlými grafy, kde je potřeba plynule se pohybovat po různých částech.
- Přesun jednotlivých vrcholů lze provést tažením levým tlačítkem myši. Tímto způsobem si uživatel může přeuspořádat vizuální zobrazení grafu tak, aby odpovídalo například logickému nebo geografickému rozmístění uzlů.
- Označování vrcholů se provádí pravým tlačítkem myši – první klik označí počáteční vrchol, druhý klik cílový. Označené uzly se zvýrazní, aby bylo jasně viditelné, na kterých bude spuštěn algoritmus pro hledání nejdelší cesty.



Obrázek 23: Přehled hlavního okna aplikace s vykresleným grafem a zvýrazněnými označenými uzly

6.2.2 Menu File – práce se vstupními daty

Sekce *File* je určena pro práci se vstupními daty a základní operace nad grafem. Obsahuje následující funkce:

- *Load Graph* – otevře dialogové okno, ve kterém si uživatel zvolí soubor typu *.xml*, který obsahuje informace o daném grafu. Aplikace podporuje dva formáty:
    - MesoRail – vstupní data přímo ze softwaru pro železniční infrastrukturu. V souboru jsou kromě uzlů a hran uloženy také jejich reálné souřadnice, identifikátory a délky hran, které odpovídají délce kolejí. V tomto případě je vyžadováno, aby název vstupního souboru obsahoval řetězec *Railway* – např. *MyNetwork_Railway.xml*.
    - yEd GraphML – soubory určené pro obecné grafové struktury, vytvořené pomocí grafického editoru *yEd Graph Edit*or. Jedná se o XML dokumenty, jejichž struktura odpovídá standardu GraphML. I přesto, že mají příponu *.graphml*, jsou z pohledu struktury plně kompatibilní s XML a lze je editovat i ručně. Obsahují uzly, hrany, případně atributy jako orientace nebo váha hran.

Tento formát je vhodný pro testování, srovnávání algoritmů a tvorbu teoretických modelů.

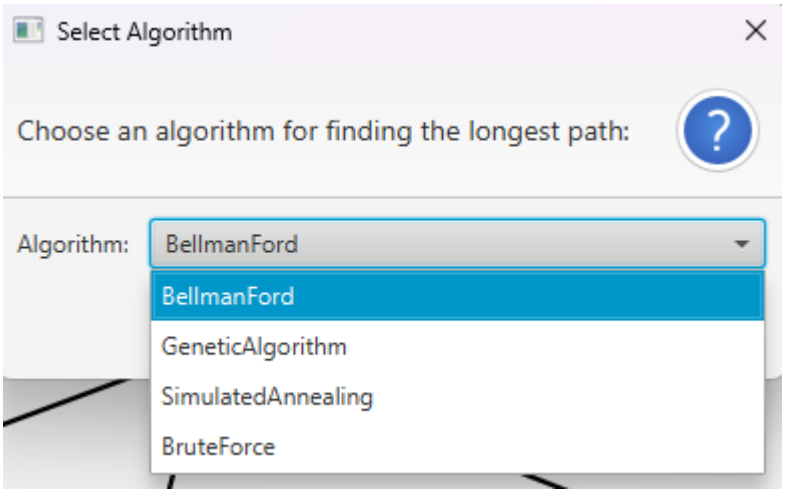
Při načtení dat dochází k jejich internímu zpracování a převedení do struktury vhodné pro výpočty – konkrétně do objektové reprezentace *Graph*, která obsahuje seznam uzlů a hran.

- *Reload Graph* – resetuje aktuální zobrazení grafu (pozice, přiblížení) a umožňuje návrat do výchozího stavu. To je užitečné např. po větším posunu uzlů.
- *Exit* – ukončí běh aplikace.

6.2.3 Menu Algorithm – volba výpočtu nejdelší cesty

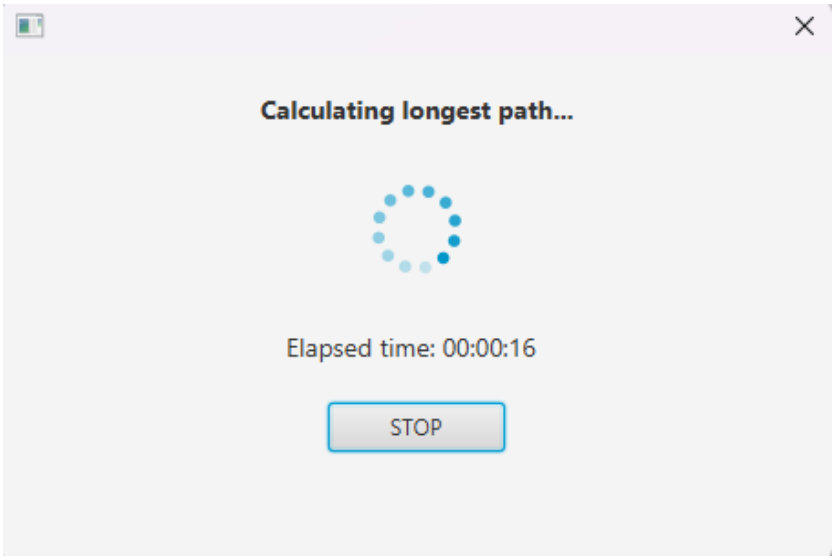
Tato sekce je středobodem celé aplikace. Po výběru dvou vrcholů uživatel zvolí volbu *Find Longest Path*, která otevře dialogové okno s výběrem algoritmu. K dispozici jsou:

- DAG – v nabídce pouze pro acyklické orientované grafy
- Brute Force
- Bellman-Ford
- GA – Vstupní parametry lze upravit v rozšířeném dialogu:
    - velikost populace,
    - počet generací,
    - pravděpodobnost mutace,
    - počet elitních jedinců,
    - počet jedinců pro turnajový výběr.
- SA – Vstupní parametry lze upravit v rozšířeném dialogu:
    - počáteční teplotu,
    - ochlazovací koeficient,
    - maximální počet iterací.
- TS – Vstupní parametry lze upravit v rozšířeném dialogu:
    - maximální počet iterací,
    - velikost tabu seznamu.



Obrázek 24: Dialogové okno pro výběr algoritmu

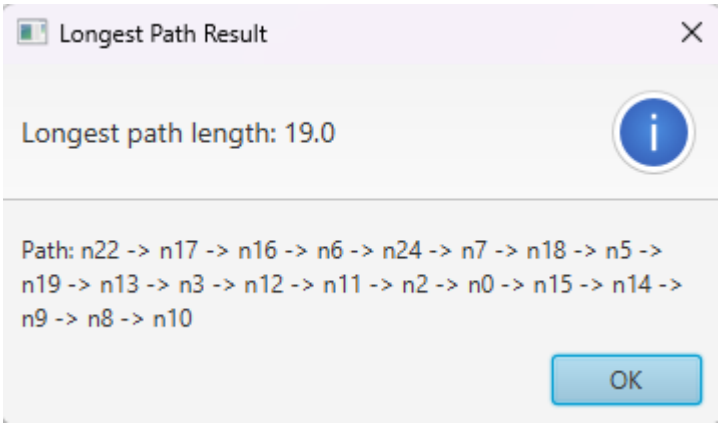
Provádění některých výpočtů může být značně náročné. Po spuštění výpočtu se zobrazí dialogové okno s informací o uplynulém čase. Výpočet lze kdykoliv přerušit pomocí tlačítka *Stop*, nebo je možné vyčkat na jeho dokončení. V případě úspěšného dokončení se v dialogovém okně zobrazí oznámení o ukončení výpočtu a časomíra se zastaví.



Obrázek 25: Dialogové okno s uplynulým časem hledání nejdelší cesty

Po provedení výpočtu a zvolení poslední z možností tohoto menu – *Results* se v dialogovém okně zobrazí výsledky, které obsahují:

- seznam vrcholů v nejdelší nalezené cestě (zobrazeno názvem nebo ID),
- celkovou délku cesty nebo počet kroků (hran) v cestě.

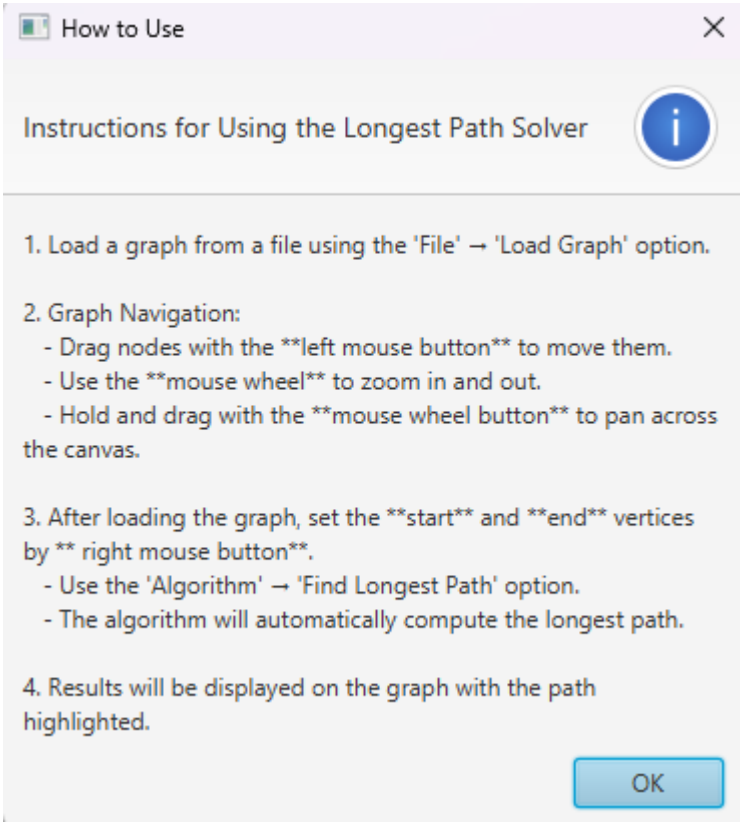


Obrázek 26: Dialogové okno s podrobnostmi o nalezené cestě

6.2.4 Menu Info – uživatelská nápověda

Sekce *Info* obsahuje položku *How to Use*, která otevře dialog s krokovým návodem, jak aplikaci používat. Návod zahrnuje:

- výběr vstupního grafu,
- pohyb po ploše a manipulace s grafem,
- označení uzlů a výběr algoritmu,
- zobrazení výsledků.



Obrázek 27: Dialogové okno s nápovědou použití aplikace

7 PROGRAMOVÁ ČÁST

V následujících podkapitolách je popsána struktura systému, včetně jednotlivých balíčků, jejich odpovědnosti a klíčových tříd.

7.1 Struktura

Pro dosažení větší čitelnosti, udržitelnosti a rozšiřitelnosti byl návrh a implementace systému rozdělen do logicky oddělených balíčků. Každý balíček má jasně vymezenou odpovědnost a zapadá do celkového architektonického návrhu aplikace. V následujících podkapitolách jsou jednotlivé balíčky stručně popsány včetně jejich hlavních tříd a funkcionality.

7.1.1 Balíček algorithms

Tento balíček obsahuje jednotlivé implementace algoritmů pro hledání nejdelších cest v grafu.

Zahrnuje následující třídy:

- BellmanFordLongestPath
- BruteForceLongestPath
- DAGLongestPath
- GALongestPath
- SALongestPath
- TSLongestPath

Tyto třídy není třeba podrobněji popisovat, jelikož jejich výstižné názvy reflektují, k čemu slouží.

7.1.2 Balíček controllers

Obsahuje třídy zajišťující uživatelské rozhraní a obsluhu vstupních parametrů pro jednotlivé algoritmy:

- GAConfigController – grafické rozhraní pro konfiguraci parametrů GA (počet generací, velikost populace, míra mutace, elitismus, turnajový výběr).
- SAConfigController – grafické rozhraní pro konfiguraci parametrů SA (počáteční teplota, koeficient ochlazování, maximální počet iterací).
- TSConfigController – grafické rozhraní pro konfiguraci parametrů TS (maximální počet iterací, velikost tabu seznamu).

- MainController – hlavní kontroler aplikace, který integruje všechny funkcionality, včetně výběru algoritmu, načítání grafu, zoomování, posouvání a obsluhy interakcí uživatele.

7.1.3 Balíček parsers

Balíček pro parsování vstupních datových struktur z různých zdrojů:

- RailwayXMLParser – parser pro železniční grafy ve formátu používaném systémem *MesoRail*.
- YedXMLParser – parser pro grafy vytvořené v editoru *yEd*, sloužící pro obecné testování algoritmů.

7.1.4 Balíček utils

Nástroje a pomocné třídy:

- Configuration – uchovává výchozí parametry GA, SA a TS. Tyto hodnoty jsou přepisovány při spuštění na základě vstupu z GUI.
- PathUtil – zajišťuje hledání cest pro GA, SA a TS.
- RailwayEdgeUtil – zabraňuje nesprávnému průchodu vlaku po kolejových spojkách typu *single-cross* a *double-cross*.

7.1.5 Balíček view

Balíček zaměřený na vizualizaci grafu:

- VertexView, EdgeView, GraphView – třídy pro grafické zobrazení vrcholů a hran. Řeší vykreslování barev, tloušťky čar, popisků, resetování stylů apod.

7.1.6 FXML soubory

Součástí zdrojového kódu jsou také soubory s příponou *.fxml*, které definují grafické rozvržení uživatelského rozhraní. Konkrétně se jedná o hlavní okno aplikace, konfigurační dialogy pro jednotlivé metaheuristické metody a dialogové okno zobrazující průběh výpočtu algoritmu.

8 POUŽITÉ TECHNOLOGIE A NÁSTROJE

Při vývoji aplikace a práci s daty byly využity následující technologie a nástroje. Výběr byl ovlivněn osobními preferencemi a zkušenostmi, což umožnilo rychlejší a pohodlnější vývoj.

8.1 Java a JavaFX

Aplikace byla naprogramována v jazyce Java (verze 23), a to hlavně kvůli autorově dobré znalosti tohoto jazyka. Přestože třeba C++ může být výkonnější, Java nabídla lepší čitelnost, jednoduchost při práci s datovými strukturami a širokou podporu knihoven, což pro potřeby této práce bohatě stačilo.

Pro uživatelské rozhraní byla použita JavaFX (verze 23), která umožňuje vytvářet moderní GUI a zároveň oddělit logiku od vzhledu aplikace pomocí FXML.

8.2 SceneBuilder

SceneBuilder sloužil k návrhu grafického rozhraní. Díky němu nebylo potřeba psát FXML ručně – rozhraní se dalo poskládat vizuálně. To nejen urychlilo práci, ale i snížilo pravděpodobnost chyb v kódu.

8.3 IntelijIdea

Celý vývoj probíhal v IntelliJ IDEA (verze Ultimate) – oblíbeném IDE s velmi dobrou podporou pro Java i JavaFX. Nabízí chytré doplňování kódu, ladění, správu verzí a další funkce, které výrazně zjednodušují práci. Z autorova pohledu jde o jedno z nejpohodlnějších vývojových prostředí pro Javu.

8.4 yEd Graph Editor

Pro ruční návrh testovacích grafů byl použit yEd Graph Editor – jednoduchý nástroj na kreslení grafových struktur. Umožnil snadné vytváření a přehledné vizualizace dat, která se pak použila k ladění a testování algoritmů.

8.5 Mesorail

Mesorail je software pro návrh a simulaci železničních sítí. V rámci práce byl použit model železnice, který byl k dispozici právě v tomto nástroji. Mesorail pomohl ověřit, že převod reálné železniční topologie do vlastní aplikace odpovídá skutečnosti. Ostatní funkce (například simulace provozu) využity nebyly.

9 ZÁVĚR

Cílem této diplomové práce bylo prozkoumat problematiku hledání nejdelší jednoduché cesty v grafu a ověřit využitelnost různých přístupů při řešení tohoto úkolu, zejména v kontextu železniční infrastruktury. Jelikož se jedná o výpočetně náročný problém (NP-těžký), byla práce zaměřena především na využití heuristických a metaheuristických metod, které mohou při vhodném nastavení nabídnout dostatečně kvalitní řešení v rozumném čase.

V rámci práce bylo implementováno několik algoritmů, které jsou vhodné pro různé typy grafů, přičemž největší prostor dostaly čtyři z nich – Brute Force, Genetický algoritmus, Simulované žíhání a Tabu Search. Tyto algoritmy byly nejprve otestovány na konkrétním železničním modelu Pardubického hlavního nádraží, přičemž zde bylo zjištěno, že díky omezením zde není tolik cest, a proto je nejvíce vhodný právě BF. Metaheuristiky však také dosahovaly dostatečně kvalitních řešení v nízkém čase.

V další fázi experimentů byly tyto vybrané algoritmy aplikovány na obecných grafech různých velikostí a hustoty. Výsledky ukázaly, že i přes jednoduchost základního Brute Force přístupu se s narůstající velikostí grafu velmi rychle stává nepoužitelným. Oproti tomu metaheuristické algoritmy prokázaly, že jsou schopné nalézt velmi kvalitní řešení i v případech, kde by exaktní přístupy selhávaly z hlediska výpočetního času.

Z provedených experimentů vyplývá, že nejlepších výsledků v průměru dosahoval algoritmus Tabu Search, přestože jeho výpočetní náročnost byla vyšší než u ostatních metod. Genetický algoritmus prokázal, že při správném nastavení parametrů (velikost populace, počet generací) je schopen nalézt řešení blízká optimu. Simulované žíhání bylo v testech konzistentní a často nabízelo dobrý kompromis mezi kvalitou výsledku a časovou náročností.

Součástí práce byla i tvorba vlastní desktopové aplikace, která umožňuje snadno zadat strukturu grafu a spustit výpočty pomocí výše zmíněných algoritmů. Aplikace je navržena tak, aby ji bylo možné případně dále rozšířit, např. o nové typy metaheuristik, dynamické změny grafu nebo případně i o další funkcionality.

Z pohledu železniční dopravy se ukazuje, že algoritmické hledání nejdelších (a nesouběžných) cest může být užitečné například při plánování provozu nebo návrhu železniční infrastruktury. Výsledky této práce mohou posloužit i jako základ pro další vývoj v oblasti optimalizace dopravních procesů i mimo akademickou sféru.

POUŽITÁ LITERATURA

[1] KESHAVARZ-KOHJERDI, Fatemeh; BAGHERI, Alireza a ASGHARIAN-SARDROUD, Asghar. A linear-time algorithm for the longest path problem in rectangular grid graphs. Online. *Discrete Applied Mathematics*. 29. 9. 2011, č. 160, s. 210-217. ISSN 0166-218X. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0166218X11003088#br000010. [cit. 2025-05-16].

[2] KOVÁŘ, Petr. *Úvod do Teorie grafů.* Online. Ostrava, 2014. Příprava přednášek a cvičení. Vysoká škola báňská – Technická univerzita Ostrava a Západočeská univerzita v Plzni. Dostupné z: https://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/uvod_do_teorie_grafu.pdf. [cit. 2025-05-06].

[3] MATOUŠEK, Jiří. *Kapitoly z diskrétní matematiky.* Online. Praha: Karolinum, 2002. ISBN 80-246-0084-6. Dostupné z: https://lms.umb.sk/pluginfile.php/58452/mod_lesson/page_contents/9220/Nesetril.pdf. [cit. 2025-05-06].

[4] *Teorie grafů*. Online. Ostrava: Vysoká škola Báňská, 21. 6. 2006. Dostupné z: http://books.fs.vsb.cz/SystAnal/texty/21.htm. [cit. 2025-05-01].

[5] RAK, Josef. *Teorie grafů*. Univerzita Pardubice, c2011-2020.

[6] MOCAS, Sarah Easton. *SEPARATING EXPONENTIAL TIME CLASSES FROM POLYNOMIAL TIME CLASSES*. Online, Disertační práce. Boston: NORTHEASTERN UNIVERSITY, 27. 8. 1993. Dostupné z: https://www.proquest.com/openview/7975a13f39ef75802e496af5bf3d34aa/1?cbl=18750&diss=y&pq-origsite=gscholar#. [cit. 2025-05-16].

[7] NASEERA, Shaik. *P, NP, NP-Hard & NP-complete problems*. Kalikiri: Department of CSE, JNTUACEK Online, prezentace. [2017]. Dostupné z: https://www.jntua.ac.in/gate-online-classes/registration/downloads/material/a159262902029.pdf. [cit. 2025-05-16].

[8] TUTORIALSPOINT. P and NP Class. *Design and Analysis of Algorithms*. Online. Hyderabad: Tutorials Point, 3. 6. 2017. Dostupné z: https://www.tutorialspoint.com/design_and_analysis_of_algorithms/p_and_np_class.htm. [cit. 2025-05-14].

[9] MONTEIRO, Jucemar. *Diagram of intersection among classes P, NP, NP-complete and NP-hard problems*. Online. In: ResearchGate. 2019. Dostupné z: https://www.researchgate.net/figure/Diagram-of-intersection-among-classes-P-NP-NP-complete-and-NP-hard-problems_fig13_336890186. [cit. 2025-05-16].

[10] KNOBLOCK, Craig A. *Abstracting the Tower of Hanoi*. Pittsburgh: School of Computer Science, Carnegie Mellon University, 1990. Technická zpráva. Online. Dostupné z: https://usc-isi-i2.github.io/papers/knoblock90-hanoi.pdf. [cit. 2025-05-14].

[11] SHE, Adrian. *Hamiltonian Path is NP-Complete*. Toronto: University of Toronto, CSC 463. Online. 5. 3. 2020. Dostupné z: http://www.cs.toronto.edu/~ashe/ham-path-notes.pdf. [cit. 2025-05-16].

[12] SOROUDI, Alireza. *The Shortest/Longest Path in a Graph*. LinkedIn. Online. Dublin, 11. 1. 2023. Dostupné z: https://www.linkedin.com/pulse/shortestlongest-path-graph-alireza-soroudi. [cit. 2025-05-16].

[13] DEY, Abhishek. *Parallel Jobs Scheduling Problem with Precedence*. Online. Seattle, c2025. Dostupné z: https://www.thealgorists.com/Algo/JobScheduling/ParallelJobScheduling. [cit. 2025-05-16].

[14] KHAN, Mumit. *Longest path in a directed acyclic graph (DAG)*. CSE 221, 10. 4. 2011. Online. Dostupné z: https://blogs.asarkar.com/assets/docs/algorithms/Longest%20Path%20in%20a%20DAG%20-%20Khan.pdf. [cit. 2025-05-16].

[15] PEARCE, David J. a KELLY, Paul H. J. *A Dynamic Topological Sort Algorithm for Directed Acyclic Graphs*. ACM Journal of Experimental Algorithmics, 2006, roč. 11, článek č. 1.7, s. 1–24. Online. Dostupné z: https://www.doc.ic.ac.uk/~phjk/Publications/DynamicTopoSortAlg-JEA-07.pdf. [cit. 2025-05-16].

[16] *Bellman-Ford Algorithm*. Online. cp-algorithms.com, c2014–2025 cp-algorithms contributors, 12. 10. 2024. Dostupné z: https://cp-algorithms.com/graph/bellman_ford.html. [cit. 2025-05-16].

[17] *Bellman-Ford*. SESV Tutorial. Online. 2. 1. 2023. Dostupné z: https://www.sesvtutorial.com/bellman-ford/#longest-path-with-bellman-ford-algorithm. [cit. 2025-05-16].

[18] JAVAID, Muhammad Adeel. *Understanding Dijkstra's Algorithm*. 2013, 27 stran. Institute of Electrical and Electronics Engineers (IEEE). Dostupné z: http://dx.doi.org/10.2139/ssrn.2340905. [cit. 2025-05-16].

[19] *Dijkstra Algorithm*. Online. cp-algorithms.com, c2014–2025 cp-algorithms contributors, 24. 7. 2023. Dostupné z: https://cp-algorithms.com/graph/dijkstra.html. [cit. 2025-05-16].

[20] DEY, Abhishek. *Longest Paths Algorithm*. Online. Seattle, c2025. Dostupné z: https://www.thealgorists.com/Algo/LongestPaths. [cit. 2025-05-16].

[21] SRIYANI, Violina. *Analysis of Brute Force and Branch & Bound Algorithms to solve the Traveling Salesperson Problem (TSP)*. Turkish Journal of Computer and Mathematics Education. Online. 20. 4. 2021, vol. 12, č. 8, s. 1226–1229. Dostupné z: https://www.proquest.com/openview/48388af438d35d576a0998dc22fa3478/1?cbl=2045096&pq-origsite=gscholar. [cit. 2025-05-16].

[22] *What Are Metaheuristic Methods? – Applications in Optimization*. Eurystic Solutions. Online. 1. 3. 2025. Dostupné z: https://eurysticsolutions.com/2025/03/01/what-are-metaheuristic-methods-applications-in-optimization/. [cit. 2025-05-16].

[23] ZHAO, Jiao, HUI HU, Yi HAN a Yao CAI. *A review of unmanned vehicle distribution optimization models and algorithms*. Journal of Traffic and Transportation Engineering (English Edition). Online. 2023, vol. 10, č. 4, s. 548–559. ISSN 2095-7564. DOI: 10.1016/j.jtte.2023.07.002. Dostupné z: https://www.sciencedirect.com/science/article/pii/S209575642300079X. [cit. 2025-05-16].

[24] HOLLAND, John H**.** *Genetic Algorithms: Computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand*. Online. *Scientific American*. 1992, vol. 267, č. 1, s. 66–73 [cit. 2025-05-16]. Dostupné z: https://www.jstor.org/stable/24939139. [cit. 2025-05-16].

[25] WIDMER, Marino, HERTZ, Alain a COSTA, Daniel. *Metaheuristics and scheduling.* Online. In: Scheduling: Theory and Applications. Kapitola 3, s. 36–70. Dostupné z: https://www.gerad.ca/~alainh/Scheduling.pdf [cit. 2025-05-16].

[26] LAARHOVEN, P. J. M. van a AARTS, E. H. L**.** *Simulated annealing.* Online. In: *Simulated Annealing: Theory and Applications*. Dordrecht: Springer, 1987. Mathematics and Its Applications, vol. 37. ISBN 978-94-015-7744-1. Dostupné z: https://doi.org/10.1007/978-94-015-7744-1_2. [cit. 2025-05-16].

[27] SCHOLVIN, John Kenneth. *Approximating the longest path problem with heuristics: a survey*. Online, Diplomová práce. University of Illinois at Chicago, 1999. Dostupné z: https://scholvin.com/thesis.pdf. [cit. 2025-05-16].

10 PŘÍLOHY

1) Zdrojový kód aplikace
2) Spustitelný soubor aplikace
3) Model železniční infrastruktury a grafové modely