Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Vyhledávání souběžných a nejdelších jízdních cest v modelech kolejové železniční infrastruktury

Bc. Martin Kopecký

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. Martin Kopecký**
Osobní číslo:	**I23278**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Vyhledávání souběžných a nejdelších jízdních cest v modelech kolejové železniční infrastruktury**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

V úvodní části práce je nutné představit vhodné modely železniční kolejové infrastruktury, které jsou reprezentovány různými typy hranově nebo vrcholově ohodnocených grafů.
Primárním cílem diplomové práce je návrh a ověření algoritmů pro vyhledávání souběžných jízdních cest a nejdelších jízdních cest v modelech/grafech odrážejících zkoumanou část železniční kolejové infrastruktury.
Nalezeným souběžným jízdním cestám, kterým odpovídají vzájemně disjunktní cesty v grafech, odpovídají možnosti jízd vlaků bez vzájemných interakcí ve zkoumaném kolejišti. Délka vyhledané nejdelší přípustné jízdní cesty vyjadřuje skutečnost, jaký nejdelší vlak lze umístit v rámci vyšetřované části železniční infrastruktury. Při vyhledávání uvedených jízdních cest jsou v uplatňovány specifické omezující podmínky, které odrážejí technické a technologické možnosti jízd vlaků na kolejové infrastruktuře.
Navržené algoritmy budou ověřovány na vhodné případové studii využívající modely odlišných kolejových infrastruktur.

Rozsah pracovní zprávy: **50**
Rozsah grafických prací: **–**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

VESELY, Petr, Antonin KAVICKA a Pavel KRYZE. Automated Construction of Mesoscopic Railway Infrastructure Models Supporting Station Throat Capacity Assessment. IEEE Access. 2023, 11, 37869--37899. ISSN 2169-3536. doi:10.1109/ACCESS.2023.3266813.
VOLEK, Josef a LINDA, Bohdan. *Teorie grafů – aplikace v dopravě a veřejné správě*. Pardubice: Univerzita Pardubice, 2012. ISBN 978-80-7395-225-9.

Vedoucí diplomové práce: **prof. Ing. Antonín Kavička, Ph.D.**
Katedra softwarových technologií

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:

Práci s názvem Vyhledávání souběžných a nejdelších jízdních cest v modelech kolejové železniční infrastruktury 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., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, 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. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 18. 5. 2025

Martin Kopecký v. r.

PODĚKOVÁNÍ

Tímto bych rád poděkoval prof. Ing. Antonínu Kavičkovi, Ph.D. za cenné rady a pomoc při zpracování této práce a za jeho trpělivost během konzultací.

ANOTACE

Tato práce se zabývá vyhledáváním souběžných a nejdelších jízdních cest v modelech kolejové železniční infrastruktury. Práce je rozdělena na dvě části. První část je zaměřena na popis modelu kolejové železniční infrastruktury a základních principů hledání souběžných a nejdelších cest. Dále jsou popsány existující metody a algoritmy pro řešení uvedených úloh a jejich uplatnění v praxi. Druhá část je zaměřena na prezentaci praktického řešení. Popisuje samotný nový algoritmus, jeho implementaci, uživatelské rozhraní a aplikaci na reálných modelech železniční infrastruktury. Získané výsledky jsou následně porovnány.

KLÍČOVÁ SLOVA

Model infrastruktury, souběžné jízdní cesty, nejdelší jízdní cesta, algoritmy hledání cest

TITLE

Searching for simultaneous and longest track routes within railway infrastructure models

ANNOTATION

This thesis deals with the search for simultaneous and longest paths in railway infrastructure models. The work is divided into two parts. The first part focuses on the description of the railway infrastructure model and the basic principles of finding simultaneous and longest paths. Then the existing methods and algorithms for solving these tasks and their application in practice are described. The second part is focused on the presentation of the practical solution. It describes the new algorithm itself, its implementation, user interface and application on real models of railway infrastructure. The obtained results are then compared.

KEYWORDS

Railway infrastructure, simultaneous routes, longest route, route-finding algorithms

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

DFS	Depth-First Serach
BFS	Breadth-First Serach
DAG	Directed Acyclic Graph
LIFO	Last In First Out
FIFO	First In First Out
GPS	Global Positioning System
RIP	Routing Information Protocol
OSPF	Open Shortest Path First
IS-IS	Intermediate System to Intermediate System
NPC	Non-Player Character
IDE	Integrated Development Environment
UI	User Interface
XML	Extensible Markup Language
CSV	Comma-Separated Values
API	Application Programming Interface

ÚVOD

Kolejová železniční infrastruktura představuje klíčovou součást moderní dopravy, která zajišťuje efektivní a spolehlivou přepravu osob i zboží. Díky své schopnosti zvládat vysokou kapacitu přepravy je železniční doprava klíčovým prvkem v propojení regionů a podporuje rozvoj ekonomiky. S rostoucími nároky na železniční dopravu, zahrnujícími vyšší objem přepravovaného zboží, častější spoje a přesnější časové harmonogramy, je nezbytné zlepšovat metody plánování a řízení provozu. Tyto požadavky zahrnují nejen optimalizaci cest pro maximální využití kapacity dostupné infrastruktury, ale také minimalizaci rizika konfliktů a zpoždění vlaků na vytížených úsecích a tím zajistit plynulý provoz v rámci složitých železničních sítí.

Problémy, které se v této oblasti řeší, jsou nalezení souběžných a nejdelších jízdních cest v modelech železniční infrastruktury. Řešení těchto úloh je cílem této práce, a to návrh a implementace algoritmu, který bude schopen efektivně a přesně hledat souběžné a nejdelší jízdní cesty v různých typech železniční infrastruktury. Jejich nalezení má široké přínosy, a to nejen pro provozovatele železniční infrastruktury, ale i pro dopravce a konečné uživatele. Nalezené cesty poté hrají zásadní roli nejen při optimalizaci provozu, ale také pro předcházení konfliktů, identifikaci kritických bodů a s tím spojené plánování investic, rozšíření, či modernizace železniční sítě s ohledem na budoucí požadavky.

Práce je rozdělena na teoretickou a praktickou část. Teoretická část se zaměřuje na popis modelu kolejové železniční infrastruktury, včetně základních pojmů a terminologie v této oblasti. Dále se věnuje vysvětlení principů vyhledávání souběžných a nejdelších jízdních cest, přehledu existujících algoritmů a jejich srovnání z hlediska efektivity a využití v praxi.

Na teoretickou část navazuje část praktická, která zahrnuje konkrétní návrh a implementaci nového algoritmu. Tato část začíná popisem technologického prostředí a nástrojů, které byly využity při vývoji řešení. Následně je popsána struktura programu, zahrnující jak základní logiku algoritmu, tak i implementaci uživatelského rozhraní pro snadnou manipulaci a testování. Algoritmus je poté aplikován na reálný model železniční infrastruktury a jeho alternativní scénáře.

Práce je zakončena analýzou získaných výsledků, shrnuje hlavní poznatky, zhodnocuje přínos této práce pro plánování jízdních cest v železniční dopravě a diskutuje možné směry dalšího výzkumu a rozvoje.

MOTIVACE A CÍLE

Pro efektivní řízení železničního provozu je identifikace souběžných a nejdelších jízdních cest jedním z problémů k řešení při optimalizaci využití tratí, prevenci konfliktů v provozu a strategickém plánování dopravy. Obzvláště je to důležité ve složitých železničních sítích s vysokou hustotou provozu, kde je potřeba maximálně využít dostupnou kapacitu sítě a zároveň minimalizovat zpoždění vlaků při průjezdu železniční sítí.

Cílem této práce je vytvoření aplikace pro vyhledávání souběžných a nejdelších jízdních cest v modelech kolejové železniční infrastruktury a možnost jejich následné vizualizace. Modelování železniční infrastruktury je možné realizovat různými způsoby na základě toho, jaké požadavky musí model splňovat, aby byl vhodný pro konkrétní typ analýzy. Pro účely této práce je model reprezentován jako neorientovaný graf, což umožnuje efektivní analýzu možných jízdních cest.

V praxi je kromě textového výstupu často požadována také vizualizace nalezených souběžných a nejdelších cest, která umožňuje jejich lepší prezentaci. To je řešeno funkcí, která provede vizualizaci vybraných cest přímo v modelu infrastruktury. Tím se získá jasná představa kudy jednotlivé cesty vedou.

V práci jsou procesy načtení modelu, vyhledání cest a vizualizace podrobně popsány spolu s algoritmy, které jsou k tomu využity. Rovněž je zde popsána samotná aplikace, která uživateli umožňuje hledání souběžných a nejdelších cest a jejich následnou vizualizaci. Uživatel díky tomu nemusí mít žádné hluboké znalosti o použitých algoritmech nebo o technikách vizualizace.

TEORETICKÁ ČÁST

Teoretická část se zaměřuje na popis procesu vytváření modelu kolejové železniční infrastruktury, vysvětlení klíčových pojmů a popisu vlastností grafu, který reprezentuje model infrastruktury použitý v této práci. Dále se věnuje definici souběžných a nejdelších jízdních cest, problematice jejich vyhledávání a v závěru části je uveden přehled a popis existujících algoritmů využitelných při prohlídkách a hledání cest v grafech.

1 MODEL KOLEJOVÉ ŽELEZNIČNÍ INFRASTRUKTURY

Kolejová železniční infrastruktura, ze které poté vychází její model, je komplexní systém zahrnující všechny fyzické prvky potřebné k zajištění provozu vlaků. Zahrnuje tratě, stanice, výhybky, signalizační zařízení a další komponenty, které společně umožňují bezpečný a efektivní pohyb vlaků po železniční síti. Model infrastruktury by taky měl být navržen tak, aby ho bylo možné měnit, nebo rozšiřovat podle potřeb.

Model takovéto železniční infrastruktury potom vznikne z transformace dostupných mapových podkladů, které mohou být ve formě papírové dokumentace nebo elektronických souborů. U papírových podkladů se nejprve provede digitalizace, nejčastěji pomocí jejich naskenování, čímž se získají podklady pro tzv. vektorizaci, což je proces, při kterém jsou ze získaných podkladů překresleny koleje, výhybky a další prvky v prostředí specializovaného grafického editoru. U elektronických podkladů, například výkresů vytvořených v programech AutoCAD nebo MicroStation, je model infrastruktury získán přímo konverzí do formátu určeného například pro simulační nástroje. Obě metody zaručují potřebnou úroveň přesnosti a věrnosti vytvořeného modelu a není potřeba používat zjednodušené modely, kde je riziko nepřesností z přílišného zjednodušení. [1]

Z transformace podkladů vzniká fyzická úroveň modelu infrastruktury, která odráží základní podobu kolejiště. Obsahuje základní prvky, jako jsou koleje, jednoduché výhybky, křižovatkové výhybky a kolejové křižovatky a každý prvek má své jedinečné identifikační číslo, což usnadňuje jejich správu a použití. Po vytvoření fyzického modelu následuje přiřazení konkrétní funkce jednotlivým kolejím, čímž vzniká logická úroveň modelu infrastruktury. Tento krok vyžaduje znalost reálného či projektovaného kolejiště, jelikož funkce koleje určuje její roli v rámci modelu, například zda se jedná o vjezdovou, výtažnou kolej, svažný pahrbek nebo spojka pro odstup lokomotiv. Tato informace je potom zásadní při určování technologií a plánování technologických postupů, například při přesunech lokomotiv nebo sestavování vlakových souprav. [1]

Posledním krokem při vytváření modelu infrastruktury je stanovení vlakových cest. Způsob stanovení jízdních cest závisí na provedení modelu. Model může být připraven buď pro statické definování jízdních cest, které následně určují pohyb mobilních prostředků a vycházejí z reálných údajů, například z protokolů elektronických stavědel, nebo pro dynamické stanovení jízdních cest, které taky určují pohyb mobilních prostředků, nicméně se stanovení provádí podle aktuálních podmínek. Takto sestavené cesty se následně modelují pomocí editorů, a to buď autonomních, nebo těch, které jsou součástí simulačních nástrojů. [1]

V rámci popisu vytváření modelu infrastruktury je několikrát zmíněna simulace a simulační nástroje. Je to z důvodu, že model, který je zde použit, byl vytvářen výše popsaným způsobem a je určen pro simulaci. Pro účely této práce je model použit čistě jen jako model infrastruktury, a proto je použita jen fyzická úroveň modelu infrastruktury, jelikož při vyhledávání souběžných a nejdelších jízdních cest se funkce kolejí nezohledňuje a není ani potřeba seznam používaných jízdních cest.

1.1 Základní pojmy a terminologie

V rámci praktické části i v modelu infrastruktury se vyskytuje řada pojmů, které jsou důležité pro pochopení celého tématu a v této oblasti se běžně používají.

1.1.1 Výhybka

Výhybka je zařízení v místě, kde se koleje schází či rozcházejí a umožňují přechod kolejových vozidel z jedné koleje na druhou bez přerušení jízdy. [2]

V modelu se vyskytuje několik typů výhybek, které mají odlišný počet zaústěných směrů, odlišné způsoby průjezdu a liší se i značení výhybek, kdy značení ve schématu modelu je odlišné od toho, jak je výhybka zanesena ve finálním modelu grafu.

1.1.1.1 Jednoduchá výhybka

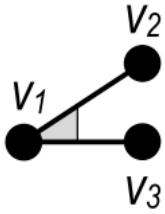
Prvním takovým typem je jednoduchá výhybka, která má dvě větve, přičemž jedna větev je přímá. Výhybku potom můžeme dále dělit na pravou a levou, podle toho, na jaké straně je příslušná odbočka při jízdě proti hrotu výhybky. [2]

Výhybka je ve schématu infrastruktury značena buď jako V (Obrázek 1), nebo jako Y (Obrázek 2).

Obrázek 1: Jednoduchá výhybka ve schématu ve tvaru V [4]

Obrázek 2: Jednoduchá výhybka ve schématu ve tvaru Y [4]

Znázornění výhybky v modelu grafu je na Obrázku 3.



Obrázek 3: Jednoduchá výhybka v grafu [4]

Je zde ovšem konstrukční omezení pro průjezd výhybkou, které se vztahuje ke směru příjezdu na výhybku. Symbolický trojúhelník u vrcholu V_1 udává, že pokud kolejové vozidlo jede z vrcholu V_2, nemůže na výhybce pokračovat do vrcholu V_3, ale pouze po kolejovém segmentu, který vede z V_1. To platí i při jízdě z vrcholu V_3. Pouze z V_1 je možné použít oba kolejové segmenty, které jsou modelovány jako hrany, za předpokladu, že směr příjezdu do V_1 není z V_2 ani V_3.

Všechny povolené a zakázané průjezdy jednoduchou výhybkou, která je na Obrázku 3, jsou uvedeny v Tabulce 1.

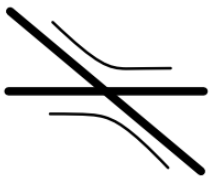
Tabulka 1: Povolené a zakázané průjezdy jednoduchou výhybkou [Zdroj vlastní]

Povolené průjezdy	Zakázané průjezdy
$V_1 \rightarrow V_3$ $V_1 \rightarrow V_2$ $V_2 \rightarrow V_1$ $V_3 \rightarrow V_1$	$V_2 \rightarrow V_1 \rightarrow V_3$ $V_3 \rightarrow V_1 \rightarrow V_2$

1.1.1.2 Plná křižovatková výhybka

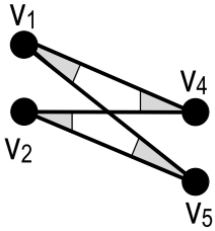
U této výhybky, které se hovorově říká úplný Angličan, dochází ke křížení čtyř zaústěných směrů, což umožňuje vedení kolejových vozidel do protilehlého směru jak přímým, tak odbočným směrem.[3]

Značení výhybky ve schématu je na Obrázku 4.



Obrázek 4: Plná křižovatková výhybka ve schématu [4]

Znázornění výhybky v modelu grafu je na Obrázku 5.



Obrázek 5: Plná křižovatková výhybka v grafu [4]

Stejně jako u jednoduché výhybky, i zde je konstrukční omezení pro průjezd výhybkou podle směru příjezdu na výhybku, které je označeno symbolickým trojúhelníkem. To znamená, že například při cestě z V_5 do V_1 nelze poté pokračovat do V_4.

Všechny povolené a zakázané průjezdy plnou křižovatkovou výhybkou na Obrázku 5. jsou uvedeny v Tabulce 2.

Tabulka 2: Povolené a zakázané průjezdy plnou křižovatkovou výhybkou [Zdroj vlastní]

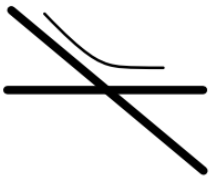
Povolené průjezdy	Zakázané průjezdy
$V_1 \rightarrow V_4$	$V_1 \rightarrow V_4 \rightarrow V_2$
$V_1 \rightarrow V_5$	$V_1 \rightarrow V_5 \rightarrow V_2$
$V_2 \rightarrow V_4$	$V_2 \rightarrow V_4 \rightarrow V_1$
$V_2 \rightarrow V_5$	$V_2 \rightarrow V_5 \rightarrow V_1$
$V_4 \rightarrow V_1$	$V_5 \rightarrow V_1 \rightarrow V_4$
$V_4 \rightarrow V_2$	$V_5 \rightarrow V_2 \rightarrow V_4$
$V_5 \rightarrow V_2$	$V_4 \rightarrow V_1 \rightarrow V_5$
$V_5 \rightarrow V_1$	$V_4 \rightarrow V_2 \rightarrow V_5$

1.1.1.3 Poloviční křižovatková výhybka

Tato výhybka, hovorově poloviční Angličan, taktéž disponuje křížením čtyř zaústěných směrů, nicméně jen do jednoho protilehlého směru umožňuje jízdu přímým i odbočným směrem a do druhého protilehlého směru umožňuje pouze přímým směrem. [3]

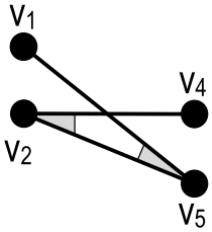
Značení této výhybky ve schématu je na Obrázku 6 a 7, přičemž značení záleží na tom, zda je možnost odbočného směru v horní nebo spodní půlce výhybky.

Obrázek 6: Poloviční křižovatková výhybka ve schématu s dolním odbočným směrem [4]



Obrázek 7: Poloviční křižovatková výhybka ve schématu s horním odbočným směrem [4]

Znázornění výhybky v modelu grafu je na Obrázku 8.



Obrázek 8: Poloviční křižovatková výhybka v grafu [4]

I zde je symbolickým trojúhelníkem označeno konstrukční omezení pro průjezd výhybkou podle směru příjezdu.

Všechny povolené a zakázané průjezdy poloviční křižovatkovou výhybkou na Obrázku 8. jsou uvedeny v Tabulce 3.

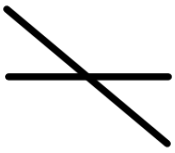
Tabulka 3: Povolené a zakázané průjezdy poloviční křižovatkovou výhybkou [Zdroj vlastní]

Povolené průjezdy	Zakázané průjezdy
$V_1 \rightarrow V_5$	$V_1 \rightarrow V_5 \rightarrow V_2$
$V_2 \rightarrow V_4$	$V_2 \rightarrow V_5 \rightarrow V_1$
$V_2 \rightarrow V_5$	$V_4 \rightarrow V_2 \rightarrow V_5$
$V_4 \rightarrow V_2$	$V_5 \rightarrow V_2 \rightarrow V_4$
$V_5 \rightarrow V_1$	
$V_5 \rightarrow V_2$	

1.1.1.4 Kolejová křižovatka

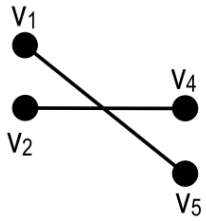
V tomto případě se nejedná přímo o výhybku – dochází zde pouze ke křížení čtyř zaústěných směrů a do obou protilehlých směrů lze pokračovat výhradně přímým směrem. [2]

Značení této křižovatky ve schématu je na Obrázku 9.



Obrázek 9: Kolejová křižovatka ve schématu [4]

Značení křižovatky v modelu grafu je na Obrázku 10.



Obrázek 10: Kolejová křižovatka v grafu [4]

1.1.2 Model infrastruktury s využitím neorientovaného grafu

Model infrastruktury je reprezentován hranově ohodnoceným neorientovaným grafem G, jehož vlastnosti jsou popsány v Tabulce 1.

Tabulka 4: Vlastnosti grafu reprezentujícího model infrastruktury [Zdroj vlastní]

G	Hranově ohodnocený neorientovaný graf s matematickou definicí: $G=(V,E,\varphi,\omega)$
V(G)	Množina vrcholů grafu G $V_{branch}(G) \subseteq V(G)$ V(G): Množina všech vrcholů grafu. $V_{branch}(G)$: Podmnožina V(G), která obsahuje tzv. hraniční vrcholy.
E(G)	Množina hran grafu G
φ	Incidenční funkce grafu G: $\varphi: E(G) \rightarrow \{(v, u) \mid u, v) \in V(G) \times V(G), v \neq u\}$ $\forall v \in V(G): \deg(v) \in \{1,2,3\}, m \leq \lfloor (3/2)\, n \rfloor$ pro $n \geq 4$, $n = \lvert V(G) \rvert$, $m = \lvert E(G) \rvert$
ω	Funkce ohodnocení hran graf G $\omega: E(G) \rightarrow R^{+}$

1.1.2.1 Ohodnocení hrany

Ohodnocení hran je popsáno funkcí ω, která každé hraně z množiny E(G) přiřadí ohodnocení, které je kladné reálné číslo a odpovídá hodnotě délky kolejového segmentu v modelu infrastruktury.

1.1.2.2 Incidenční funkce

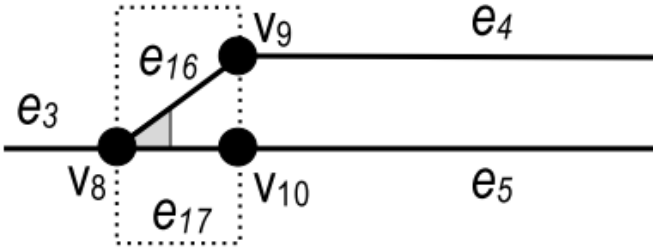
Vztah mezi hranami z množiny E(G) a vrcholy z množiny V(G) je popsán funkcí φ. Funkce každé hraně přiřazuje neuspořádaný pár vrcholů (v, u), které hrana propojuje. Funkce také udává, že graf neobsahuje smyčky (v ≠ u) ani násobné hrany.

1.1.2.3 Hraniční vrcholy

Množina $V_{branch}(G)$, která je podmnožinou V(G), obsahuje vrcholy, které jsou označeny jako hraniční vrcholy. V těchto vrcholech poté musí začínat a končit cesty nalezené v modelu infrastruktury, které se poté používají při hledání souběžných a nejdelších cest

1.1.2.4 Zakázaná odbočení

Tento pojem vychází z konstrukčního omezení výhybky, které je popsané v podkapitole 1.1.1. V grafu je toto omezení zavedeno tak, že každý vrchol si pro své incidenční hrany, kterých se toto omezení týká, vede množinu incidenčních hran, po kterých nelze pokračovat, pakliže směr příjezdu do vrcholu je po dané hraně.



Obrázek 11: Ukázka modelu jednoduché výhybky v grafu [4]

Příklad je ukázán na Obrázku 11. Omezení výhybky, které je znázorněno šedivým trojúhelníkem, se týká hran e_{16} a e_{17}. Vrchol V_8 bude pro incidenční hrany e_{16} a e_{17} evidovat množinu hran, na které se z těchto hran nedá dostat. Pro hranu e_{17} tato množina obsahuje hranu e_{16} a naopak, pro hranu e_{16} obsahuje hranu e_{17}. Na rozdíl od toho pro hranu e_3 není potřeba žádná taková množina, protože z ní lze pokračovat jak na hranu e_{16}, tak na hranu e_{17}.

1.2 Problém vyhledávání vlakových cest

Vyhledávání vlakových cest je jedním z klíčových úkolů při řízení železniční dopravy a jsou vyhledávány tak, aby byla zajištěna plynulost, bezpečnost a spolehlivosti provozu a efektivní využití kapacity infrastruktury a minimalizace vzniku zpoždění.

Proces hledání vlakových cest potřebuje velké množství vstupních dat a ovlivňuje ho celá řada faktorů a požadavků, což z toho činí složitý a komplexní úkon [5]. Faktory, které je potřeba zohledňovat, zahrnují především možnosti křižovaní na trati, aby tak nedocházelo ke kolizím,

různé priority vlaků (osobní vlaky mají přednost před nákladními), technické parametry trati (sklon trati, brzdné možnosti atd.) a platné směrnice (následná mezidobí mezi vlaky, maximální povolená rychlost atd.) [6]. Je nutné také brát úvahu požadavky objednavatelů veřejné dopravy, požadavky na kvalitu dopravy a sezonní požadavky na spojení [6].

Faktorů je samozřejmě mnohem více, nicméně pro zpracování tohoto tématu nejsou potřeba, jelikož jediné, co se zde při hledání souběžných a nejdelších cest kontroluje je, jestli jsou souběžné jízdní cesty disjunktní a jestli nejsou porušována konstrukční omezení výhybek.

2 PRINCIPY HLEDÁNÍ SOUBĚŽNÝCH A NEJDELŠÍCH JÍZDNÍCH CEST

Hledání souběžných a nejdelších jízdních cest je součástí plánování železniční dopravy a slouží hlavně k optimalizaci dopravy a maximálnímu využití kapacity infrastruktury, přičemž jsou brány v úvahu bezpečnostní pravidla a aktuální situace v železniční síti. Tyto koncepty potom mají přímý vliv na plynulost provozu a dobu jízdy vlaku.

2.1 Definice souběžných jízdních cest

V první řadě je potřeba si ujasnit pojem cesta v rámci grafu, který je modelem kolejiště. Jedná se o posloupnost vrcholů a hran, ve které se neopakují vrcholy [8]. Souběžné cesty, také nazývané disjunktní cesty, tvoří neuspořádanou n-tici cest, které nemají žádný společný vrchol [8]. Tím pádem je zaručeno, že vlaky mohou po těchto cestách projíždět současně a nehrozí riziko jejich kolize. Velky význam to má v infrastrukturách s vysokou frekvencí spojů, kde je nutné zajistit plynulost provozu a minimalizovat překrývání cest, s čímž je potom spojené zpoždění, kvůli čekání na uvolnění koleje.

V rámci této práce probíhá hledání souběžných cest tak, že se nejdříve pomocí algoritmu prohledávání grafu do hloubky, anglicky Depth First Search (DFS) [9], najdou všechny unikátní cesty na základě zadaných hraničních vrcholů. Z těchto unikátních cest se poté tvoří veškeré možné n-tice cest, které jsou navzájem disjunktní.

2.2 Definice nejdelších jízdních cest

Při hledání nejdelší jízdní cesty je cílem nalézt nejdelší možnou cestu mezi dvěma hraničními vrcholy grafu. Informace o této cestě je důležitá hlavně při plánování cesty pro vlaky s dlouhou soupravou, což jsou především nákladní vlaky, ale i třeba dálkové osobní vlaky a je potřeba například tyto vlaky odstavit v železniční stanici. Nejdelší cesta potom udává, zda je vlak možné umístit v infrastruktuře stanice, nebo je příliš dlouhý a zasahoval by na traťové koleje.

Hledání nejdelší cesty v grafu je možné řešit pomocí některých algoritmů pro hledání nejkratší cesty, jako jsou Dijkstrův algoritmus a Bellman-Ford algoritmus, které je ovšem potřeba modifikovat na nejdelší cestu [10]. Lze také použít DAG-based Shortest Path algoritmus, nicméně je nutné zajistit, aby v grafu nevznikaly cykly [11].

V této práci je nejdelší cesta hledána pomocí modifikovaného Dijkstrova algoritmu s prioritní frontou implementovanou pomocí binární haldy, protože je pro použitý model infrastruktury nejvíce vhodný. Je to z důvodu, že se jedná o řídký graf ($|E(G)| << |V(G)|^2$) [12], jelikož každý vrchol je incidentní nanejvýš s 3 hranami. Asymptotická výpočetní složitost

tohoto algoritmu potom je O ((|V(G)| + |E(G)|) × log(|V(G)|)), což odpovídá polynomiální asymptotické výpočetní složitosti [12]. Podrobněji je tento i další algoritmy popsány v Kapitole 3.

2.3 Kritéria a omezení vyhledávání

Vyhledávání cest je ovlivněno řadou našich kritérií, které definují vlastnosti nalezených cest, ale i omezeními, které mohou dostupné možnosti pro vyhledávání značně omezit. Vzhledem k tomu, že železniční provoz je dynamický, je potřeba brát v úvahu vnější i vnitřní faktory, které mohou vyhledávání také ovlivnit a často se jedná o věci, kterým nelze úplně předejít.

Klíčové kritérium pro souběžné cesty je nezávislost cest, aby nedocházelo k jejich křížení. Dalším kritériem je nalezení co největšího množství souběžných cest. Kritériem může být i zohledňování priorit vlaků, aby vlaky s vyšší prioritou dostali nejlepší z nalezených cest.

Základním kritériem pro nejdelší cestu je, aby trasa měla maximální možnou délku. Důležitým kritériem také je, aby vlak bylo možné na cestu umístit v celku a nemusel být rozdělen. Kritéria také mohou být, aby vlak nezasahoval do traťových kolejí, a tím neblokoval průjezdy vlaků, a aby cesta nevedla po nástupištních kolejích.

Omezení, které ovlivňují vyhledávání cest, jsou často společné pro souběžné a nejdelší cesty. Jedná se o výluky kolejí, ať už kvůli poruše, plánované údržbě, mimořádné události, nehodě či jiné technické poruše. Omezení může vzniknout také kvůli počasí, například z důvodu silného deště, sněžení, povodní nebo extrémního větru. Aktuální kapacita infrastruktury a obsazenost kolejí a výhybek může možnosti průjezdu také výrazně ovlivnit. Mohou také být nastavena omezení, aby byly při vyhledávání vynechány konkrétní koleje, výhybky i cesty z jiných než technických důvodů. Omezit vyhledávání lze také výběrem počátečních a cílových bodů cest. U souběžných cest může být počet omezen i přijatými nařízeními nebo bezpečnostními opatřeními, které specifikují maximální povolený počet souběžných cest.

Z těchto důvodu je zřejmě, že tyto trasy nelze vyhledávat staticky a je potřeba pracovat s aktuálními provozními podmínkami, které se mohou měnit v reálném čase. Je tedy potřeba, aby vyhledávací algoritmy byly dostatečně flexibilní a umožňovali rychlou adaptaci jízdních cest podle aktuálních podmínek.

3 PŘEHLED EXISTUJÍCÍCH METOD A ALGORITMŮ PRO HLEDÁNÍ CEST A PROHLÍDKY GRAFŮ

Metod a algoritmů pro hledání souběžných a nejdelších cest je celá řada. Jejich použití se bude odvíjet od vlastností datové struktury či požadavků na rychlost a výstup algoritmu. Některé algoritmy se nedají použít pro hledání nejdelší cesty, nicméně jsou používány pro hledání nejkratší cesty, což je také podstatná informace, proto jsou zde krátce zmíněny.

3.1 Popis běžně používaných algoritmů

3.1.1 Dijkstrův algoritmus

Dijkstrův algoritmus je jeden z nejznámějších algoritmů pro hledání nejkratší cesty v grafu. Dá se aplikovat na neorientované i orientované grafy, nicméně podmínkou je, aby hrany byly ohodnoceny pouze kladnými reálnými čísly. Paměťová složitost algoritmu se odvíjí od použité datové struktury a od toho, jestli se hledá nejkratší cesta do konkrétního vrcholu, nebo se prochází celý graf a hledá se nejkratší cesta do všech vrcholů. [13]

Inicializace algoritmu začíná definicí výchozího uzlu a nastavením jeho vzdálenosti na 0 a u všech ostatních uzlů se vzdálenost nastaví na ∞ (nekonečno). Všechny vrcholy jsou také označeny jako dočasné. Poté se ve smyčce opakuje následující krok. Z dočasných vrcholů se vybere vrchol s nejmenší hodnotou délky cesty od počátečního vrcholu a jeho stav se nastaví na tzv. definitivní. Poté se aktualizují hodnoty vzdáleností u sousedních dočasných vrcholů. Porovnává se součet hodnoty aktuálního vrcholu a ceny hrany do sousedního vrcholu s dosavadní hodnotou vzdálenosti sousedního vrcholu. [15]

Pokud je nová hodnota menší než současná hodnota vzdálenosti sousedního vrcholu, nastaví se mu nová hodnota. Tato operace aktualizace hodnoty se nazývá relaxace hran. [13]

Algoritmus končí v momentě, kdy jsou buď všechny vrcholy označeny jako definitivní, nebo je cílový vrchol označen jako definitivní. Aby bylo možné provést rekonstrukci nejkratší cesty, je potřeba u každého vrcholu uchovávat informaci o jeho předchůdci [16].

To, jak je algoritmus implementován, má vliv na rychlost a asymptotickou výpočetní složitost algoritmu. Lze ho implementovat bez prioritní fronty, takže je pro dočasné vrcholy použito pole nebo seznam vrcholů, kde se vrchol s nejnižší vzdáleností hledá lineárním prohledáváním. Celková asymptotická výpočetní složitost této implementace je $O(|V(G)^2|)$, jelikož při každém průchodu jsou procházeny všechny dočasné vrcholy. Druhým způsobem je implementace s prioritní frontou. Z té se v každém kroku v čase $O(\log(|V(G)|))$ nalezne a smaže vrchol s nejmenší vzdáleností od výchozího vrcholu a případně se upraví vzdálenosti

jeho sousedních vrcholů. Při implementaci prioritní fronty pomocí binární hlady bude poté celková asymptotická výpočetní složitost O ((|V(G)| + |E(G)|) log(|V(G)|)). Prioritní frontu je nicméně možné implementovat i pomocí jiných druhů hald, a to třeba Fibonacciho halda, kde je hodnota prvku upravována v konstantním čase, a proto je výsledná asymptotická výpočetní složitost O (|E(G)| + |V(G)| log(|V(G)|)) nebo k-regulární halda. [15]

Dijkstrův algoritmus je možné použít i pro hledání nejdelší cesty, nicméně je potřeba provést několik úprav. První je změna výchozí hodnoty vzdálenosti vrcholů z ∞ na -∞. Druhá změna je, že do fronty se vkládají negace hodnoty vzdálenosti, tím pádem bude fronta seřazena podle nejdelší vzdálenosti. Třetím krokem je změna při aktualizaci vzdálenosti vrcholu tak, aby se hodnota aktualizovala, pokud je nová hodnota větší než stávající. Poslední změnou je, aby se po skončení algoritmu hledal nevzdálenější vrchol.

Jedná se o spolehlivý, flexibilní a, pokud je implementována prioritní fronta, tak i efektivní algoritmus. Nicméně, jak již bylo zmíněno, Dijkstrův algoritmus je spolehlivý pouze pokud graf neobsahuje záporně ohodnocené hrany, protože by jinak nefungoval správně. Algoritmus taktéž není moc vhodný pro použití na velké husté grafy. [13]

3.1.2 Bellman-Fordův algoritmus

Stejně jako Dijkstrův algoritmus slouží Bellman-Fordův algoritmus pro hledání nejkratší cesty z konkrétního vrcholu do všech ostatních vrcholů. Čím se ale liší je, že dokáže pracovat i s hranami, které mají záporné ohodnocení. Druhou odlišností je, že se v každém kroku provádí relaxace všech hran v grafu, na rozdíl od Dijkstrova algoritmu, kde se prováděla relaxace jen pro incidenční hrany aktuálního vrcholu. [14]

Jedná se v podstatě o dva cykly. První cyklus má počet iterací roven |V(G)| − 1. Druhý cyklus, který je vnořen do prvního, potom provede relaxaci všech hran. [13]

Při inicializaci algoritmu se nastaví vzdálenost počátečnímu vrcholu na 0 a všem ostatním ∞. Poté se provádějí ony dva do sebe vnořené cykly, které jsou zmíněny výše. [13]

V posledním kroku algoritmu je provedena ještě jednou relaxace hran s cílem zjistit, jestli bude nějaká relaxace ještě úspěšná a dojde ke zmenšení vzdálenosti některého vrcholu. Pokud k tomu dojde, znamená to, že graf obsahuje hranu se záporným ohodnocením. [15]

Asymptotická výpočetní složitost tohoto algoritmu je O (|V(G)| × |E(G)|), jelikož relaxace hrany má konstantní složitost. Bellman-Fordův algoritmus se používá především tehdy, kdy je reálný předpoklad, že graf obsahuje záporně ohodnocené hrany z důvodu nižší rychlosti výpočtu algoritmu. [15]

3.1.3 Floyd-Warshallův algoritmus

Floydův–Warshallův algoritmus je z kategorie dynamických programovacích algoritmů a slouží pro hledání nejkratší cesty mezi každou dvojicí vrcholů v grafu [16]. Když je více cest se stejnou délkou, algoritmus vybere tu s nejmenším počtem hran. Podmínkou pro použití algoritmu je, aby graf neobsahoval záporné cykly [13].

Spuštění algoritmu začíná inicializací čtvercové matice sousednosti délek o velikosti $|V(G)| \times |V(G)|$ se vzdálenostmi mezi vrcholy (Tabulka 5). Pokud mezi dvěma vrcholy (i, j) vede hrana s vahou l, nastaví se tato hodnota do matice na index (i, j). Pokud mezi vrcholy (i, j) nevede přímá hrana, nastaví se do matice na daný index ∞ (nekonečno). Na diagonále má matice samé 0. [15]

V každé iteraci algoritmu se potom přepočítají hodnoty v matici, přičemž se každou iterací zvětšuje množina prostředníků, která udává, přes které vrcholy může cesta mezi dvojicí vrcholů procházet. V první iteraci tedy matice vyjadřuje vzdálenosti z možností využití jednoho daného prostředníka. V *k* iteraci budou v matici vzdálenosti s možností využití *k* daných prostředníků. Nová délka cesty mezi dvojicí vrcholů se porovná se stávající, pokud je menší, upraví se hodnota na patřičném indexu v matici. Výsledkem algoritmu je matice s nejkratší vzdáleností mezí každou dvojicí vrcholů (Tabulka 6). Pokud pole v matici neobsahuje hodnotu, znamená to, že cesta mezi vrcholy neexistuje. [14]

Jelikož se algoritmus skládá ze tří vnořených cyklů (k, i, j), jeho asymptotická výpočetní složitost je $O(|V(G)|^3)$ a paměťová složitost je $O(|V(G)|^2)$ kvůli matici vzdálenosti. Právě z důvodu vysoké výpočetní složitosti je algoritmus nevhodný pro grafy s velkým množstvím vrcholů. Výhodou algoritmu je jeho snadná implementace a rychlost naprogramování. [13]

Tabulka 5: Matice sousednosti délek při inicializaci algoritmu [15]

Vrchol	A	B	C	D	E
A	0	3	8	∞	-4
B	∞	0	∞	1	7
C	∞	4	0	∞	∞
D	2	∞	-5	0	∞
E	∞	∞	∞	6	0

Tabulka 6: Matice délek nejkratších cest při skončení algoritmu [15]

Vrchol	A	B	C	D	E
A	0	1	-3	2	-4
B	3	0	-4	1	-1
C	7	4	0	5	3
D	2	-1	-5	0	-5
E	8	5	1	6	0

3.1.4 Johnsonův algoritmus

Johnsonův algoritmus umožňuje hledání nejkratších cest mezi všemi páry vrcholů grafu, který může obsahovat záporně ohodnocené hrany, nicméně nesmí existovat záporné cykly. Tento algoritmus je kombinací Dijkstrova a Bellman-Fordova algoritmu a jeho základní myšlenkou je n-násobné použití Dijkstrova algoritmu, kde *n* je počet vrcholů grafu, a výpočet transformace vstupního grafu, která odstraní záporné ohodnocení hran pomocí Bellman-Fordova algoritmu. [14]

Prvním krokem algoritmu je vytvoření nového pomocného uzlu *Q*, který bude mít hranu s nulovým ohodnocením do všech vrcholů grafu. V druhém kroku se použije Bellman-Fordův

algoritmus a z vrcholu *Q* vyhledá pro každý vrchol *V* nejkratší cestu s hodnotou *H(V)*, což odpovídá cestě z *Q* do *V*. Pokud bude kdykoliv během tohoto kroku zjištěn záporný cyklus, algoritmus se ukončí. V dalším kroku se pro všechny hrany přepočítá ohodnocení s pomocí hodnoty z předchozího kroku. Pro každou hranu z *U* do *V* s ohodnocením L bude přepočtené ohodnocení dáno vztahem: $L'(U,V) = L(U,V) + H(U) - H(V)$. Posledním krokem je spuštění Dijkstrova algoritmu pro každý vrchol s přepočítanými vahami hran. Nalezené nejkratší cesty v transformovaném grafu mají kvůli přepočtu hran jinou délku a neodpovídají skutečným hodnotám, nicméně jsou totožné s cestami v původním grafu. To znamená, že nejkratší cesta v původním grafu bude taktéž nejkratší cesta v transformovaném grafu. [14]

Celková asymptotická výpočetní složitost Johnsonova algoritmu je $O(|V(G)|^2 \times \log(|V(G)|) + (|V(G)| \times |E(G)|)$ [15]. Při použití na řídké grafy je rychlejší než Floyd-Warshallův algoritmus, nicméně i přes tuto skutečnost se v praktických příkladech příliš nepoužívá, stejně jako Floyd-Warshallův algoritmus [14].

3.1.5 (A-star) algoritmus

A* algoritmus je vyhledávácí algoritmus pro hledání nejkratší cesty mezi dvěma vrcholy. Jedná se o upravený Dijkstrův algoritmus, který kromě skutečné délky cesty pracuje i s heuristickým odhadem vzdálenosti do cílového vrcholu. [18]

Součet těchto dvou hodnot potom tvoří celkový heuristický odhad délky, který má přiřazený každá cesta. Tento výpočet je vyjádřen funkcí: $f(n) = g(n) + h(n)$, kde $g(n)$ je funkce vyjadřující vzdálenost od počátečního vrcholu do vrcholu n, $h(n)$ představuje heuristickou funkci, která odhaduje vzdálenost z vrcholu n do konečného vrcholu, a $f(n)$ potom vyjadřuje odhad délky cesty, která vede přes vrchol n. [18]

Inicializace algoritmu začíná vytvořením prioritní fronty, ve které budou udržovány tzv. nenavštívené vrcholy, kde vrcholy s nižší hodnotu *f* mají vyšší prioritu. Na začátku se do ní vloží startovní vrchol a nastaví se jeho hodnoty $g(0) = 0$ a $f(0) = h(0)$. V každém následujícím kroku algoritmu se vybere z fronty vrchol s nejnižší hodnotou *f*, odstraní se z fronty a pro jeho sousední vrcholy se vypočítá hodnota *h* a *f*. Tyto vrcholy jsou poté přidány do fronty, nebo se aktualizují hodnoty *f*, pokud již dané vrcholy ve frontě jsou a nová hodnota *f* je nižší. Algoritmus skončí v okamžiku, kdy je prioritní fronta prázdná. Hodnota *f* koncového vrcholu potom vyjadřuje délku nejkratší cesty mezi počátečním a koncovým vrcholem. [18]

Asymptotická výpočetní složitost algoritmu závisí na kvalitě a typu zvolené heuristiky, přičemž nejhůře může být exponenciální a nejlépe polynomiální [19]. Typ heuristiky se vybírá podle vlastností prostředí a možností pohybu, kde na výběr jsou Manhattanská vzdálenost,

Diagonální vzdálenost, Euklidovská vzdálenost, Maximova metrika a další [21]. Prostorová složitost, stejně jako asymptotická výpočetní složitost, může být v nejhorším případě exponenciální, nicméně ji lze redukovat za cenu suboptimálnosti algoritmu, například eliminací špatných cest z prioritní fronty, které pravděpodobně nebudou nejkratší cestou [20].

3.1.6 DAG-based Shortest Path algoritmus

DAG-Based Shortest Path algoritmus slouží pro hledání nejkratší cesty ze zdrojového vrcholu do všech ostatních vrcholů v orientovaných acyklických grafech, který může obsahovat i záporně ohodnocené hrany. Algoritmus pro hledání používá topologické uspořádání grafu, což znamená, že každý vrchol je zpracován dříve než jeho následníci. [22]

Při inicializaci algoritmu se počátečnímu vrcholu nastaví vzdálenost na 0 a všem ostatním vrcholům na ∞. Poté se podle topologického pořadí zpracovávají vrcholy a pro každý vrchol se aktualizují vzdálenosti jeho sousedních vrcholů v případě, že součet vzdálenosti aktuálního vrcholu a ohodnocení hrany je menší než hodnota vzdálenosti daného sousedního vrcholu. [11]

Asymptotická výpočetní složitost algoritmu je O (|V(G)| + |E(G)|), což je rychlejší než Dijkstrův a Bellman-Fordův algoritmus, nicméně nevýhodou je již zmíněná nutnost mít graf bez cyklů. [11]

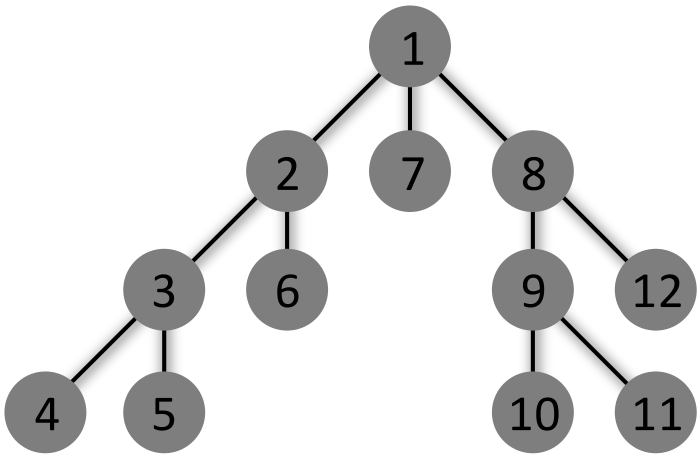
3.1.7 Depth First Search

Depth-first search (DFS), česky Prohlídka do hloubky, je algoritmus pro prohledávání datových struktur strom a graf s využitím zásobníku (LIFO). U struktury strom algoritmus začíná v kořeni stromu (root), v případě grafu začíná ve zvoleném počátečním vrcholu. [15]

Algoritmus začíná vložením počátečního vrcholu do zásobníku a vytvořením seznamu navštívených vrcholů. Poté v cyklu opakuje krok, kdy se ze zásobníku vybere vrchol, vloží se na seznam navštívených vrcholů a do zásobníku se vloží jeho sousední vrcholy, které ještě nejsou v seznamu navštívených vrcholů. Když se dojde do vrcholu, který nemá sousední vrcholy, nebo už jsou všechny navštívené a nejde tím pádem pokračovat dál, algoritmus se postupně vrací k nejbližšímu nenavštívenému vrcholu a navštívené vrcholy ze zásobníku odstraňuje. Algoritmus končí ve chvíli, kdy je zásobník prázdný. [15]

DFS algoritmus lze implementovat dvěma způsoby: rekurzivně a iterativně. U rekurzivního DFS se u každého vrcholu rekurzivně volá funkce s DFS pro sousední vrcholy a přidává se do interního zásobníku volání funkcí. Když se algoritmus dostane do slepého

vrcholu, aktuální funkce skončí a vrací se do předchozího stavu. U iterativního DFS je explicitně vytvořen zásobník pro správu vrcholů, který nahrazuje rekurzivní volání funkcí. [24]



Obrázek 12: Prohlídka grafu do hloubky [25]

Obě implementace prohledají strukturu stejným způsobem, nicméně volba bude záviset na velikosti datové struktury a dalších požadavcích. Rekurzivní DFS je snazší na implementaci a pochopení a je vhodné pro malé, střední i velké struktury. Nevýhodou je, že u velmi velkých struktur hrozí riziko přetečení zásobníku. U iterativního DFS přetečení zásobníku nehrozí díky jeho manuální správě, nicméně je složitější na implementaci a má obtížnější čitelnost. [24]

Asymptotická výpočetní složitost algoritmu je O (|V(G)| + |E(G)|) a prostorová složitost je O (|V(G)|) [15]. Na Obrázku 12 je ukázková datová struktura stromu a čísla vrcholů ukazují, v jaké pořadí DFS algoritmus vrcholy prochází.

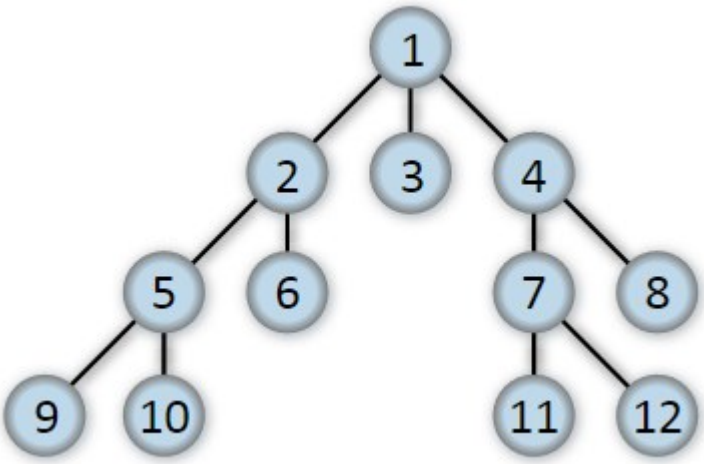
3.1.8 Breadth-first Search

Breadth-first search (DFS), česky Prohlídka do šířky, je algoritmus pro prohledávání datových struktur strom a graf po jednotlivých úrovních s využitím fronty (FIFO). To znamená, že se nejdřív projdou všechny sousední vrcholy od vybraného vrcholu, než se přejde na další úroveň vrcholů. Algoritmus začíná v kořeni, v případě stromu, nebo ve zvoleném počátečním vrcholu, v případě grafu. [15]

Inicializace algoritmu zahrnuje vytvoření seznamu navštívených vrcholů a vložení počátečního vrcholu do fronty. Následně se v cyklu opakuje krok, kdy se z fronty vybere vrchol, který se zároveň odstraní, vloží se do seznamu navštívených vrcholů a do fronty se vloží všechny jeho sousední vrcholy, které ještě nebyly navštívené. V okamžiku, kdy je fronta prázdná, algoritmus končí. [15]

Co je výhodou BFS je to, že je efektivní při hledání nejkratší cesty, nicméně jen

v neohodnocených grafech. Kdy naopak BFS není vhodný, je pro velké a hluboké grafy, kde může být poměrně paměťově náročný, kvůli velkému počtu vrcholů ve frontě. [26]



Obrázek 13: Prohlídka grafu do šířky [28]

Složitost algoritmu je stejná jako u DFS, kdy asymptotická výpočetní složitost je O (|V(G)| + |E(G)|) a prostorová složitost je O (|V(G)|) [26]. Na Obrázku 13 je datová struktura strom, kde čísla vrcholů udávají, v jakém pořadí BFS algoritmus vrcholy prošel.

3.2 Příklady uplatnění v praxi

Algoritmy pro prohledávání grafů a hledání cest mezi vrcholy se využívají v široké škále aplikací. Uplatňují se v oblastech dopravy, síťové analýzy, umělé inteligence, navigačních systémů, herního průmyslu, matematice atd. a hrají zde klíčovou roli.

1. **Navigační systémy a plánování tras**

   Navigační a dopravní systémy, jako jsou Google Maps, Uber, GPS navigace, využívají algoritmy pro hledání nejkratší nebo optimální cesty mezi dvěma místy. Patří mezi ně Dijkstrův, Floyd-Warshallův, Johnsonův, A* a Bellman-Fordův algoritmus. Tyto algoritmy umožňují nejen hledání nejkratší a nejrychlejší cesty mezi dvěma lokacemi, ale i plánování cest kombinujících více způsobů dopravy. Umožňují i návrh cest na základě aktuální dopravní situace a omezení s cílem minimalizovat dobu jízdy a zároveň se vyhnout dopravním zácpám nebo jiným komplikacím. [15]

2. **Počítačové sítě a telekomunikace**

   Algoritmy jako A*, Dijkstrův, Floyd-Warshallův a Bellman-Fordův se používají pro optimalizaci přenosu dat a určení optimální nebo nejkratší cesty při směrování paketů v počítačových sítích. Tím zajišťují minimální latenci a spolehlivost přenosu dat. Algoritmy

se využívají ve směrovacích protokolech, jako jsou RIP, OSPF, IS-IS. [14]

3. **Sociální sítě**

   Sociální sítě, jako jsou Facebook, Instagram nebo Twitter, využívají algoritmy BFS, DFS a Dijkstrův algoritmus pro zjišťování vztahů mezi uživateli a jejich vlivu. Na základě těchto analýz se potom doporučuje obsah, skupiny nebo další uživatelé, jako návrhy na sledování. [29]

4. **Umělá inteligence a hry**

   Algoritmy DFS, BFS a A* se rozsáhle využívají při trénovaní umělé inteligence a ve vývoji her. Uplatňují se při logických hrách, jako jsou šachy, bludiště, puzzle nebo Sudoku, kde pomáhají s průzkumem možných řešení a určením nejlepšího kroku. V bludišti pomáhají s navigací a hledáním cesty. Ve hrách, jako třeba Warcraft nebo v žánru Tower Defense, se uplatňují také při tvorbě NPC postav, kterým umožňují pohyb v dynamickém světě, vyhýbání se překážkám a optimální navigaci pro dosažení cíle. [20]

5. **Analýza grafů**

   BFS a DFS algoritmy se používají při analýze grafů a zjišťování jejich vlastností, například k identifikaci připojených komponent, průzkumu grafu, mapování cest, hledání kostry, detekci cyklů, testování bipartitnosti grafu, topologickému řazení atd. [15]

6. **Robotika a průmyslová automatizace**

   Pro pohyb robotů a autonomních vozidel se využívají Dijkstrův, A* a Bellman-Fordův algoritmus, které umožňují pohyb v prostředí po nejkratších nebo optimálních cestách a zajišťují, aby nedocházelo ke kolizím s objekty a zvládali na ně reagovat v reálném čase. Algoritmy DFS a BFS se používají k navigaci v neznámém prostředí pomocí mapování okolí a hledání cest. [29]

PRAKTICKÁ ČÁST

V teoretické části byla vysvětlena a popsána problematika vyhledávání souběžných a nejdelších jízdních cest, vysvětleny důležité pojmy spojené s modelem železniční infrastruktury. Cílem praktické části je popsat aplikaci, která bude v poskytnutém modelu infrastruktury vyhledávat souběžné a nejdelší cesty. Po nalezení cest bude k dispozici možnost jejich vizualizace v modelu, aby byly vedle textové podoby dostupné i v grafickém znázornění.

4 IMPLEMENTACE A TVORBA ŘEŠENÍ

V této podkapitole je popsána zvolená implementace, představeny všechny použité nástroje a technologie, popsán postup vytvoření projektu, a nakonec ukázána výsledná aplikace s popisem jejích funkcí a algoritmů, které tyto funkce využívají.

4.1 Technologické prostředí a vývojové nástroje

4.1.1 C# .NET

Aplikace je implementována v C# na platformě .NET. Tuto volbu jsem učinil primárně na základě svých předchozích zkušeností a také proto, že k C# a .NET existuje velké množství materiálů a manuálů.

C# je open-source, multiplatformní, objektově orientovaný programovací jazyk, který běží na .NET frameworku. [32]

.NET je open-source platforma určená pro vývoj desktopových, webových a mobilních aplikací. Tyto aplikace lze spustit na jakémkoli operačním systému. .NET umožňuje vytvářet aplikace v řadě různých jazycích, jako třeba C#, F#, Visual Basic a další. Poskytuje také nástroje a knihovny, které podporují vývoj škálovatelného a výkonného softwaru. [34]

4.1.2 Microsoft Visual Studio 2022

Pro vývoj aplikace jsem zvolil vývojové prostředí, dále IDE, Microsoft Visual Studio 2022. Vybral jsem ho z důvodu, že umožňuje vývoj aplikací v programovacím jazyce C# a mám s ním již předchozí zkušenosti.

4.1.3 Windows Forms

Aplikace je vytvořena pomocí frameworku WindowsForms. WinForms, jak je zkráceně označován, je grafický framework v rámci .NET, který slouží k vývoji desktopových aplikací pro Windows. Pomocí grafického designeru umožňuje tvorbu formulářových aplikací. K tomu je k dispozici sada připravených ovládacích prvků (označovány také jako komponenty), jako například tlačítka, textová pole, seznamy, panely nebo je možné si vytvořit vlastní komponenty, případně upravit již existující. [36]

4.1.4 MesoRail

Model železniční infrastruktury, který je v této práci použit, byl vytvořen v editoru simulačního nástroje MesoRail. Jedná se o mesoskopický simulátor železničního provozu, zaměřený na

zkoumání dopravních charakteristik železniční stanice. Jeho hlavním cílem je umožnit efektivní provádění simulačních studií [38].

4.1.5 SkiaSharp

V aplikaci je pro vykreslování obrázku infrastruktury se zvýrazněnými nalezenými cestami použito SkiaSharp. Jedná se o multiplatformní 2D grafické API pro platformu .NET, které je založené na open-source knihovně Skia Graphics Library od Google. Poskytuje komplexní 2D API pro vykreslování grafiky v různých prostředích, od mobilních aplikací až po servery a stolní počítače. [40]

4.2 Vytvoření projektu

Jako první je potřeba si zkontrolovat, zda máte v IDE nainstalované potřebné moduly, aby bylo možné aplikaci vytvářet a kompilovat. Jedná se konktrétně o moduly *Vývoj desktopových aplikací pomocí .NET* a *Vývoj pro Univerzální platformu Window*s. Ty se dají nainstalovat buď během prvotní instalace IDE, nebo je možné je instalovat dodatečně přes instalátor IDE.

Dalším krokem je samotné vytvoření projektu. V okně při vytváření nového projektu vyberte ze seznamu šablonu *Aplikace Windows Forms*, přičemž je nutné si dát pozor, aby se jednalo o šablonu pro C#. Následně zvolte umístění a pojmenování projektu, v mém případě je název *Parallel_and_longest_paths_in_track_railway_infrastructure_models*, a při volbě architektury vyberte .NET 6.0. Tímto je projekt vytvořen.

Posledním krokem je instalace knihovny SkiaSharp, kterou do projektu přidáte jako NuGet balíček. To uděláte pomocí správce balíčků NuGet, který najdete v sekci *Nástroje → Správce balíčků NuGet → Spravovat balíčky NuGet pro řešení…*. Zde vyhledejte knihovnu SkiaSharp, zvolte verzi 3.116.1, vyberte projekt, do kterého se má knihovna přidat, a dáte *Nainstalovat*. Při používání knihovny je vhodné přidat direktivu *using SkiaSharp*, abyste nemuseli neustále psát plné názvy tříd.

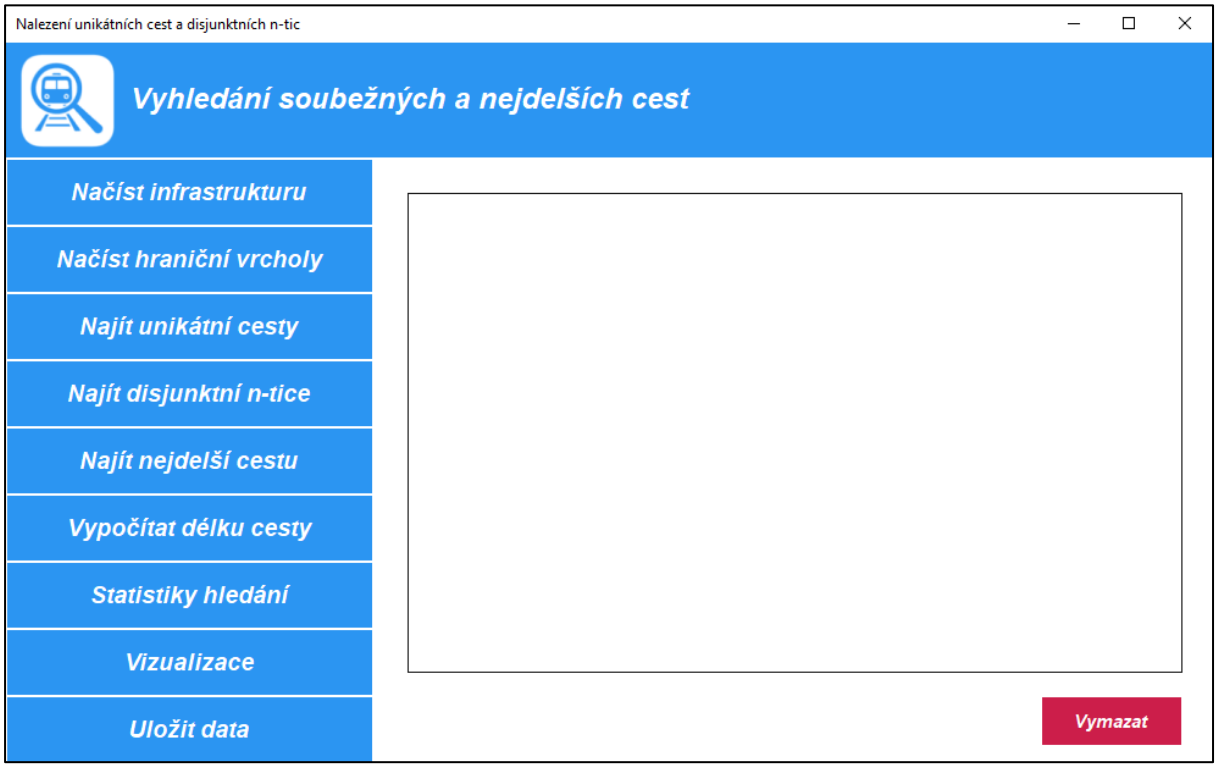
Tímto je projekt kompletně vytvořen a nastaven. Jelikož se jedná o WinForms aplikaci, bude fungovat pouze na systému Windows, protože je úzce propojena s Windows API. Pokud byste chtěli aplikaci spustit na Linuxu nebo macOS, je proto pár způsobů. První je Wine, což je Windows emulátor na Linuxu nebo macOS. Druhá možnost je Mono, která funguje na Linuxu a macOS a jedná se o open-source alternativu k .NET. Obě varianty ale mají nevýhodu, že v nich aplikace nemusí fungovat správně a může mít odlišně chování nebo způsobovat chyby. Složitějším řešením je přepsat aplikaci v alternativě WinForms, která je multiplatformní. Touto alternativou je například .NET MAUI nebo Avalonia UI.

5 UŽIVATELSKÉ ROZHRANÍ A JEHO FUNKCE

Tato kapitola obsahuje popis uživatelského rozhraní aplikace, jaké nabízí funkce, jak tyto funkce fungují a jaké algoritmy využívají.

5.1 Hlavní rozhraní aplikace

Po spuštění aplikace se zobrazí hlavní okno aplikace, které je možné vidět na Obrázku 14.



Obrázek 14: Hlavní okno aplikace [Zdroj vlastní]

Prostřednictvím tohoto okna je aplikace ovládána a jsou z něho dostupné všechny funkce. Nachází se zde sada tlačítek pro ovládání aplikace a pole, kam se zapisují výsledky funkcí a informace o stavu. Informační pole zajišťuje komponenta *ListBox* a slouží pro zobrazování informací o stavu funkcí nebo pro výpis výsledků některých funkcí. Informace o stavu funkce zahrnuje to, zda byla operace úspěšně dokončena, případně zda při zpracování došlo k chybě, a o jakou chybu se jednalo. Funkce jako například *Statistiky hledání* nebo *Vypočítat délku cesty* do této komponenty zapisují výsledky svých výpočtů.

Ke komponentě *ListBox* náleží tlačítko *Vymazat,* kterým lze vymazat obsah informačního pole v případě, že by v něm bylo například větší množství zpráv a přestalo by být přehledné.

5.2 Načtení infrastruktury a vstupních vrcholů

Po zapnutí aplikace je načtení dat první akce, kterou je potřeba udělat, a to pomocí operací *Načíst infrastrukturu* a *Načíst hraniční vrcholy*. Načítají se dva soubory. Jeden .xml soubor, který obsahuje model infrastruktury a jeden .csv soubor, ve kterém jsou vypsané ID hraničních vrcholů. Struktura souboru .xml a .csv jsou vidět na Obrázku 15 a 16.

```
<node id="227_3" km="305.98027007561535" point="1" prototypeID="229" realX="-1.0" realY="-1.0" x="5508.0" y="689.0"/>
<node id="110_0" km="305.47" point="0" prototypeID="108" realX="-1.0" realY="-1.0" x="3510.0" y="648.0"/>
<node id="108_1" km="305.45738900000003" point="1" prototypeID="108" realX="-1.0" realY="-1.0" x="3436.5" y="629.5"/>
<node id="73_1" km="305.44138000000004" point="2" prototypeID="108" realX="-1.0" realY="-1.0" x="3351.0" y="611.5"/>
<node id="74_1" km="305.44151625283956" point="3" prototypeID="108" realX="-1.0" realY="-1.0" x="3349.5" y="639.5"/>
<node id="158_0" km="305.51" point="0" prototypeID="109" realX="-1.0" realY="-1.0" x="3714.5" y="670.5"/>
<node id="109_1" km="305.497389" point="1" prototypeID="109" realX="-1.0" realY="-1.0" x="3665.0" y="670.5"/>
<node id="174_0" km="305.48138" point="2" prototypeID="109" realX="-1.0" realY="-1.0" x="3580.5" y="674.0"/>
<node id="110_1" km="305.4815245301035" point="3" prototypeID="109" realX="-1.0" realY="-1.0" x="3578.0" y="658.5"/>
</nodes>
<edges>
▼<edge endNodeID="231_1" id="231_01" prototypeID="231" prototypeKM="-1.0" prototypeType="SEGMENT" startNodeID="231_0">
  <setting arc="0.0" electrification="AC" gradient="0.0" isolationID="403" length="47.09999999994352" trackName="62-68" trackSection="" trackType="Stanicni"/>
  ▼<speeds orientation="LR">
    <speed type="R" value="100.0"/>
    <speed type="Sluz" value="80.0"/>
    <speed type="Os" value="100.0"/>
```

Obrázek 15: Ukázka dat v .xml souboru [Zdroj vlastní]

NodeName
273_1
264_1
319_1
316_1
538_1
556_1
172_1
168_1
171_1

Obrázek 16: Ukázka dat v .csv souboru [Zdroj vlastní]

Nejdříve je nutné načíst model infrastruktury ze souboru .xml a načítají se z něj vrcholy a hrany. Vrchol má následující atributy:

- *Key* (unikátní identifikátor vrcholu)
- *Edges* (seznam incidenčních hran vrcholu)
- *UnreachablesVertices* (seznamy nedostupných hran pro incidenční hrany)
- *PositionX* (souřadnice x)
- *PositionY* (souřadnice y)

Hrana má atributy:

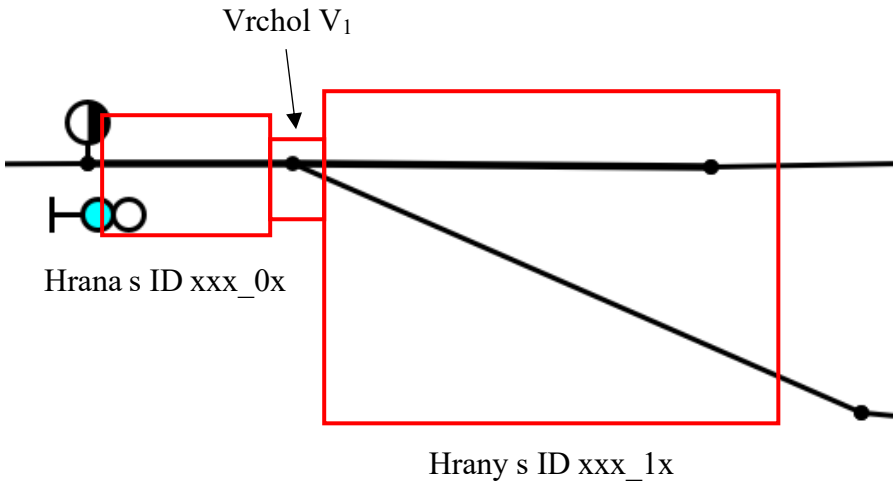
- *ID* (jednoznačný identifikátor hrany)
- *PrototypeType* (typ hrany)
- *Length* (délka hrany)
- *Start* (počáteční vrchol hrany)
- *Destination* (koncový vrchol hrany)

Jako první jsou načteny vrcholy, pro které se načte jejich klíč a souřadnice X a Y. Poté se načítají hrany. Pro ty se načte ID, typ hrany, délka a počáteční a koncový vrchol. Ještě během načítání každé hrany se podle klíče počátečního vrcholu tato hrana přiřadí patřičnému vrcholu do atributu *Edges*, ve kterém jsou uchovávány incidenční hrany vrcholu.

Po načtení všech hran se provede identifikace zakázaných odbočení. Pro každý vrchol se projdou jeho incidenční hrany a podle jejich ID a typu se identifikují zakázaná odbočení a zapíší se do atributu *UnreachablesVertices*.

Pro typ hrany *Double_cross,* což je hrana v křižovatkové výhybce, platí, že z ní nelze pokračovat na jinou hranu typu *Double_cross.* V atributu *UnreachablesVertices* bude pro každou incidenční *Double_cross* hranu seznam, ve kterém budou všechny ostatní incidenční *Double_cross* hrany daného vrcholu.

Pro typ hrany *Single_cross*, která odpovídá hraně jednoduché výhybky, je pro určení potřeba ještě ID hrany. Je to z důvodu, že z hrany *Single_cross* s ID ve formátu xxx_0x lze pokračovat do přímého i odbočného směru, zatímco z hran *Single_cross* s ID ve formátu xxx_1x nelze pokračovat po jiné hraně *Single_cross* s ID xxx_1x. Princip je znázorněn na Obrázku 17.



Obrázek 17: Výhybka s označením hran [Zdroj vlastní]

Vrchol V1 bude mít v *UnreachablesVertices* dva záznamy Každá hrana s ID ve formátu xxx_1x bude mít v seznamu zakázaných odbočení druhou hranu s ID xxx_1x.

Třetím typem hran je *Segment*, který označuje klasickou hranu bez výhybky, tudíž se zakázané odbočení na tento typ hrany nevztahuje.

Po dokončení identifikace zakázaných odbočení je potřeba načíst soubor .csv obsahující ID hraničních vrcholů. Podle těchto ID se dané vrcholy hledají v kolekci všech vrcholů, přičemž

tento proces slouží zároveň i jako kontrola, jestli vrcholy existují. Pokud jsou vrcholy nalezeny, jsou vloženy do zvláštní kolekce *inputVertexes*.

V případě, že načítání proběhne správně, zobrazí se uživateli zpráva o úspěšném načtení dat. Pokud dojde k jakékoli chybě, například nedostupný soubor, špatný formát dat, neexistující vrchol atd., vypíše se uživateli informace o neúspěšném načtení infrastruktury a co chybu způsobilo.

5.3 Nalezení unikátních cest mezi hraničními vrcholy

Před hledáním souběžných a nejdelších cest je nejprve nutné nalézt všechny unikátní cesty. Jedná se o všechny možné cesty z hraničních vrcholů, které zároveň musí také končit v hraničním vrcholu. Pro hledání cest je použit rekurzivně implementovaný DFS (popsán v podkapitole 3.1.7).

Každá nalezená unikátní cesta má atributy *Id* (unikátní identifikátor cesty), *Lenght* (délka cesty), *Nodes* (seznam vrcholů tvořící cestu). Délka cesty je vypočítána po jejím nalezení na základě vrcholů obsažených v seznamu *Nodes*. Od prvního vrcholu cesty se postupně sčítají délky hran, které spojují jednotlivé po sobě jdoucí vrcholy.

Po nalezení všech cest je potřeba odstranit duplikátní cesty. Tím se rozumí situace, kdy jedna cesta má vrcholy v pořadí 1-2-3 a druhá cesta má pořadí vrcholů 3-2-1. Takové cesty se berou jako shodné a jedna z nich je odstraněna.

Po úspěšném odstranění duplikátů se zobrazí zpráva informující uživatele o úspěšném nalezení unikátních cest. Pokud však dojde během zpracování úkonu k chybě, zobrazí se zpráva o neúspěšném nalezení cest, doplněná o informaci, jaká chyba nastala.

5.4 Nalezení souběžných cest

Tato operace slouží k nalezení vzájemně disjunktních kombinací unikátních cest. Z toho vyplívá, že se nejdříve musí nalézt unikátní cesty (podkapitola 5.3). Celý proces hledání kombinací začíná kontrolou, zda jsou k dispozici unikátní cesty. Pokud ano, započne iterace přes všechny unikátní cesty, pro které se začnou vytvářet možné kombinace s dalšími cestami. Tento proces probíhá paralelně, tedy ve více vláknech současně (Zdrojový kód 1). Díky tomu se výrazně snížila doba výpočtu.

```
public string DisjointPairs()
{
    try
    {
        // Kontrola, zda byly nalezeny unikátní cesty – pokud ne,
        // nelze pokračovat
```

```
            if (uniquePaths.GetUniquePaths().Count == 0)
            {
                return "Nejdříve je potřeba nalézt unikátní cesty.";
            }

            // Vymazání předchozích disjunktních n-tic
            disjointPairs.Clear();

            // Pro každou cestu spustí paralelní výpočet
            // disjunktních n-tic
            Parallel.ForEach(uniquePaths.GetUniquePaths().Keys, pathKey =>
            {
                // Inicializace nové n-tice s jednou cestou
                var pair = new HashSet<string> { pathKey };
                // Zahájení rekurzivního vytváření disjunktních
                // n-tic
                MakePairs(pair);
            });
```

Zdrojový kód 1: Ukázka metody DisjointPairs() s více vláknovou iterací přes unikátní cesty [Zdroj vlastní]

Poté je rekurzivně volána metoda *MakePairs()* (Zdrojový kód 2), ve které se k aktuální kombinaci cest postupně přidávají další cesty a pomocí metody *ArePathsDisjoint()* (Zdrojový kód 3) se testuje, zda jsou cesty v nové kombinaci disjunktní.

```
    private void MakePairs(HashSet<string> pair)
    {
        try
        {
            // Spuštění paralelního zpracování přes všechny unikátní cesty
            Parallel.ForEach(uniquePaths.GetUniquePaths(), path =>
            {
                // Kontrola, že nová cesta ještě v n-tici není
                // a cesty v n-tici s jsou novou cestou vzájemně
                // disjunktní
                if (!pair.Contains(path.Key) && ArePathsDisjoint(pair,
                                                path.Value.Nodes))
                {
                    // Vytvoření nové n-tice jako kopie původní,
                    // rozšířená o novou cestu
                    var newPair = new HashSet<string>(pair) { path.Key };
                    // Kontrola, jestli nová n-tice už existuje
                    if (IsPairUnique(newPair))
                    {
                        lock (disjointPairs)
                        {
                            // Vytvoření nového ID
                            var pairId = "P" + disjointPairs.Count;
                            // Uložení nové disjunktní n-tice do
                            // slovníku
                            disjointPairs.TryAdd(pairId, new
                                        DisjointPair(pairId, newPair));
                        }

                        // Rekurzivně se pokračuje ve vytváření nových
                        // kombinací na základě právě vytvořené n-tice
                        MakePairs(newPair);
                    }
                }
            });
        }
        catch (Exception e)
```

```
        {
            // V případě chyby se vyhodí výjimka s popisem chyby
            throw new Exception("Chyba při vytváření n-tic: " + e.Message);
        }
    }
```

Zdrojový kód 2: Ukázka metody MaketPairs() [Zdroj vlastní]

Pokud jsou cesty vzájemně disjunktní, metodou *IsPairUnique()* (Zdrojový kód 4) se následně zkontroluje, zda daná kombinace již nebyla nalezena. Pakliže ne, nová kombinace je uložena do slovníku společně s unikátním ID a proces pokračuje rekurzivně dál s touto novou kombinací. Pokud by se během kontroly zjistilo, že cesty v kombinaci nejsou disjunktní, nebo že daná kombinace již existuje, hledání se pro danou kombinaci ukončí a přejde se na další.

```
    private bool ArePathsDisjoint(HashSet<string> pair, List<Vertex> path)
    {
        try
        {
            // Získají se všechny unikátní cesty
            var paths = uniquePaths.GetUniquePaths();

            // Pro každý člen z kolekce (pair) se ověří, že jeho cesta nemá
            // žádný společný vrchol s danou cestou `path`
            // Pokud se žádná taková kolize nenajde, vrátí se TRUE (cesty
            // jsou disjunktní)
            return pair.All(memberOfPair =>
                    !paths[memberOfPair].Nodes.Intersect(path).Any());
        }
        catch (Exception e)
        {
            // V případě chyby se vyhodí výjimka s popisem chyby
            throw new Exception("Chyba při zjišťování disjunktnosti cest n-
                    tice: " + e.Message);
        }
     }
```

Zdrojový kód 3: Ukázka metody ArePathsDisjoint() [Zdroj vlastní]

Po dokončení procesu hledání je výsledkem množina unikátních disjunktních n-tic a uživateli se zobrazí zpráva o úspěšném dokončení operace: V případě, že by došlo k chybě, zobrazí se zpráva o neúspěšné operaci a k jaké chybě došlo.

```
    private bool IsPairUnique(HashSet<string> newPair)
    {
        try
        {
            // Prochází všechny již existující disjunktní n-tice a ověřuje,
            // zda se žádná z nich nerovná právě vytvářené n-tici
            // (porovnání množinově). Pokud je nová n-tice unikátní, vrací
            // TRUE
            return !disjointPairs.Values.Any(existingPair =>
                        existingPair.Paths.SetEquals(newPair));
        }
        catch (Exception e)
        {
            // V případě chyby se vyhodí výjimka s popisem chyby
```

```
                throw new Exception("Chyba při zjišťování unikátnosti n-tice: "
                                                    + e.Message);
            }
        }
```

Zdrojový kód 4: Ukázka metody IsPairUnique() [Zdroj vlastní]

5.5 Nalezení nejdelší cesty

Pro nalezení nejdelší cesty, která musí začínat a končit v hraničním vrcholu, je použit modifikovaný Dijkstrův algoritmus s prioritní frontou implementovanou pomocí binární haldy (podrobně popsán v kapitole 3.1.1). Algoritmus se aplikuje na každý hraniční vrchol, pro který vytvoří strom nejdelších vzdáleností do každého vrcholu grafu, který je z daného hraničního vrcholu dosažitelný.

Po dokončení algoritmu se pro daný hraniční vrchol ve vytvořeném stromě vyhledá hraniční vrchol s největší vzdáleností a porovná se s aktuálně nejdelší vzdáleností vrcholu v rámci celého grafu. Pokud je nová vzdálenost vrcholu větší, provede se pro příslušný vrchol rekonstrukce cesty a je nastavena jako nová nejdelší cesta v grafu.

Výstupem této operace je ID unikátní cesty (podkapitola 5.3), které nejdelší cestě odpovídá, délka a posloupnost vrcholů, které cestu tvoří. V případě, že by kdykoliv během běhu algoritmu došlo k chybě, je proces hledání nejdelší cesty ukončeno a uživatel je informován o tom, k jaké chybě došlo a co ji způsobilo.

V Tabulce 7 je popsán pseudokód modifikovaného Dijkstrova algoritmu pro nalezení nejdelší cesty.

Tabulka 7: Pseudokód použitého Dijkstrova algoritmu [Zdroj vlastní]

```
01 function TheLongestPath()
02 |  try
03 |  |  // kontrola, zda byly nalezeny unikátní cesty
04 |  |  if UniquePaths.GetUniquePaths() is empty then
05 |  |  |  return "Nejdříve je potřeba nalézt unikátní cesty."
06 |  |  end
07 |  |
08 |  |  text ← ""
09 |  |  // hledání nejdelší cesty pro každý hraniční vrchol
10 |  |  foreach vertex in Graph.GetInputVertices() do
11 |  |  |  GoThroughNodes(vertex.Key)
12 |  |  end
13 |  |
14 |  |  foreach path in UniquePaths.GetUniquePaths() do
```

```
15 |  |  |  // průnik vrcholů nejdelší cesty s unikátní cestou
16 |  |  |  commonVertices ← path.Value.Nodes ∩ TheLongestPath.Value
17 |  |  |  // pokud je počet vrcholů nejdelší a aktuální cesty shodný, vypíše se ID cesty
18 |  |  |  if commonVertices contains all path.Value.Nodes then
19 |  |  |  |  text ← "Nejdelší cesta je " + path.Key
20 |  |  |  |  break
21 |  |  |  end
22 |  |  end
23 |  |
24 |  |  return text + " s délkou " + TheLongestPath.Key
25 |  catch exception e do
26 |  |  return "Chyba při hledání nejdelší cesty: " + e.Message
27 |  end
28 end
29
30 function GoThroughNodes(startKey)
31 |  try
32 |  |  distances ← empty map<Vertex, double>   // inicializace mapy vzdáleností vrcholů
33 |  |  // inicializace mapy pro ukládání předchůdců pro rekonstrukci nejdelší cesty
34 |  |  previous ← empty map<Vertex, Vertex>
35 |  |  // inicializace prioritní fronty
36 |  |  priorityQueue ← new PriorityQueue<(Vertex, Edge), double>
37 |  |
38 |  |  foreach vertex in Graph.GetVertices() do
39 |  |  |  distances[vertex] ← -∞  // nastavení počátečních vzdáleností vrcholů na nekonečno
40 |  |  |  previous[vertex] ← null  // nastavení předchůdců vrcholů
41 |  |  end
42 |  |
43 |  |  // vložení startovního hraničního vrcholu do fronty
44 |  |  startVertex ← Graph.GetVertices().First(v → v.Key = startKey)
45 |  |  distances[startVertex] ← 0        // nastavení vzdálenosti vrcholu na 0
46 |  |  // vložení prvku do fronty bez hrany a s nulovou vzdáleností
47 |  |  priorityQueue.Enqueue((startVertex, null), 0)
48 |  |
49 |  |  while priorityQueue is not empty do
50 |  |  |  // vybrání vrcholu z fronty
51 |  |  |  (currentVertex, arrivalEdge) ← priorityQueue.Dequeue()
52 |  |  |  currentDistance ← distances[currentVertex]      // vybrání vzdálenosti vrcholu
53 |  |  |
54 |  |  |  // pokud je vybraný vrchol hraniční a nemá příchozí hranu, vrchol se přeskočí
55 |  |  |  if currentVertex is in Graph.GetInputVertices() and arrivalEdge ≠ null then
56 |  |  |  |  continue
57 |  |  |  end
58 |  |  |
```

```
59          // pro každou incidenční hranu vybraného vrcholu
60          foreach edge in currentVertex.Edges do
61              neighbor ← edge.Destination      // vybrání koncového vrcholu hrany
62
63               // pokud je vybrána hrana shodná s příchozí hranou do currentVertex,
64               //přeskočí se
65              if arrivalEdge ≠ null and edge.Id = arrivalEdge.Id then
66                  continue
67              end
68
69              // pokud je hrana na seznamu zakázaných odbočení pro příchozí hranu,
70               //přeskočí se
71              if currentVertex.UnreachablesVertices contains (arrivalEdge.Id, edge.Id) then
72                  continue
73              end
74
75               //nová vzdálemost sousedního vrcholu
76              newDistance ← currentDistance + edge.Length
77
78              if newDistance > distances[neighbor] then    // pokud je nová vzdálenost větší
79                  distances[neighbor] ← newDistance        // nastavení nové vzdálenosti
80                  // uložení předchůce sousedního vrcholu
81                  previous[neighbor] ← currentVertex
82                  // vložení sousedního vrcholu do fronty
83                  priorityQueue.Enqueue((neighbor, edge), -newDistance)
84              end
85          end
86      end
87
88      // vybrání vrcholu s největší vzdáleností, který je zároveň hraničním vrcholem
89      farthestVertexKey ← max({pair.Key | pair in distances where pair.Key in
90      Graph.GetInputVertices()})
91
92      // pokud je vzdálenost vybraného vrcholu větší než dosavadní celkově nejdelší
93      //vzdálenost
94      if TheLongestPath.Key < distances[farthestVertexKey] then
95          path ← empty list<Vertex>    // rekonstrukce nové nejdelší cesty
96          at ← Graph.GetVertices().First(v → v.Key = farthestVertexKey.Key)
97          while at ≠ null do
98              path.Add(at)    //vložení aktuálního vrcholu do cesty
99               at ← previous[at]     // posunutí se k předhcůdci aktuálního vrcholu
100         end
101          path.Reverse()    // otočení cesty, aby začínala v počátečním hraničním vrcholu
102
```

```
103 |  |  |  // uložení nové nejdelší cesty
104 |  |  |  TheLongestPath ← (distances[farthestVertexKey], path)
105 |  |  end
106 |  catch exception e do
107 |  |  throw "Chyba při určování vzdálenosti uzlů od počátečního uzlu: " + e.Message
108 |  end
109 end
```

Modifikace Dijkstrova algoritmu (podrobně popsány v podkapitole 3.1.1) jsou specifikovány v pseudokódu (Tabulka 7) na řádcích 39, 83, 89, 94. Na řádku 39 se nastavují vzdálenosti všech vrcholů na -∞. Na řádku 83 se vkládá vrchol do fronty se zápornou vzdáleností. Na řádku 89 se vybírá hraniční vrchol s největší vzdáleností a nakonec na řádku 94 se porovnává, zda je nově nalezená nejdelší vzdálenost vrcholu větší než dosavadní největší vzdálenost vrcholu v grafu.

Důvody, proč lze tento modifikovaný Dijkstrův algoritmus použít na neorientovaný graf, jsou, že díky uplatnění zakázaných odbočení na výhybkách se netvoří v grafu smyčky a není tím pádem potřeba je kontrolovat. Dalším důvodem je, že se jedná o řídký graf, jelikož každý vrchol je incidenční nanejvýš s třemi hranami.

5.6 Uložení dat

Při ukládání dat se ukládají unikátní cesty, n-tice souběžných cest a nejdelší cesta. Všechny tři typy cest se ukládají do samostatných .csv souborů: *UniquePaths.csv*, *UniquePairs.csv*, *LongestPath.csv*.

Pro každý typ cesty se ukládají jiná data. V případě, že by například nebyly ještě nalezeny n-tice souběžných cest, vytvoří se příslušný soubor pouze s názvy sloupců, avšak nebudou v nich žádná data.

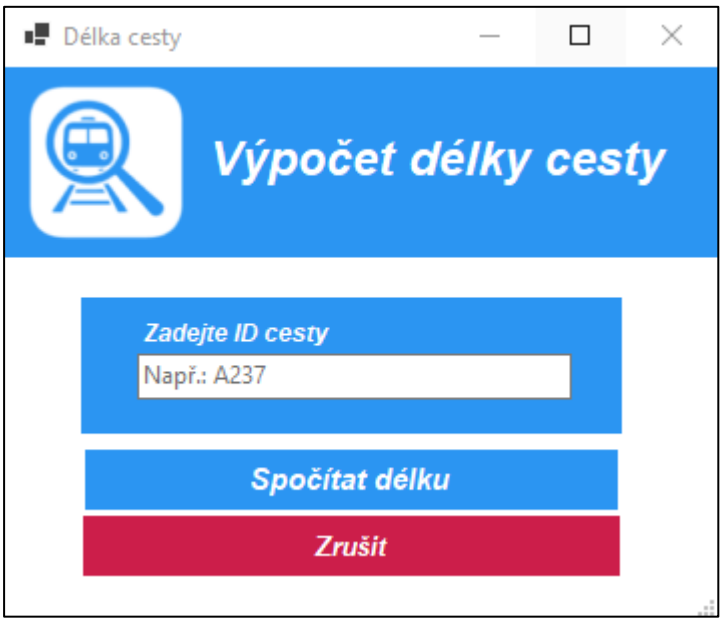
Pro unikátní cesty se ukládá ID, délka cesty a posloupnost všech vrcholů, přes které cesta prochází. Pro n-tice souběžných cest se uloží ID a seznam ID jednotlivých cest, které tuto n-tici tvoří. Pro nejdelší cestu se uloží její celková délka a seznam vrcholů, které cestu tvoří.

Pakliže uložení všech dat proběhne správně, vypíše se v informačním poli zpráva, že uložení dat proběhlo úspěšně. Pokud se ovšem vyskytne chyba, informuje se o tom uživatel spolu s tím, co chybu způsobilo.

5.7 Výpočet délky cesty

Při zvolení možnosti *Vypočítat délku cesty* se otevře nové formulářové okno s textovým polem pro zadaní ID cesty (Obrázek 18). Podle zadaného ID se vyhledá odpovídající unikátní cesta a do informačního pole se vypíše její délka. Tato délka je již vypočítána od procesu nalezení unikátních cest (podkapitola 5.3) a proto se znovu nepočítá.

Pokud by cesta se zadaným ID neexistovala, uživateli je zobrazena zpráva informující o této skutečností.



Obrázek 18: Formulář pro zadaní ID cesty pro výpočet její délky [Zdroj vlastní]

5.8 Vizualizace

Pro vizualizaci je vytvořeno nové formulářové okno se dvěma textovými poli pro zadání ID cesty a ID n-tice souběžných cest. Lze vizualizovat jedno nebo oboje najednou. Na základě zadaného ID se vyhledá požadovaná cesta nebo n-tice. Pokud by prvek neexistoval, zobrazí se chybová zpráva a vykreslování se ukončí. Kdyby se vykreslovala cesta i n-tice najednou, a například u vykreslování cesty by nastala chyba, ukončí se pouze vykreslovaní cesty a přejde se na vykreslování n-tice.

Proces vykreslení cesty začíná načtením cesty podle zadaného ID, načtením všech vrcholů grafu a přípravou kreslicí plochy s pevně danými rozměry 20 000 × 5 000 pixelů. Následně jsou definovány barvy vrcholů, hran a styly kreslení. Běžné hrany jsou černé, popis vrcholů je černý, běžné vrcholy jsou černé a hrany a vrcholy spadající do zadané cesty jsou odlišeny jinou barvou, v tomto případě hrany oranžovou a vrcholy žlutou. Před samotným vykreslením se spočítají minimální a maximální souřadnice vrcholů, aby bylo možné určit

vhodné měřítko pro přizpůsobení grafu do připravené plochy.

Vrcholy jsou vykreslovány postupně, přičemž pro každý vrchol se podle jeho X, Y souřadnic spočítá jeho pozice v upraveném měřítku. K vykreslenému bodu je přidán i textový popis, který obsahuje ID vrcholu. Hrany jsou vykresleny jako spojnice vrcholů, přičemž ty, které patří k vybrané cestě, jsou zobrazeny oranžově, zatímco ostatní zůstávají černé.

V případě vizualizace n-tice souběžných cest je postup téměř totožný. Místo jedné cesty se akorát vyhledá množina cest podle zadaného ID n-tice. Pokud není nalezena žádná odpovídající n-tice s tímto ID, zobrazí se chybová zpráva a vykreslování se ukončí. Pro každou cestu množiny se vygeneruje unikátní barva, která ji odliší od ostatních. Hrany a vrcholy jsou vykresleny stejným způsobem jako při vizualizaci jedné cesty, jen zde, pokud hrany náleží cestám ze zadané množiny, vykreslí se příslušnou unikátní barvou. Všechny ostatní běžné hrany jsou vykresleny černě.

Po dokončení vykreslování se výsledný obrázek uloží do odpovídajícího souboru *PairPicture.png* nebo *PathPicture.png* a vypíše se zpráva o úspěšném dokončení vizualizace. Pokud během procesu dojde k chybě, je uživatel o problému informován a vizualizace je ukončena. Totéž se provede i po dokončení vykreslování jedné cesty. Na Obrázku 19 je formulářové okno pro zadání ID cesty a n-tice souběžných cest.

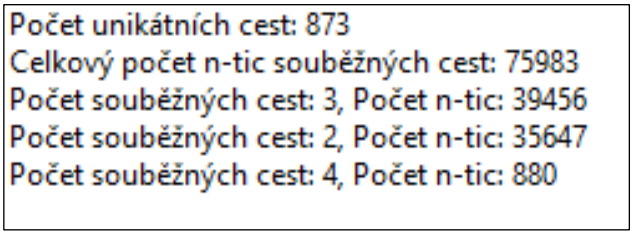
Obrázek 19: Formulář pro zadání ID cesty a n-tice [Zdroj vlastní]

V Příloze B jsou poté k vidění 2 vizualizace. Jedna je vizualizace modelu se zvýrazněnou unikátní cestou a druhá je vizualizace modelu se zvýrazněnou n-ticí souběžných cest.

5.9 Statistiky hledání

Tato operace vrací informace o tom, kolik unikátních cest bylo v modelu infrastruktury nalezeno a kolik bylo vytvořeno n-tic souběžných cest. Pro n-tice souběžných cest je vrácen celkový počet n-tic a poté počet n-tic podle typu, což znamená, kolik bylo nalezeno dvojic, trojic, čtveřic atd.

V případě, že by byly dosud nalezeny pouze unikátní cesty, statistiky vypíšou pouze počet unikátních cest. Na Obrázku 20 je ilustrace výpisu statistik v aplikaci.



Obrázek 20: Statistiky hledání unikátních a souběžných cest [Zdroj vlastní]

6 APLIKACE ALGORITMŮ NA MODELU REÁLNÉ ŽELEZNIČNÍ INFRASTRUKTURY

6.1 Popis použitého modelu

Model infrastruktury, který je v rámci této práce použit, odráží kolejiště hlavního nádraží Pardubice. Model byl vytvořen v nástroji MesoRail (podrobně popsán v podkapitole 4.1.4.). Jsou v něm obsaženy koleje, jednoduché výhybky a plné křižovatkové výhybky, přičemž každá kolej a vrchol má doplňující detaily, jako ID, délka, souřadnice, typ atd. V Příloze A jsou dva modely. První je kompletní model infrastruktury i s traťový kolejemi. Druhý model je upravený a jsou odstraněny traťové koleje, které pro účely práce nejsou potřeba. V modelu jsou také označeny hraniční vrcholy: $V_{branch}(G)$ = {273_1, 264_1, 319_1, 316_1, 538_1, 556_1, 172_1, 168_1, 171_1}

6.2 Testovací scénáře

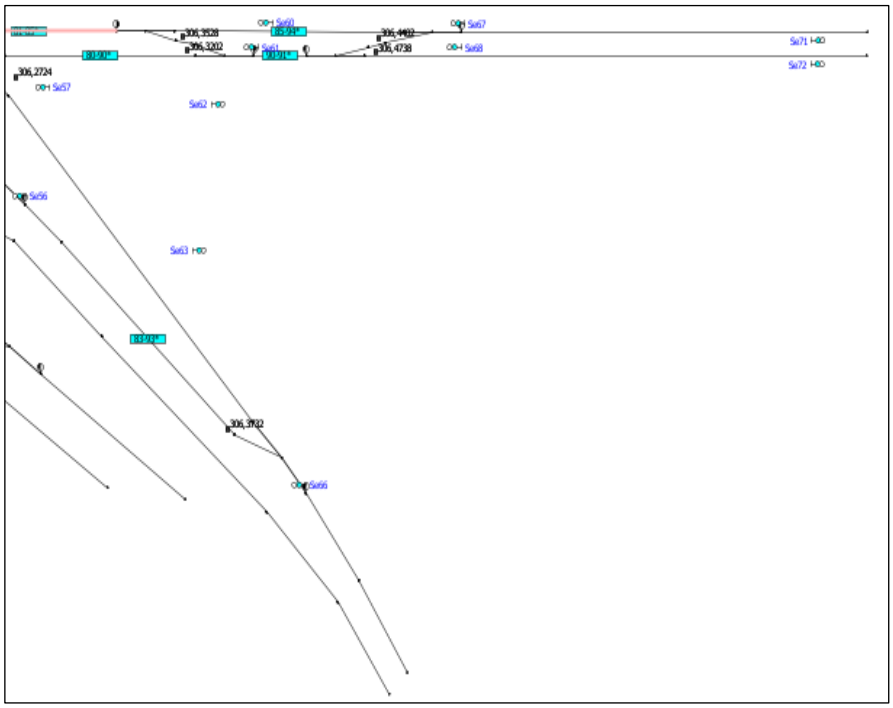
Při hledání souběžných a nejdelší cest byly na model infrastruktury aplikovány různé scénáře s cílem zjistit, jak výluky kolejí mohou ovlivnit počty unikátních a souběžných cest a délku a trasu nejdelší cesty. Tyto scénáře reprezentují reálné situace, ke kterým v železničním provozu běžně dochází. Scénáře jsou do modelu infrastruktury zaneseny tak, že koleje, kterých se scénář týká, jsou z modelu odstraněny.

6.3 Scénář Sc01: Běžný provoz

V tomto scénáři je použit plný model infrastruktury a není zde žádné omezení. Model lze vidět v Příloze A.

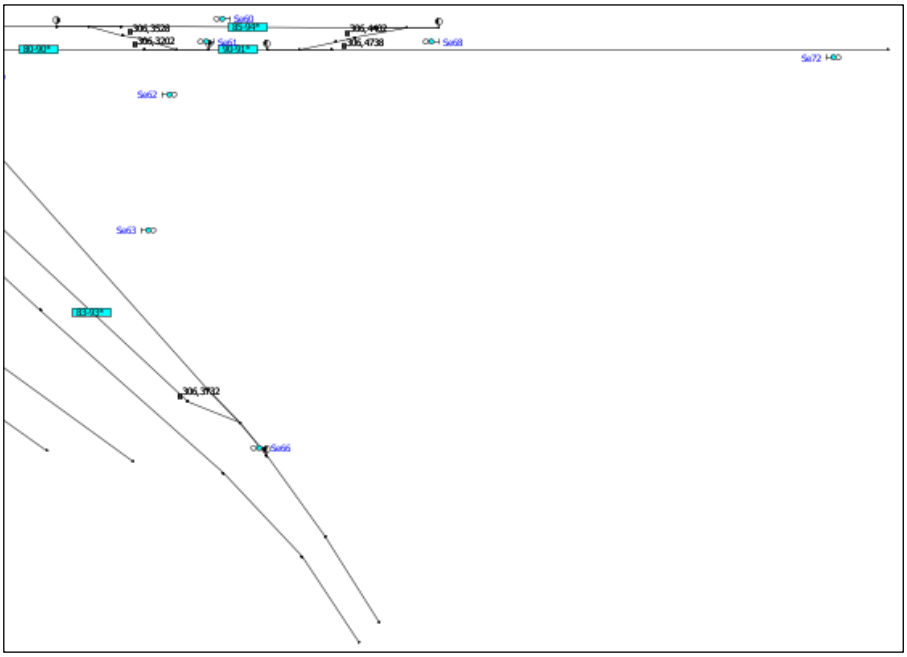
6.4 Scénář Sc02: Neprůjezdnost traťové koleje

Z důvodu poruchy lokomotivy došlo k zablokování jedné traťové koleje, což znamená, že v modelu ubyl jeden hraniční vrchol. Množina hraničních vrcholů je potom: $V_{branch}(G)$ = {264_1, 319_1, 316_1, 538_1, 556_1, 172_1, 168_1, 171_1}. Na Obrázku 21 je část infrastruktury s průjezdnou traťovou kolejí.



Obrázek 21: Část infrastruktury s průjezdnou traťovou kolejí [Zdroj vlastní]

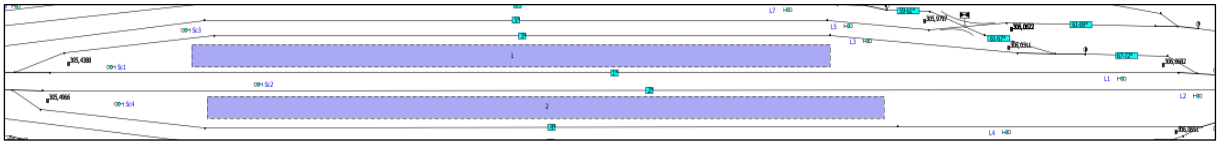
Na Obrázku 22 je část infrastruktury s neprůjezdnou traťovou kolejí a bez hraničního vrcholu.



Obrázek 22: Část infrastruktury s neprůjezdnou traťovou kolejí [Zdroj vlastní]

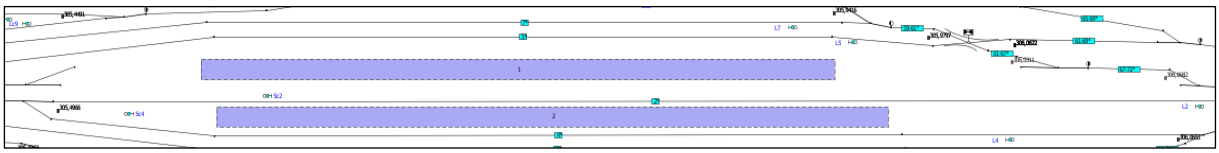
6.5 Scénář Sc03: Odstávka nástupištních kolejí

Kvůli rozsáhlým opravám nástupiště bylo z bezpečnostních důvodů, jako ochrana pracovníků, cestujících a zamezení incidentů s projíždějícími vlaky, uzavřeno nástupiště a s ním i obě nástupištní koleje. Na Obrázku 23 je část infrastruktury s funkčním nástupištěm.



Obrázek 23: Část infrastruktury s funkčním nástupištěm a kolejemi [Zdroj vlastní]

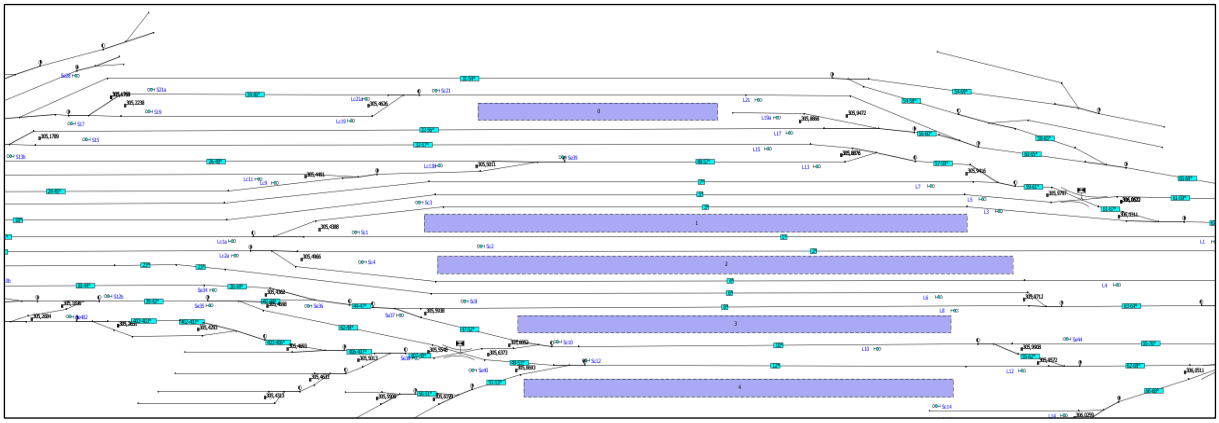
Na Obrázku 24 je část infrastruktury po uzavření nástupištních kolejí.



Obrázek 24: Část infrastruktury s uzavřenými nástupištními kolejemi [Zdroj vlastní]

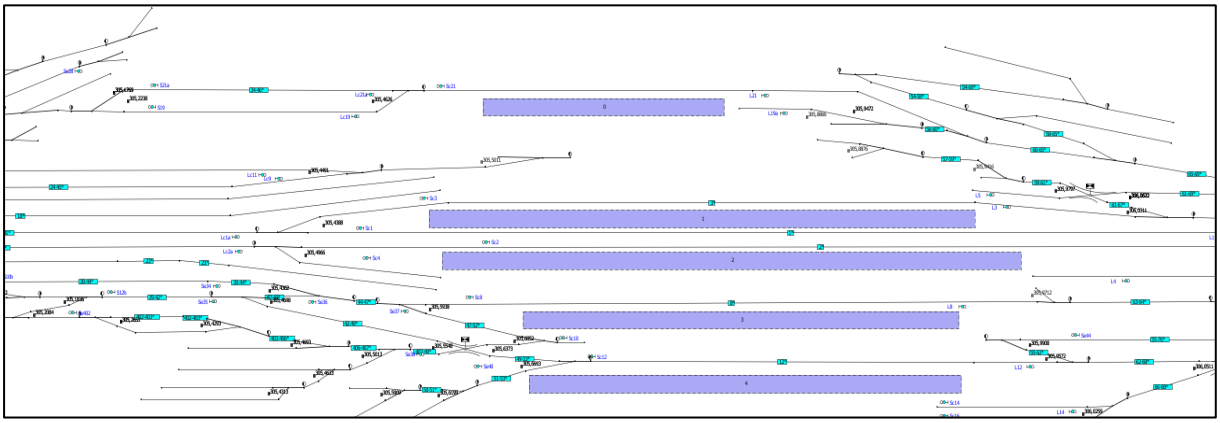
6.6 Scénář Sc04: Snížená kapacita infrastruktury

Z důvodu prací na traťové koleji musel být ve stanici odstaven velký počet nákladních vlaků, což značně snížilo kapacitu infrastruktury a možnosti průjezdu stanicí. Na Obrázku 25 je vidět infrastruktura s plnou průjezdností.



Obrázek 25: Infrastruktura s plnou průjezdností [Zdroj vlastní]

Na Obrázku 26 je infrastruktura se značně omezenou průjezdností.



Obrázek 26: Infrastruktura s omezenou průjezdností [Zdroj vlastní]

6.7 Ukázky výsledků

Aplikace byla spuštěna s připravenými scénáři (podkapitola 6.2) a pro každý z nich byla zjištěna nejdelší cesta a statistiky souběžných a unikátních cest.

6.7.1 Scénář Sc01

Tabulka 8: Výsledky hledání cest pro scénář Sc01 [Zdroj vlastní]

Nejdelší cesta	A397
Délka nejdelší cesty	2887,36 metrů
Počet unikátních cest	873
Počet n-tic souběžných cest	75983
Počet dvojic	35647
Počet trojic	39456
Počet čtveřic	880

Vizualizace nejdelší cesty je k dispozici v Příloze C.

6.7.2 Scénář Sc02

Tabulka 9: Výsledky hledání cest pro scénář Sc02 [Zdroj vlastní]

Nejdelší cesta	A249
Délka nejdelší cesty	2887,23 metrů
Počet unikátních cest	474
Počet n-tic souběžných cest	20633
Počet dvojic	12919
Počet trojic	7714

Vizualizace nejdelší cesty je k dispozici v Příloze C.

6.7.3 Scénář Sc03

Tabulka 10: Výsledky hledání cest pro scénář Sc03 [Zdroj vlastní]

Nejdelší cesta	A367
Délka nejdelší cesty	2887,36 metrů
Počet unikátních cest	813
Počet n-tic souběžných cest	66941
Počet dvojic	33055
Počet trojic	33366
Počet čtveřic	520

Vizualizace nejdelší cesty je k dispozici v Příloze C.

6.7.4 Scénář Sc04

Tabulka 11: Výsledky hledání cest pro scénář Sc04 [Zdroj vlastní]

Nejdelší cesta	A80
Délka nejdelší cesty	2815,34 metrů
Počet unikátních cest	209
Počet n-tic souběžných cest	3002
Počet dvojic	2158
Počet trojic	836
Počet čtveřic	8

Vizualizace nejdelší cesty je k dispozici v Příloze C.

7 ANALÝZA VÝSLEDKŮ

7.1 Shrnutí dosažených výsledků

Srovnání výsledků z testování scénářů (podkapitola 6.7) ukazuje, že maximálních možných kapacit infrastruktury je dosaženo za běžného provozu (Sc01).

Naopak nejvýraznější dopady na provoz mají výluka traťové koleje (Sc02) a snížená kapacita infrastruktury (Sc04). Při výluce traťové koleje a tím pádem výpadku hraničního vrcholu došlo k dramatickému poklesu unikátních cest o 45,7 % a souběžných cest o 72,85 %, nicméně trasa nejdelší cesty zůstala téměř stejná jako v běžném provozu a její délka se zkrátila jen o 0,13 metru.

K největším dopadům na provoz došlo při snížené kapacitě infrastruktury. Počet unikátních cest se snížil téměř na čtvrtinu, konkrétně o 76,63 %, a počet souběžných poklesl o necelých 96,05 % oproti běžnému provozu. Změnila se i trasa nejdelší cesty, která je kvůli tomu kratší o 72 metrů oproti té z běžného provozu.

Odstávka nástupištních kolejí (Sc03) měla negativní dopad na provoz, nicméně se nejednalo o nijak zásadní dopad. Počet unikátních cest kles 6,76 % a počet souběžných cest o 11,9 % a trasa i délka nejdelší cesty zůstala nezměněna.

Ze shrnutí lze vidět, že železniční infrastruktura je citlivá na výluky tratových kolejí a s tím spojených hraničních vrcholů a na výluky výrazného počtu dostupných staničních kolejí.

7.2 Přínos práce pro praxi

Aplikace pro hledání unikátních a souběžných cest může mít pro železniční sektor značné přínosy. Hlavní přínosy budou zejména v oblasti optimalizace provozu, a to díky schopnosti aplikace analyzovat všechny dostupné cesty, identifikovat souběžné cesty a vyhodnotit dopady omezení v provozu. V návaznosti na tato omezení aplikace rovnou navrhne i alternativní cesty, což může zmírnit negativní dopady omezení. Díky identifikaci souběžných cest lze lépe koordinovat provoz a optimalizovat trasy vlaků za účelem zkrácení času průjezdu stanicí, případně snížení zpoždění.

Druhou oblastí přínosu bude plánování. Tam může aplikace pomoci při plánování výluk tratí tak, aby byl co nejmenší dopad na provoz. Využití najde také při testování krizových scénářů, analýze pro detekci kritických bodů v železniční síti nebo při vytváření jízdních řádů a rozhodování o úpravách a rozvoji infrastruktury.

Celkově může aplikace představovat užitečný nástroj pro podporu řízení a rozvoje železniční dopravy, který napomáhá maximalizovat využití kapacity sítě a zajistit efektivnější

a spolehlivější železniční provoz.

ZÁVĚR

Cílem této práce bylo vytvořit algoritmus pro vyhledávání souběžných a nejdelších cest v modelech kolejové železniční infrastruktury a následnou vizualizaci nalezených cest. Tento hlavní cíl i další cíle stanovené v úvodu práce byly úspěšně naplněny.

V teoretické části byl popsán úvod do problematiky vytváření modelů železniční infrastruktury a zavedení klíčových pojmů, jako jsou hraniční vrchol nebo zakázaná odbočení, a dále byly představeny různé typy výhybek a způsob reprezentace infrastruktury pomocí neorientovaného grafu. Poté byly definovány souběžné a nejdelší cesty a principy jejich vyhledání. Závěrem této části byly představeny algoritmy, které je možné k hledání cest použít.

V praktické části bylo dosaženo hlavního cíle, a to vytvoření aplikace, která využívá navržené algoritmy pro hledání souběžných a nejdelších cest a jejich následnou vizualizaci. Dále je zde popsána implementace aplikace, použité technologie a podrobný rozbor jednotlivých funkcí aplikace. Nakonec jsou zde prezentovány výsledky z hledání souběžných a nejdelších cest v různých scénářích aplikovaných na model železniční infrastruktury a jejich následný rozbor a shrnutí.

I přes splnění stanovených cílů existují oblasti, které by bylo možné v budoucnu dále rozšířit a zlepšit. Jedná se především o optimalizaci procesu hledání souběžných cest. Další možností rozšíření aplikace je přidání funkce umožňující si v aplikaci definovat vrchol nebo vrcholy, mezi kterými se budou souběžné a nejdelší cesty hledat.

POUŽITÁ LITERATURA

[1] KAVIČKA, Antonín a BAŽANT, Michael. Návrh infrastruktury železničních uzlů (část 2). Online. *Automa*. 2007, roč. 13, č. 08. ISSN 1210-9592. Dostupné z: https://www.automa.cz/cz/casopis-clanky/navrh-infrastruktury-zeleznicnich-uzlu-cast-2-2007_08_34407_753/. [cit. 2025-04-06].

[2] SPRÁVA ŽELEZNIC. *Železniční svršek Díl IX Výhybky a výhybkové konstrukce*. 3. Praha: Správa železnic, 2021. Dostupné také z: https://www.spravazeleznic.cz/documents/50004227/139626480/SZDC_S3_Dil_09_sezm1az4_20210301.pdf/c368c5d0-e57a-48f2-b588-92b706c2ef46?version=3.0.

[3] ZEMAN, Zdeněk. *Výhybkový přestavník pro modelovou železnici*. Online, Diplomová práce, vedoucí Jiří Poucha. Plzeň: Západočeská univerzita v Plzni, Fakulta elektrotechnická, 2013. Dostupné z: https://theses.cz/id/qta8tp/. [cit. 2025-04-11].

[4] KAVIČKA, Antonín a DIVIŠ, Roman. Dynamic Search of Train Shortest Routes Within Microscopic Traffic Simulators. Online. *IEEE Access*. 2022, vol. 10, s. 90163-90199. ISSN 2169-3536. Dostupné z: IEEE Xplore, https://doi.org/10.1109/ACCESS.2022.3197660. [cit. 2025-04-06].

[5] ČESKÝ STATISTICKÝ ÚŘAD. *Jak vzniká vlakový jízdní řád? Roli hrají nároky dopravců, kapacita železnic a data ze sčítání*. Online. ČESKÝ STATISTICKÝ ÚŘAD. Český statistický úřad. 2021, 29. 01. 2021. Dostupné z: https://csu.gov.cz/produkty/jak-vznika-vlakovy-jizdni-rad-roli-hraji-naroky-dopravcu-kapacita-zeleznic-a-data-ze-scitani. [cit. 2025-04-11].

[6] ČESKÉ DRÁHY. *Jak se dělá jízdní řád*. Online. ČESKÉ DRÁHY. České dráhy. 2016. Dostupné z: https://www.cd.cz/jizdni-rad/-25517/. [cit. 2025-04-11].

[7] JANOŠ, Vít. *Technologie dopravy a logistika: Plánování nabídky ve veřejné dopravě I*. Online. Praha: ČVUT v Praze, Fakulta dopravní, 2017. Dostupné z: https://zolotarev.fd.cvut.cz/ma/ctrl.php?act=show,file,24669. [cit. 2025-04-11].

[8] KOVÁŘ, Petr. *Teorie grafů*. Online. Ostrava: VŠB – Technická univerzita Ostrava, 2025. Dostupné z: https://homel.vsb.cz/~kov16/files/skriptum_teorie_grafu.pdf. [cit. 2025-04-11].

[9] RYBANSKÝ, Marian. *Určení velikosti stavového prostoru*. Online, Bakalářská práce, vedoucí Jan Roupec. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2016. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=128758. [cit. 2025-04-11].

[10] KOLÁŘ, Petr. *Úvod do Teorie grafů*. Online. Ostrava: VŠB - Technická univerzita Ostrava, 2021. Dostupné z: https://homel.vsb.cz/~kov16/files/uvod_do_teorie_grafu.pdf. [cit. 2025-04-11].

[11] TPOINT TECH. *Single Source Shortest Path in a Directed Acyclic Graphs*. Online. Tpoint Tech. 2011 - 2025. Dostupné z: https://www.tpointtech.com/single-source-shortest-path-in-a-directed-acyclic-graphs. [cit. 2025-04-11].

[12] KRÁĽ, Dan; MAREŠ, Martin a ŠKODA, Petr. *Haldy, heapsort a Dijkstrův algoritmus*. Online. Programátorská encyklopedie KSP. Září 2023, listopad 2022. Dostupné z: https://ksp.mff.cuni.cz/encyklopedie/halda-a-cesty/. [cit. 2025-04-11].

[13] OTT, Lukáš. *Grafy, grafové algoritmy a jejich využití při hledání nejkratší cesty*. Online, Bakalářská práce, vedoucí Martina Bobalová. Brno: Vysokého učení technické v Brně, Fakulta podnikatelská, 2010. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=31080. [cit. 2025-04-20].

[14] HAMERNÍK, Michal. *Grafy a algoritmy pro hledání nejkratších cest*. Online, Bakalářská práce, vedoucí Martina Bobalová. Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2009. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=14723. [cit. 2025-04-20].

[15] PAVLÁSEK, Ondřej. *Grafy a grafové algoritmy*. Online, Bakalářská práce, vedoucí Martina Bobalová. Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2010. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=31232. [cit. 2025-04-20].

[16] JÁGR, Petr. *Hledání nejkratších cest grafem*. Online, Bakalářská práce, vedoucí Jiří Jaroš. Brno: Vysoké učení technické v Brně, Fakulta informačních technologií, 2007. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=115691. [cit. 2025-04-20].

[17] RUDOVÁ, Hana. *PB165 – Grafy a sítě: Hledání nejkratších cest*. Online. Brno: Masarykova univerzita, 2011. Dostupné z: https://is.muni.cz/el/fi/podzim2011/PB165/um/4.pdf. [cit. 2025-04-20].

[18] TPOINT TECH. *A* Search Algorithm in Artificial Intelligence*. Online. Tpoint Tech. 17 Mar 2025. Dostupné z: https://www.tpointtech.com/ai-informed-search-algorithms. [cit. 2025-04-21].

[19] BLAHA, Milan; HOLČÍK, Jiří a KOMENDA, Martin. Metoda A*. Online. In: *Matematická biologie: e-learningová učebnice*. Brno: Masarykova univerzita, 2015. ISBN 978-80-210-8095-9. Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence--prohledavani-stavoveho-prostoru--metody-prohledavani--informovane-heuristicke-prohledavani--metoda-a. [cit. 2025-04-21].

[20] GEEKSFORGEEKS. *A* Search Algorithm*. Online. GeeksforGeeks. 2008, 30 Jul, 2024. Dostupné z: https://www.geeksforgeeks.org/a-search-algorithm/. [cit. 2025-04-21].

[21] BAJER, Lukáš. *Algoritmy pro pathfinding*. Online. Praha: Univerzita Karlova, 2006. Dostupné z: https://diana.ms.mff.cuni.cz/pogamut_files/lectures/2015-2016/04.2-Pathfinding-Bajer060410.pdf. [cit. 2025-04-21].

[22] GEEKSFORGEEKS. *Shortest Path in Directed Acyclic Graph*. Online. GeeksforGeeks. 2008, 03 Feb, 2023. Dostupné z: https://www.geeksforgeeks.org/shortest-path-for-directed-acyclic-graphs/. [cit. 2025-04-21].

[23] MOORE, Karleigh; JENNISON, Ken a KHIM, Jimin. *Depth-First Search (DFS)*. Online. Brilliant. C2025. Dostupné z: https://brilliant.org/wiki/depth-first-search-dfs/. [cit. 2025-04-21].

[24] MCKEE, Amberle. *Depth-First Search in Python: Traversing Graphs and Trees*. Online. DataCamp. 2024. Dostupné z: https://www.datacamp.com/tutorial/depth-first-search-in-python. [cit. 2025-04-21].

[25] DRICHEL, Alexander. *Depth-first-tree*. Online. In: Wikimedia Commons. 28 March 2008. Dostupné z: https://commons.wikimedia.org/wiki/File:Depth-first-tree.svg. [cit. 2025-04-21].

[26] MCKEE, Amberle. *Breadth-First Search in Python: A Guide with Examples*. Online. DataCamp. 2024. Dostupné z: https://www.datacamp.com/tutorial/breadth-first-search-in-python. [cit. 2025-04-21].

[27] GEEKSFORGEEKS. *Breadth First Search or BFS for a Graph*. Online. GeeksforGeeks. 2008, 29 Mar, 2025. Dostupné z: https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/. [cit. 2025-04-21].

[28] *Breadth-first-tree*. Online. In: Maxnilz. Dostupné z: https://maxnilz.com/docs/algo/graph/breadth-first-search/. [cit. 2025-04-21].

[29] GEEKSFORGEEKS. *Applications of Dijkstra's shortest path algorithm*. Online. GeeksforGeeks. 2008, 23 Sep, 2022. Dostupné z: https://www.geeksforgeeks.org/applications-of-dijkstras-shortest-path-algorithm/. [cit. 2025-04-21].

[30] GEEKSFORGEEKS. *Floyd Warshall Algorithm*. Online. GeeksforGeeks. 2008, 16 Apr, 2025. Dostupné z: https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/. [cit. 2025-04-21].

[31] KUMAR, Rajesh. *The A* Algorithm: A Complete Guide*. Online. DataCamp. 2024. Dostupné z: https://www.datacamp.com/tutorial/a-star-algorithm. [cit. 2025-04-21].

[32] MICROSOFT. *C#*. Online. MICROSOFT. NET | Build. Test. Deploy. C2025. Dostupné z: https://dotnet.microsoft.com/en-us/languages/csharp. [cit. 2025-04-21].

[33] W3SCHOOLS. *C# Tutorial*. Online. W3Schools. C1999-2025. Dostupné z: https://www.w3schools.com/cs/index.php. [cit. 2025-04-21].

[34] AMAZON WEB SERVICES. *What Is .NET?* Online. Amazon Web Services. C2025. Dostupné z: https://aws.amazon.com/what-is/net/. [cit. 2025-04-21].

[35] CODECADEMY. *What is .NET?* Online. Codecademy. 2020, Nov 7, 2022. Dostupné z: https://www.codecademy.com/article/what-is-net. [cit. 2025-04-21].

[36] HARTINGER, David. *Lekce 1 - Úvod do Windows Forms aplikací*. Online. ITnetwork. C2025. Dostupné z: https://www.itnetwork.cz/csharp/winforms/c-sharp-tutorial-windows-forms-okenni-aplikace-uvod. [cit. 2025-04-21].

[37] MICROSOFT. *Desktop Guide (Windows Forms .NET)*. Online. MICROSOFT. Microsoft Learn. 04/02/2025. Dostupné z: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-9.0. [cit. 2025-04-21].

[38] DIVIŠ, Roman a KAVIČKA, Antonín. The method of nested simulations supporting decision-making process within a mesoscopic railway simulator. Online. In: JIMÉNEZ, Emilio; LONGO, Francesco; LOUCA, Loucas S. a ZHANG, Lin, BRUZZONE, Agostino G. (ed.). *THE 28TH EUROPEAN MODELING & SIMULATION SYMPOSIUM*. Janov: University of Genoa, September 2016, s. 100-106. ISBN 978-88-97999-76-8. Dostupné z: https://www.msc-les.org/proceedings/emss/2016/EMSS2016_100.pdf. [cit. 2025-04-21].

[39] DIVIŠ, Roman. *Mesoskopický simulátor železniční dopravy MesoRail se základní podporou řízení provozu s podporou rozhodování pomocí vnořených simulací*. Online. Starfos. 2016. Dostupné z: https://starfos.tacr.cz/cs/vysledky-vyzkumu/RIV%2F00216275%3A25530%2F16%3A39901948. [cit. 2025-04-21].

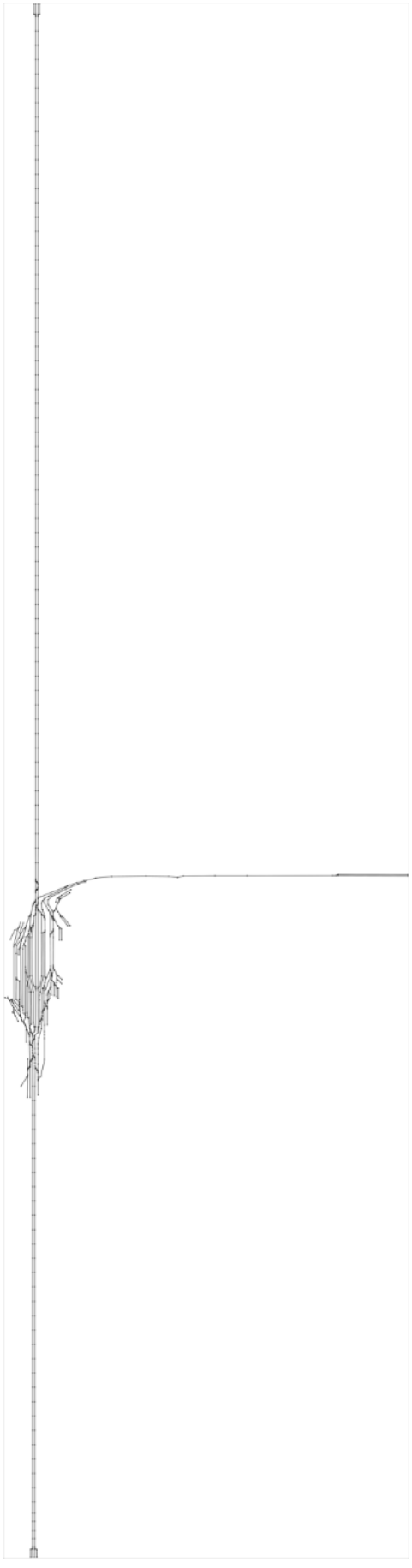
[40] NUGET. *SkiaSharp*. Online. MICROSOFT. NuGet. C2025. Dostupné z: https://www.nuget.org/packages/SkiaSharp/. [cit. 2025-04-21].

PŘÍLOHY

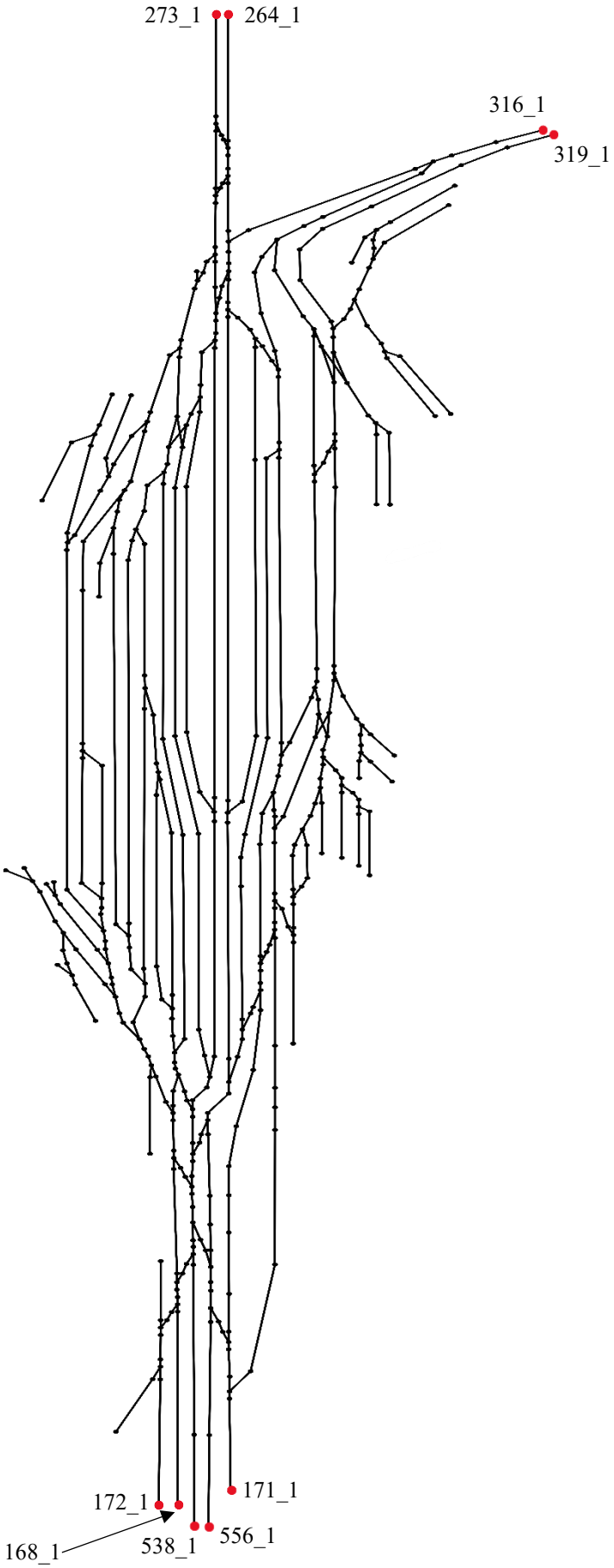
PŘÍLOHA A – MODEL DEMONSTRAČNÍ INFRASTRUKTURY

Tato příloha obsahuje modely železniční infrastruktury hlavního nádraží Pardubice.

Seznam obrázků:



Obrázek A1: Originální model infrastruktury [Zdroj vlastní]

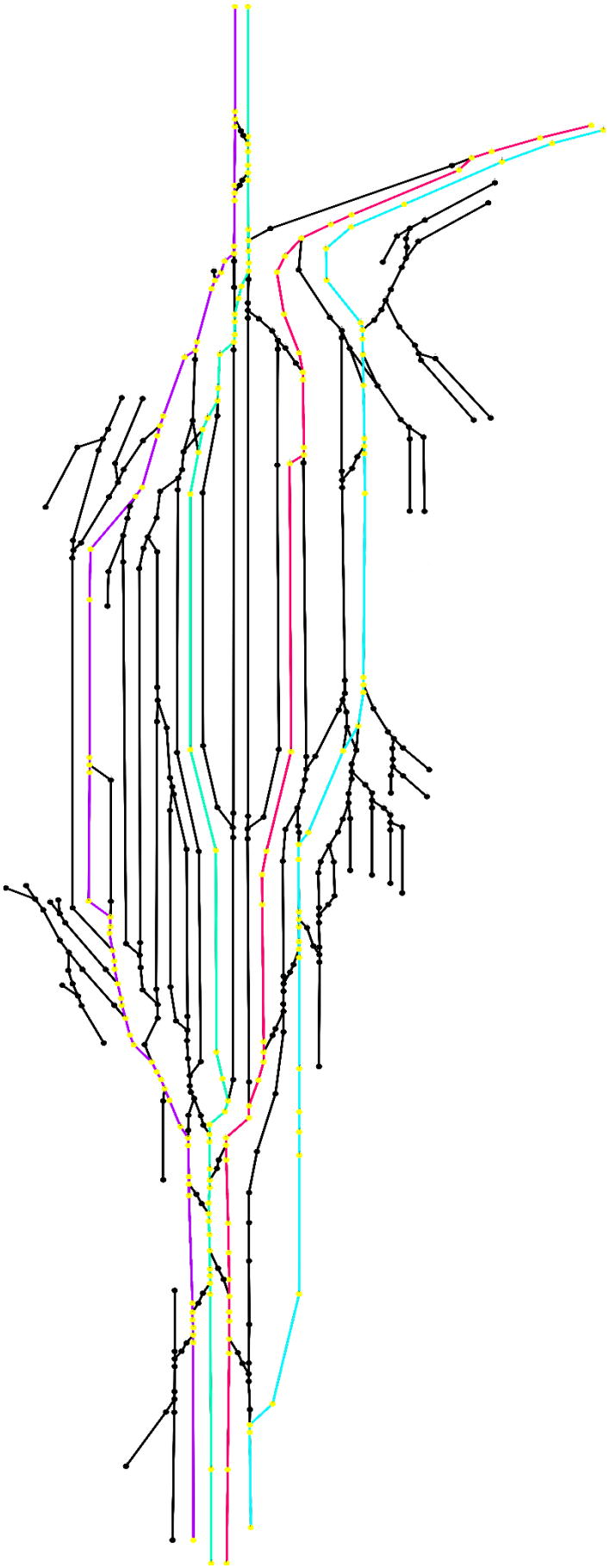


Obrázek A2: Upravený model infrastruktury s hraničními vrcholy [Zdroj vlastní]

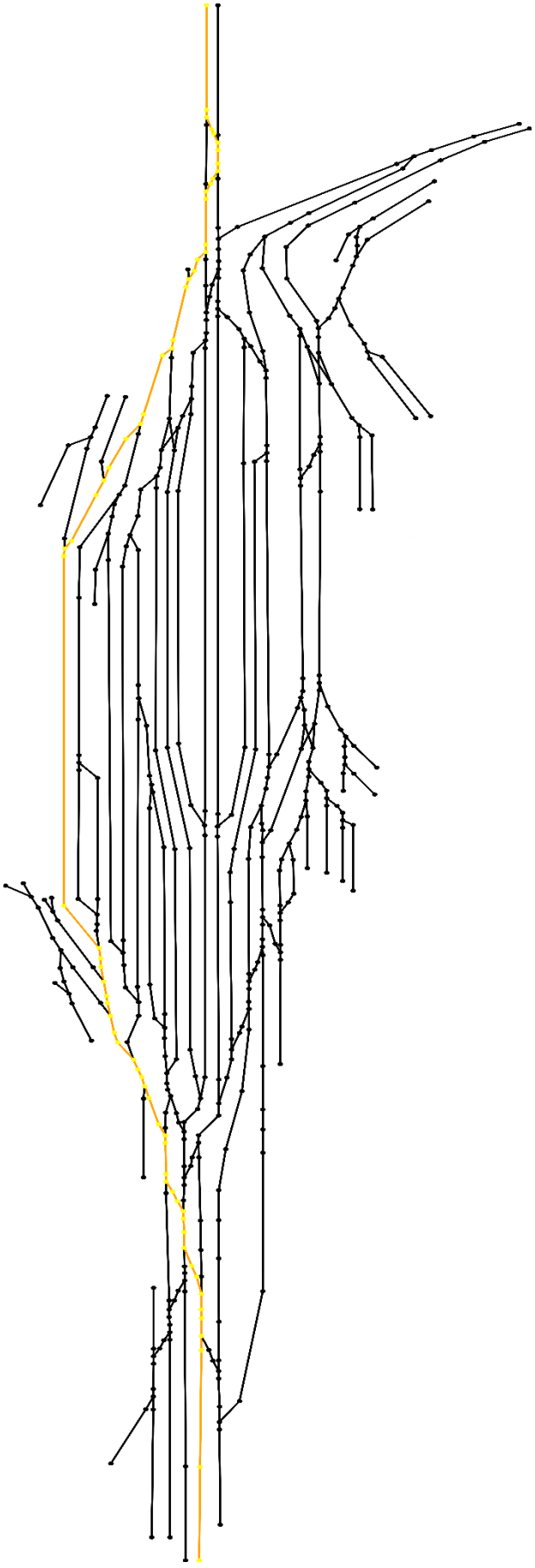
PŘÍLOHA B – VIZUALIZACE MODELU SE ZVÝRAZNĚNÝMI CESTAMI

Tato příloha obsahuje vizualizace modelu se zvýrazněnými cestami.

Seznam obrázků:



Obrázek B1: Model se zvýrazněnou čtveřicí souběžných cest [Zdroj vlastní]

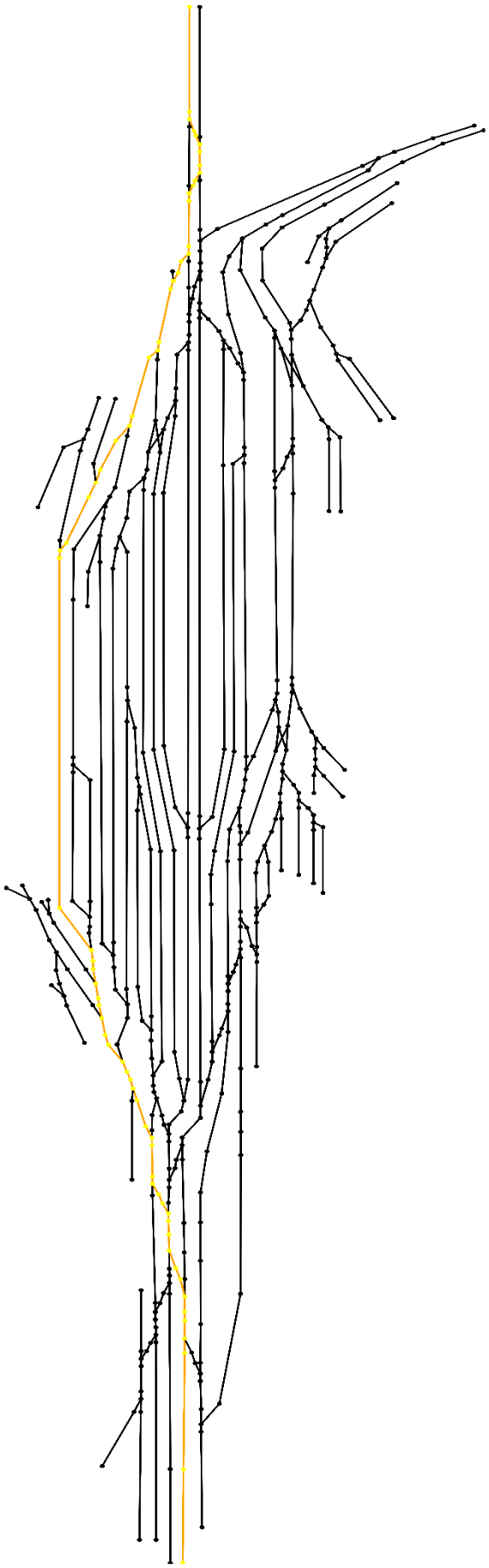


Obrázek B2: Model se zvýrazněnou nejdelší cestou [Zdroj vlastní]

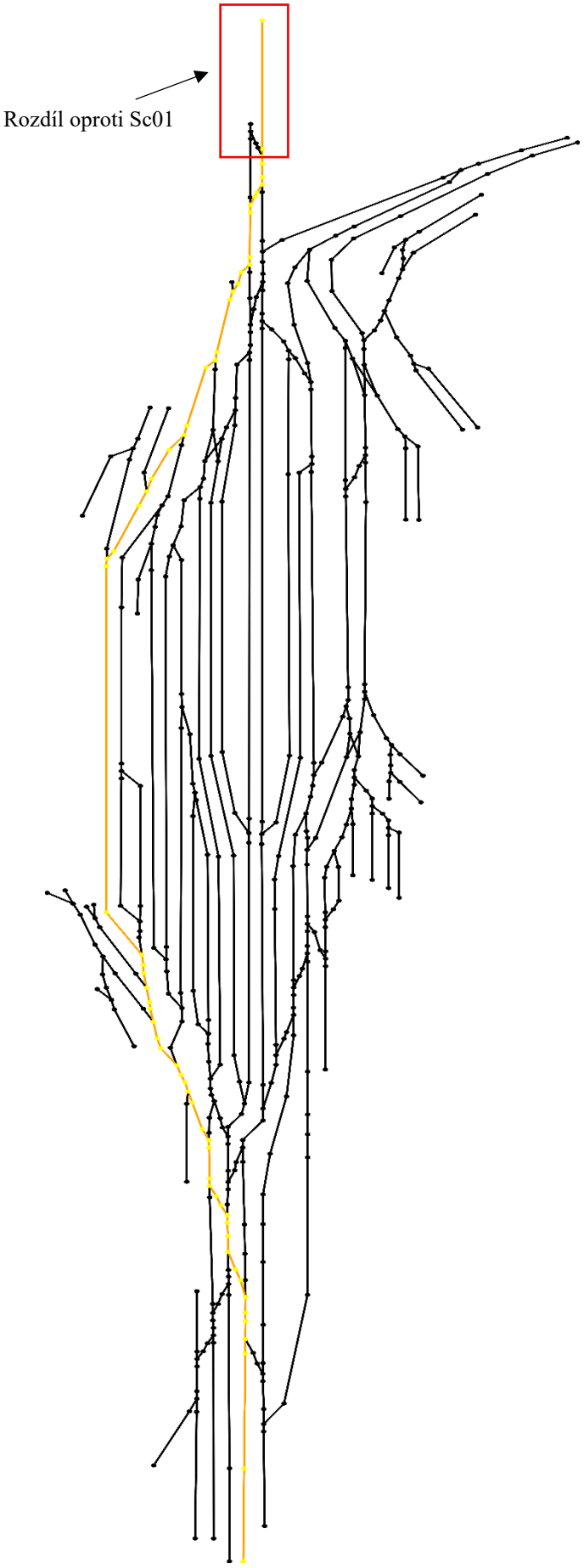
PŘÍLOHA C – VIZUALIZACE NEJDELŠÍCH CEST PRO TESTOVACÍ SCÉNÁŘE

Tato příloha obsahuje vizualizace modelu se zvýrazněnou nejdelší cestou pro jednotlivé scénáře.

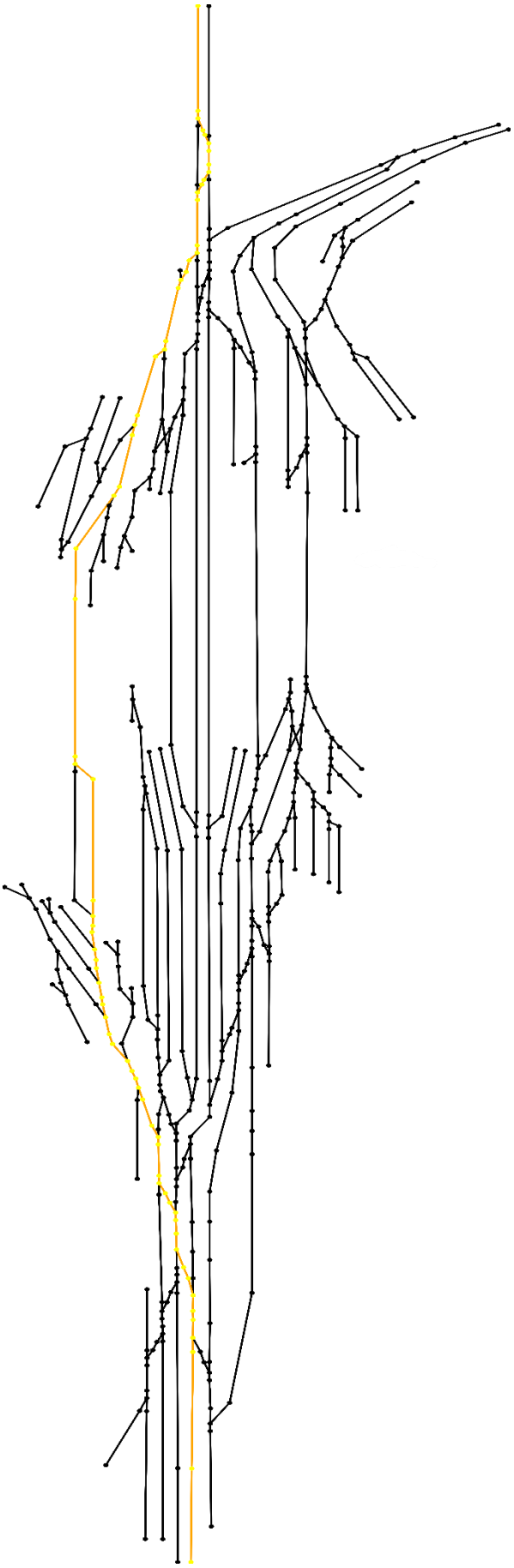
Seznam obrázků:



Obrázek C1: Nejdelší cesta v modelu pro scénáře Sc01 a Sc03 [Zdroj vlastní]



Obrázek C2: Nejdelší cesta v modelu pro scénář Sc02 [Zdroj vlastní]



Obrázek C3: Nejdelší cesta v modelu pro scénář Sc04 [Zdroj vlastní]

PŘÍLOHA D – UŽIVATELSKÁ PŘÍRUČKA

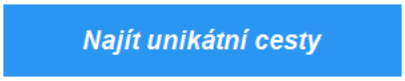
Tato příloha obsahuje stručnou uživatelskou příručku, jak aplikaci ovládat.

Spuštění aplikace – Spuštění aplikace se provede pomocí .exe souboru.

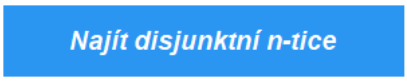
Seznam operací:

- Slouží k načtení modelu infrastruktury. Pomocí průzkumníka souborů vyberete patřičný .xml soubor s modelem infrastruktury. Po spuštění aplikace je tuto operaci potřeba provést jako první.

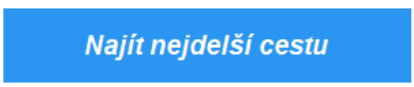
- Slouží k načtení hraničních vrcholů modelu. Pomocí průzkumníka souborů vyberete patřičný .csv soubor ID hraničních vrcholů. Tuto operaci je nutné provést po načtení modelu infrastruktury.



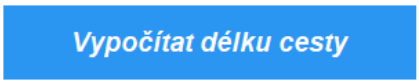
- Slouží k nalezení všech unikátních cest z hraničních vrcholů. Pro hledání unikátních cest je potřeba mít načtený model infrastruktury a hraniční vrcholy.



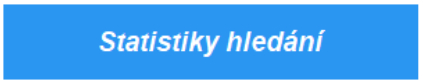
- Slouží k nalezení všech disjunktních n-tic složených z unikátních cest. Pro vyhledání disjunktních n-tic je nutné mít nalezené unikátních cesty. Jedná se časově náročnou operaci, takže je zobrazeno okno s informací, že se operace provádí.



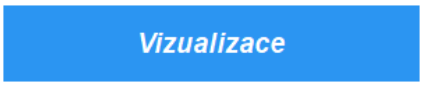
- Slouží k nalezení unikátní cesty s nejdelší délkou. Pro hledání nejdelší cesty je potřeba mít nalezené unikátní cesty.



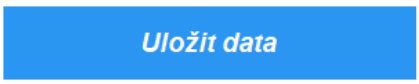
- Slouží k výpočtu délky unikátní cesty. Do zobrazeného formuláře zadejte ID cesty, u níž chcete délku vypočítat. Pro výpočet je nutné mít nalezené unikátní cesty.



- Slouží k zobrazení statistiky ohledně unikátních cest a disjunktních n-tic. Zobrazuje se počet unikátních cest, počet disjunktních n-tic a rozdělení podle typu (dvojice, trojice atd.).



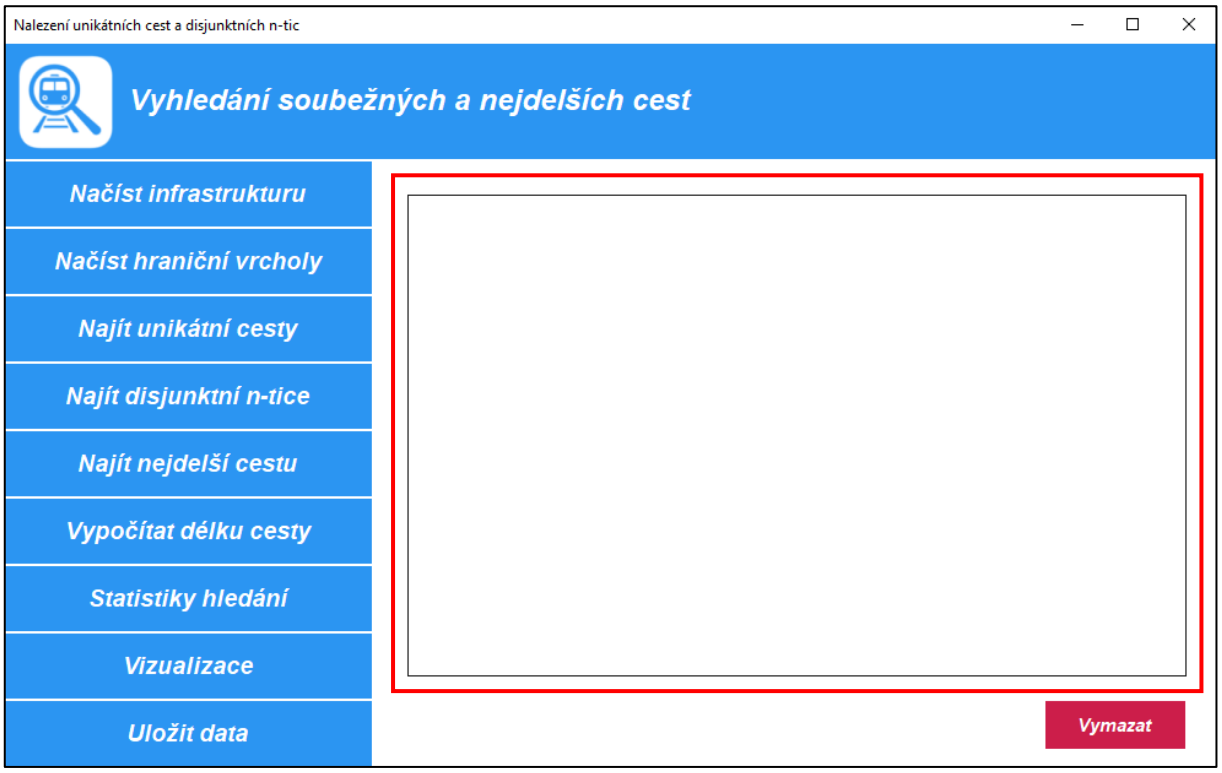
- Slouží k vizualizaci unikátních cest a disjunktních n-tic. V zobrazeném formuláři zadejte ID unikátní cesty, ID disjunktní n-tice nebo obojí, a poté pomocí průzkumníka souborů vyberte složku, do které se vizualizace uloží.



- Slouží k uložení nalezených unikátních cest, disjunktních n-tic a nejdelší cesty. Pomocí průzkumníka souborů vyberete složku, do které se data uloží.

- Vymaže všechny zprávy z výstupního/informačního panelu.

Informační panel:



Do označeného informačního panelu se vypisují všechny výsledky operací, především informace o tom, zda operace proběhla úspěšně. V případě chyby je zde uveden popis problému. Dále se v panelu zobrazují statistiky hledání cest a n-tic, vypočtená délka cesty a také informace o nalezené nejdelší cestě.