FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Porovnání algoritmů pro dělení silniční sítě

Lucie Roy

PLZEŇ

2024

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Porovnání algoritmů pro dělení silniční sítě

Lucie Roy

Vedoucí práce
Ing. Tomáš Potužák, Ph.D.

PLZEŇ 2024

Citace v seznamu literatury:
ROY, Lucie. *Porovnání algoritmů pro dělení silniční sítě*. Plzeň, 2024. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Ing. Tomáš Potužák, Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Lucie ROY**
Osobní číslo:	**A20B0224P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Porovnání algoritmů pro dělení silniční sítě**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s problémy dělení grafu a dělení silniční sítě.
2. Seznamte se s existujícími algoritmy pro dělení silniční sítě.
3. Vyberte alespoň tři různé algoritmy především na základě kvality jejich popisu umožňující implementaci.
4. Implementujte aplikaci pro snadné testování algoritmů v jednotném prostředí a s možností přidání dalších algorithmů. Vybrané algoritmy implementujte v rámci této aplikace.
5. Funkčnost a vlastnosti algoritmů prověřte na různorodých silničních sítích a porovnejte jejich vlastnosti.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Tomáš Potužák, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **2. října 2023**
Termín odevzdání bakalářské práce: **2. května 2024**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 25. října 2023

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Plzni dne 02. května 2024

. . . . . .
Lucie Roy

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Abstrakt

Simulace dopravy je jedním z nástrojů pro analýzu chování a řízení dopravy. Lze třeba zjišťovat, jak se změní doprava při uzavírce pruhu, změně světelných plánů semaforů apod. Podrobná simulace však může být výpočetně náročná, aby běžela na jednom počítači dost rychle. Jednou z cest, jak simulaci urychlit, je přizpůsobit ji pro simulaci distribuovanou, kdy se využije více propojených počítačů. Každý pak může paralelně simulovat jen část sítě. Pro tento účel je však potřeba silniční síť rozdělit na požadovaný počet částí. Protože kvalita dělení může výrazně ovlivnit výslednou rychlost distribuované simulace, bylo vytvořeno několik algoritmů založených na různých principech a optimalizujících různé vlastnosti výsledného dělení. Cílem této práce je vytvořit nástroj pro porovnání různých algoritmů dělení silniční sítě v jednotném prostředí. Kromě zmíněného nástroje byly implementovány tři různé algoritmy a porovnány dělením několika silničních sítí.

Abstract

Traffic simulation is one of the tools for behavior analysis and traffic management. It allows, among other things, to find out how traffic will change when a lane is closed, traffic light plans are changed, and so on. However, a detailed simulation can be computationally demanding to run fast enough on a single computer. One way to speed up the simulation is to adapt it for a distributed simulation environment where multiple interconnected computers are used. Each computer can then parallely simulate only part of the network. But for this purpose, the road network needs to be divided into the required number of parts. Since the quality of the division can significantly affect the resulting speed of the distributed simulation, several algorithms have been created based on different principles and optimizing different properties of the resulting division. The aim of this work is to create a tool for comparing different road network division algorithms in a unified environment. In addition to the mentioned tool, three different algorithms were implemented and compared by dividing several road networks.

Klíčová slova

dělení silniční sítě • distribuce • METIS • SParTSim • Inertial Flow

Poděkování

Na tomto místě bych ráda poděkovala svému vedoucímu bakalářské práce panu Ing. Tomáši Potužákovi, Ph.D. za jeho velkou trpělivost, vstřícnost, čas, podporu a cenné rady.

Lucie Roy,
autorka bakalářské práce
(květen 2024)

Obsah

Úvod 1

V dnešní době, kdy je ve velkoměstech velká hustota obyvatel a tím pádem i hodně dopravních prostředků, není jednoduché zachovat plynulost dopravy. Jedním z nástrojů, který tomu může pomoci je simulace dopravy umožňující mj. předvídat, jak bude doprava vypadat při uzavírce pruhu nebo změně světelných plánů křižovatek. Simulace velké sítě (např. celého velkoměsta) může být ale pro jeden počítač příliš náročná, pokud má být dostatečně rychlá. Jednou z používaných možností je využití distribuované simulace, kdy se využije více propojených počítačů a každý počítač simuluje pouze část silniční sítě. Proto je nutné silniční síť před samotnou simulací rozdělit. Jelikož kvalita dělení může hodně ovlivnit rychlost simulace, bylo již vytvořeno několik algoritmů řešících tento problém.

Kvalita algoritmů je často testována na různých sítích, jsou implementovány v různých jazycích, a proto je vzájemné porovnání na základě testů popsaných ve vědeckých publikacích obtížné. Cílem této práce je vytvořit nástroj umožňující porovnání různorodých algoritmů v jednotném prostředí. Díky tomuto nástroji budou algoritmy implementovány v jednom jazyce a testovány stejnou sadou testů. Kromě samotného nástroje budou implementovány a porovnány tři vybrané algoritmy.

Obsah této práce je následující. V další kapitole bude uvedena definice grafu a silniční sítě a popsáno jejich dělení. V Kap. 3 budou uvedeny některé existující algoritmy pro dělení silniční sítě a v Kap. 4 budou 3 vybrané algoritmy popsány detailněji. Následuje Kap. 5, kde je provedena analýza vytvářené aplikace, a Kap. 6, kde je popsána její implementace společně s implementací vybraných algoritmů. Dále je Kap. 7 Testování a porovnání vybraných algoritmů, kde je popis testování vybraných algoritmů ve vytvořené aplikaci a jejich porovnání podle výsledných hodnot, a závěr.

Graf a silniční síť 2

V této kapitole se blíže podíváme na graf a silniční síť. Nejdříve je uvedena definice grafu, po které následuje popis problému dělení grafu. Dále je představena silniční síť a její převod na graf. Poté jsou popsána specifika problému dělení silniční sítě.

2.1 Základní definice grafu

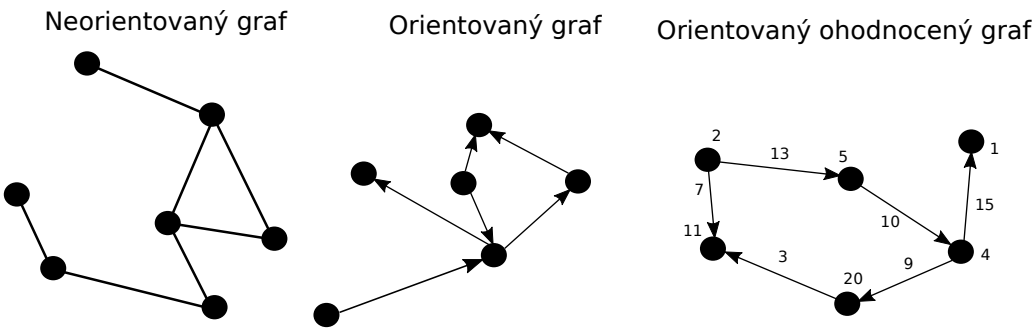
Minimálně v této kapitole se budeme zabývat grafem G jakožto uspořádanou dvojicí (V, E), kde V je neprázdná množina vrcholů a E je množina dvouprvkových podmnožin množiny V [1]. Dvouprvková podmnožina množiny V se nazývá hrana. Tato hrana spojuje dva vrcholy grafu. Hrany mohou být orientované (v takovém případě se značí křivkou zakončenou šipkou určují směr hrany), nebo neorientované (v takovém případě se značí křivkou bez šipky). Celý graf může být neohodnocený nebo ohodnocený. Ohodnocený graf může mít ohodnocené hrany a/nebo vrcholy. Hodnocení je provedeno přiřazením hodnoty ke každé hraně a/nebo vrcholu. Hodnota je často reprezentována reálným číslem, ale záleží na problému, který chceme grafem řešit. Příklady grafů jsou ukázány na Obr. 2.1. [1]

2.1.1 Hypergraf

Generalizací grafu definovaného výše je hypergraf. Rozdíl mezi grafem a hypergrafem je ten, že hrany hypergrafu mohou spojovat libovolný počet vrcholů [2]. Hypergrafem se ale budeme v této práci zabývat pouze okrajově.

2.2 Dělení grafu

Dělením grafu se v tomto textu myslí rozklad množiny V na požadovaný počet k disjunktních množin tak, aby se součet počtu vrcholů v jednotlivých množinách rovnal počtu vrcholů v původním grafu [3].



Obrázek 2.1: Příklady různých typů grafu.

2.2.1 Požadavky na dělení grafu

Tento problém se může zdát na první pohled jednoduchý, avšak toto dělení je často opatřeno ještě dalšími podmínkami. Máme ohodnocený graf, který může mít velké množství uzlů a/nebo hran (stovky, tisíce, desetitisíce). Tento graf je nutno rozdělit na k částí tak, aby jednotlivé části byly stejně veliké (tzv. vyvážený rozklad grafu [3]) a zároveň, aby tyto části byly spojovány co nejméně hranami. Stejná velikost v tomto případě znamená, že součty hodnocení v jednotlivých částech se alespoň přibližně rovnají. [3]

2.2.2 Metody dělení grafu

Obecně se výše popsaný problém, kde se graf dělí na k stejných částí, řadí do kategorie NP-úplných problémů. Tento problém se často objevuje v algoritmech rozděl a panuj, kdy se po rozdělení grafu pracuje s jeho jednotlivými částmi. Jelikož se tento problém řadí do NP-úplných problémů, je většinou řešen pomocí nějaké heuristiky. Tato heuristika často pracuje rekursivně a provádí půlení [4]. Příkladem takové heuristiky je rekurzivní bisekce, která dělení provádí rekurzivně postupným půlením částí grafu.

Rozklad grafu se dá také provést spektrálně. Tato metoda rozkladu využívá techniky lineární algebry. Spektrální rozklad je především vhodný k hrubému rozdělení grafu a následně se na tento rozklad aplikují lokální optimalizační metody [4].

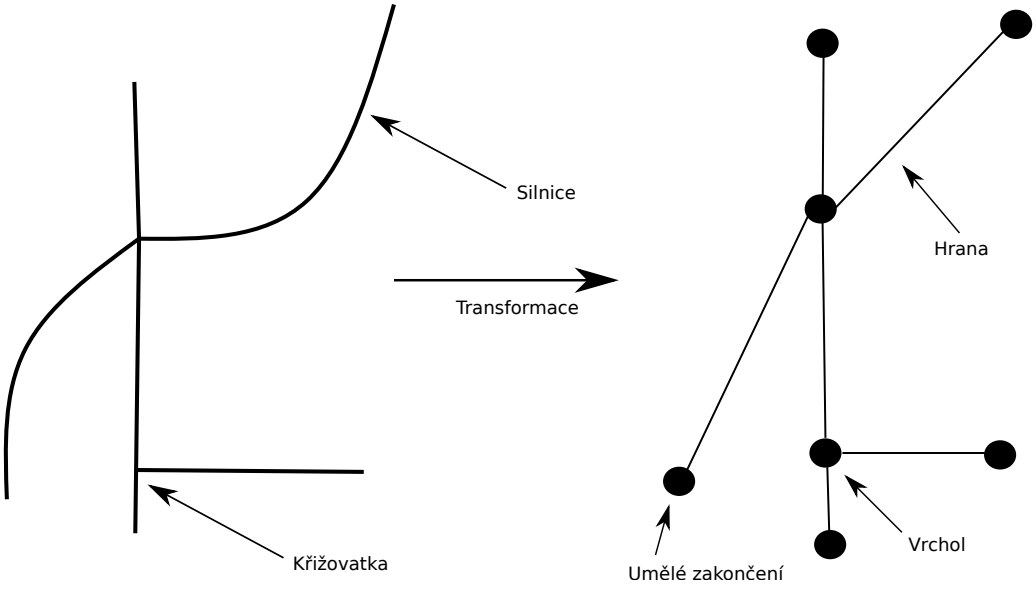
Další běžně používanou metodu je víceúrovňové schéma dělení grafu, které využívá zhrubování a zjemňování grafu [5]. Tento postup je podrobněji popsán v Kap.4.1.

2.3 Základní definice silniční sítě a její transformace na graf

Silniční síť se dá definovat jako systém silnic a křižovatek. Platí, že křižovatky jsou navzájem propojeny silnicemi. Transformace silniční sítě na graf je potřeba jako vstup do všech algoritmů uvedených v Kap.3.

2.3.1 Běžný způsob transformace

Nejběžnějším způsobem transformace je takový graf, kde křižovatky představují vrcholy a silnice představují hrany, které tyto vrcholy spojují (viz Obr. 2.2). Ohodnocení hran může mít různý význam. Jako hodnocení se často používá délka silnice, někdy také šířka a hustota dopravy, či průměrná rychlost na této silnici, nebo nějaká kombinace uvedeného. Pro některé algoritmy je vhodné ohodnotit i vrcholy grafu, hodnocení může být například na základě velikosti křižovatky, počtu silnic spojovaných křižovatkou, atd. Pokud má být výsledný graf neorientovaný, pak se během transformace silniční sítě neřeší směry jízdy a sousední křižovatky jsou propojeny jednou neorientovanou hranou. Graf vzniklý ze silniční sítě se vyznačuje tím, že téměř všechny jeho vrcholy mají poměrně nízký stupeň, který je často menší nebo roven 4. [6]



Obrázek 2.2: Transformace silniční sítě na graf, kde křižovatky jsou vrcholy a silnice jsou hrany.

2.3.2 Další způsoby transformace

Transformace silniční sítě popsaná v Kap.2.3.1 není jediná možná, je však nejvíce intuitivní. Pro simulaci dopravy však mohou být vhodnější jiné transformace, např. tzv. duální graf. V tomto grafu jeden vrchol představuje silnici v jednom směru jízdy a hrany reprezentují odbočovací směry mezi vstupními a výstupními silnicemi v křižovatkách. Tato přístup umožňuje snadno modelovat pouze povolené směry odbočení v křižovatce. Více informací je k dispozici v odborném článku [7].

2.3.3 Rozdělení na zóny

Zvláštním případem je transformace silniční sítě na zóny. Síť je rozdělena například na nákupní zónu, obytnou zónu, atp. Vrcholy grafu pak představují zóny a dopravní toky mezi zónami jsou reprezentovány hranami grafu. Tento graf, též matice počátků a cílů (origin-destination matrix - ODM) slouží pro znázornění toků mezi zónami a bývá použita jako jeden ze vstupních parametrů pro simulaci dopravy [8].

2.4 Dělení silniční sítě

Důvodů pro dělení dopravní sítě může být několik.

2.4.1 Důvody k dělení dopravní sítě

Jedním z hlavních důvodů pro dělení je distribuovaná simulace dopravy. Tato simulace většinou spočívá v tom, že máme velkou silniční síť, například celé velkoměsto, a chceme v ní simulovat různé dopravní situace, například co se stane, když někde přidáme nebo ubereme semafory, nebo je jen jinak nastavíme. Je zřejmé, že může být obtížné na jediném počítači detailně simulovat velkou síť v přijatelně krátkém čase. Protože výkon jednoho počítače by na simulaci nestačil, silniční síť se dělí mezi více počítačů a ty spolu pak vzájemně komunikují a předávají si data. Ovšem k dělení dopravní sítě mohou být i jiné důvody.

Dalším velmi častým důvodem je správa a řízení dopravy. Například můžeme síť rozdělit na několik částí, které jsou uvnitř co nejvíce podobné, a synchronizovat v nich semafory. Toto dělení můžeme také použít, pokud chceme odhadnout, jak dlouho bude trvat jízda auta z jedné části do jiné [9].

2.4.2 Homogenní a heterogenní dělení

Pokud má simulace probíhat co nejrychleji, nesmí na sebe jednotlivé simulační procesy dlouho čekat. Každému počítači by tam měla být přidělená část dopravní sítě úměrná jeho výkonu. Když mají všechny používané počítače téměř stejný výkon,

tak jednoduše rozdělíme síť na stejně velké části. Stejně velké v tomto případě znamená, že jsou stejně výpočetně náročné, což může záviset např. na množství vozidel pohybujících se v jednotlivých částech sítě. Tento typ dělení se nazývá homogenní. Ovšem jsou-li výkony počítačů různé, musíme síť rozdělit tak, aby nejvýkonnější počítač měl tu nejsložitější část a ten nejslabší počítač tu výpočetně nejjednodušší část. Tento typ dělení se nazývá heterogenní [10].

V obou případech je vhodné, aby části sítě měly co nejmenší počet hran, které je spojují (rozdělené hrany), a/nebo aby počet sousedních částí byl co nejmenší. Oba požadavky snižují nároky na meziprocesovou komunikaci [10].

2.4.3 Statické a dynamické dělení

Dále se můžeme setkat se statickým a dynamickým dělením. Statické dělení, také nazývané jako offline dělení, je dělení grafu silniční sítě ještě před začátkem simulace a toto rozdělení následně zůstává neměnné po celou dobu simulace. Dynamické dělení, také známé jako online dělení, se na rozdíl od statického mění během simulace. Algoritmus musí tedy během simulace vyhodnocovat, jestli by nebylo vhodné některé části grafu přeřadit k jinému počítači [10].

2.4.4 Statické dělení sítě pro distribuovanou simulaci dopravy

V této práci se budeme především zabývat dělením silniční sítě za účelem distribuované simulace dopravy s homogenním statickým dělením. Jak již je uvedeno výše, i pro tento typ simulace je výhodné, aby byly části rozdělené silniční sítě co nejpodobněji výpočetně náročné a aby byly spojovány co nejméně hranami a měly celkově minimální počet sousedů [10].

Jak již bylo řečeno výše, podobně výpočetně náročné neznamená, že musí být stejně velké, tedy aby pokrývaly stejnou plochu (například v km^2). Například, pokud porovnáme část silniční sítě někde na venkově a stejně velkou část silniční sítě pokrývající stejnou plochu ale ve středu velkého města, je zřejmé, že potřebné výkony pro tyto dvě části nebudou proporcionální jejich velikosti. Ve středu města bude určitě větší hustota aut a také propracovanější infrastruktura. Z toho vyplývá, že střed města bude určitě mnohem náročnější na simulaci než venkov. Tento problém by mohlo vyřešit správné ohodnocení hran a vrcholů, bohužel často pouze víme, kolik aut do sítě vjíždí a vyjíždí a už nevíme, jak se v té síti budou pohybovat, právě simulací se to snažíme zjistit. Jsme tedy většinou odkázáni na nějaké alternativní metriky, které jsou snadněji zjistitelné [11].

Vhodným příkladem alternativní metriky může být hustota obyvatel v dané oblasti nebo počet jízdních pruhů silnice. Také zde mohou hrát roli různé typy krajin, které mohou sloužit jako přirozený oddělovač. Například řeka může rozdělovat

město na dvě části. Tato metrika umožňuje především rozdělovat silniční síť tak, aby bylo přerušeno co nejméně silnic. Obecně je poměrně obtížné vybrat tu správnou metriku, jelikož, co fungovalo u jedněch dat, nemusí fungovat u jiných [12].

Existující algoritmy pro dělení silniční sítě — 3

Během této práce byly prohledány desítky odborných článků a vybrány ty nejvhodnější pro tuto kapitolu. V této kapitole budou uvedeny odborné články, které jsou především věnovány statickému homogennímu dělení. Kromě klasických modifikací algoritmů pro dělení obecného grafu jsou popsány především algoritmy určené přímo pro dělení silniční sítě.

3.1 Jednoduché algoritmy dělení

3.1.1 BFS

Mezi implementačně nejjednodušší algoritmy patří upravený BFS (Breadth-First Search - prohledávání do šířky). Tento algoritmus postupuje tak, že přidává do části grafu vrcholy, dokud nedosáhne kumulativního součtu ohodnocení hran (a/nebo vrcholů) odpovídající počtu částí. Například, pokud je graf dělen na dvě části, přidávání vrcholů do jedné části probíhá, dokud není kumulativní součet ohodnocení roven polovině celkového ohodnocení grafu [10].

3.1.2 Geografické dělení

Dalším primitivním algoritmem, nebo spíše metodou, je geografické rozdělení silniční sítě na stejně velké obdélníky. Tato metoda je velice jednoduchá a ze své podstaty nevyžaduje ani transformaci silniční sítě na graf. Takové dělení však může v mnoha případech vytvořit velmi výpočetně nevyvážené části silniční sítě [10].

3.2 Pokročilé metody dělení grafu

Dále se můžeme také setkat s pokročilejšími a zároveň složitějšími algoritmy pro dělení grafu.

3.2.1 METIS

Jedním z klasických algoritmů používaných i pro dělení obyčejného grafu je algoritmus zvaný METIS. Tento algoritmus využívá víceúrovňové schéma. Graf se nejprve zjednodušší pomocí seskupování vrcholů (tzv. zhrubování grafu). Tento zjednodušený graf se potom rozdělí na požadovaný počet částí a nakonec se zjednodušený graf opět upraví (zjemní) na původní graf. Zjemnění grafu je typicky doprovázeno využitím lokální optimalizace pro zlepšení kvality dělení (např. snížení počtu rozdělených hran, lepší vyvážení částí sítě atd.) [13], [5]. Podrobněji viz Kap. 4.1.

3.2.2 hMETIS

Podobným algoritmem je hMETIS, který na rozdíl od METISu dělí hypergraf, jinak ale pracuje velice podobně. Hypergraf nejdříve zhrubne, poté ho rozdělí a nakonec ho zjemní s využitím lokální optimalizace. Výsledné dělené má většinou menší hodnotu řezu (tj. menší počet rozdělených hran) než dělení pomocí METIS. [2][13]

3.2.3 Obecné víceúrovňové schéma

METIS není jediný algoritmus využívající víceúrovňové schéma. Podobný postup se v literatuře objevuje poměrně často [14], [15]. Jednotlivé postupy se typicky liší algoritmy, které jsou použity v jednotlivých fázích dělení - tedy při zhrubování grafu , dělení grafu, zjemňování grafu a optimalizaci řezu. Např. v [16] je pro dělení zhrubnutého grafu využit genetický algoritmus. Tento algoritmus nejdříve provede zhrubnutí vstupního grafu. V dalším kroku se pak tento zhrubnutý graf použije jako vstup pro dělící genetický algoritmus, jehož jedinec představuje přiřazení každého vrcholu do jedné z částí, tedy jednu možnost rozdělení grafu. Poté je vybrán nejlépe hodnocený jedinec, podle kterého je zhrubnutý graf rozdělený. Nakonec se provede zjemnění grafu a jeho rozdělení podle grafu zhrubnutého [16].

3.3 Metody pro dělení silniční sítě

Metody pro dělení silniční sítě jsou často upravenou metodou pro dělení běžného grafu s využitým dodatečných informací, které jsou pro silniční síť k dispozici, případně s využitím jejích specifických vlastností.

3.3.1 PUNCH

Příkladem algoritmu, který byl vyvinut především pro silniční síť, je PUNCH (Partitioning Using Natural Cut Heuristics). Tento algoritmus je vhodný především pro větší územní celky, například pro celý stát. Jako vstupní parametr pro tento algoritmus se udává maximální velikost jedné části U. Výstupní rozdělení tedy bude mít

části velké maximálně U. Dalo by se říci, že PUNCH upřednostňuje minimální počet hran spojujících jednotlivé části před vybalancovaným rozdělením (přibližně stejný potřebný výkon na jednotlivé části). Algoritmus se dá i využít pro vyvážené dělení, ovšem počet hran spojujících jednotlivé části už nemusí být minimální. PUNCH se skládá ze dvou částí nebo tří částí, pokud počítáme i transformaci silniční sítě na graf. První část se nazývá fáze filtrování, během níž se graf zjednoduší a určí se v něm přirozené řezy (například řeka, hory, atd.). V druhé části nazvané fáze sestavení se vzniklý graf z předešlé fáze rozdělí na požadovaný počet částí pomocí žravého algoritmu, který je doplněn optimalizačními algoritmy [12].

3.3.2 Inertial Flow

Algoritmus s názvem Inertial Flow (Inerciální tok/proudění) byl také vyvinut speciálně pro silniční síť. Tento algoritmus používá dva základní principy. Nejdříve vrcholy ortogonálně promítne na přímku zadanou jako parametr a následně spočítá maximální tok grafem, kde počáteční vrchol se vytvoří z prvních x vrcholů na přímce a cílový vrchol je z posledních x vrcholů na přímce. Podle toho se pak určuje, kde má být řez [17].

3.4 Metody využívající dodatečná data

Dalším příkladem jsou algoritmy, které používají nebo mohou používat dodatečné informace o silniční síti pro vylepšení dělení.

3.4.1 SParTSim

Jedním takovým algoritmem je například SParTSim. SParTSim používá nejen přirozené řezy, ale také informace o různých třídách silnic. Algoritmus spočívá především v tom, že se na začátku určí takový počet počátečních vrcholů, aby se rovnal počtu požadovaných částí, a k těmto bodům se pak postupně přidávají další okolní vrcholy. Tento růst oblastí se děje pro všechny počáteční body najednou, mezi body tak vznikne soutěž, kdo bude větší. Tento růst končí, když už se žádná z oblastí nemůže dál rozrůstat [11].

3.4.2 Dělení používající korelace cestovní rychlosti

Druhý algoritmus, který používá nějaká data navíc kromě těch obvyklých, je popsán v článku [18]. V tomto článku jsou použity rychlosti pro danou oblast silniční sítě. Tato data jsou získána z GPS taxi služeb. Následně jsou tato data použita pro přerozdělení grafu silniční sítě podle rychlostí. Tato metoda se hlavně zaměřuje na obydlené oblasti.

3.4.3 Shlukovací algoritmus

Třetí příklad algoritmu využívající dodatečná data, je algoritmus kombinující „canopy“ a k-means shlukování. Jako doplňující informace používá průměrné rychlosti v úsecích a hustotu dopravy. Nejdříve transformuje silniční síť pouze na body se čtyřmi souřadnicemi. Tyto souřadnice obsahují zeměpisnou délku, zeměpisnou šířku, průměrnou rychlost v dané sekci a hustotu dopravy. Body jsou poté shlukovány podle „canopy“ shlukovacího algoritmu. Tento algoritmus určí, na kolik částí bude silniční síť dělena a toto číslo se pak použije jako vstup pro k-means shlukování. Výsledné dělení se příliš nehodí na distribuovanou simulaci dopravy, jelikož vytváří spíše části, které jsou uvnitř homogenní, ale jednotlivé části mohou být mezi sebou značně heterogenní. Jedná se tedy o příklad algoritmu, který není příliš vhodný pro distribuovanou simulaci dopravy, ale spíše pro optimalizaci a řízení dopravy dané silniční sítě [19].

Vybrané algoritmy 4

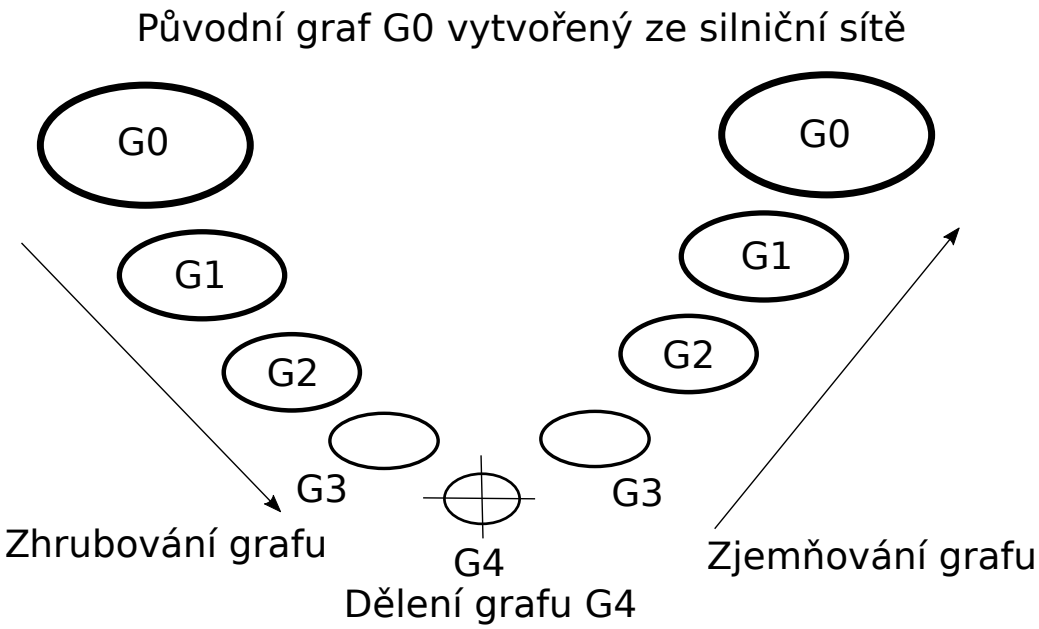
V této práci byly implementovány tři vybrané algoritmy, a to METIS, SParTSim a Inertial Flow. Algoritmy byly vybrány na základě kvality jejich popisu, který musel být dostatečný pro následnou implementaci pro účely této práce, požadovaných dat pro běh algoritmu a vhodnosti pro homogenní statické dělení silniční sítě. Zároveň byly vybrány algoritmy, které si nejsou navzájem moc podobné.

4.1 Algoritmus METIS

Algoritmus METIS je zevrubně popsán v [5]. Na základě tohoto popisu byla vytvořena implementace jedné z variant. V tomto článku jsou popsány jednotlivé varianty algoritmu a některé z nich jsou otestovány a porovnány. Zde bude popsána pouze varianta, která byla v článku nejlépe ohodnocena, jelikož právě ta byla v rámci této práce naimplementována. Vstupem tohoto algoritmu je graf a počet požadovaných částí. Jak již bylo zmíněno v Kap. 3 tento algoritmus se skládá ze tří hlavních částí: zhrubování grafu, dělení grafu a zjemňování grafu (viz Obr. 4.1). K tomuto algoritmu je sice k dispozici implementace zmíněná v článku a dostupná na GitHub.com [20], ale tato implementace je poměrně nepřehledná a pro účel této práce zbytečně komplikovaná, proto bude algoritmus pro tuto práci implementován především dle [5].

4.1.1 Zhrubování grafu

Pro zhrubování grafu byla vybrána metoda „Heavy Edge Matching“ v překladu párování těžkých hran. Ve zmíněném článku byla tato metoda ohodnocena jako nejrychlejší s velice dobrými výsledky. Zhrubování grafu se provádí párováním vrcholů, které mají mezi sebou hrany s největším ohodnocením. Vrcholy se prochází v náhodném pořadí a pokud ještě nebyl v této iteraci spárován, tak se vždy spáruje/sjednotí s tím sousedem, který s vrcholem sdílí hrany s největším ohodnocením a ještě také nebyl spárován v této iteraci. Iterací se provádí vždy tolik, aby zhrubnutý graf měl pouze 100 vrcholů. Detailněji viz Alg. 4.1.



Obrázek 4.1: Základní proces algoritmu METIS.

4.1.2 Dělení grafu

Dále probíhá dělení zhrubnutého grafu. Opět byla vybrána metoda, která měla podle odborného článku nejlepší výsledky, a to „Greedy Graph Growing Algorithm“, což se dá přeložit do češtiny jako žravý algoritmus pro růst grafu. Na začátku dělení se vybere náhodný počáteční vrchol a od něj se pak graf prochází do šířky, ovšem přednost se dává vrcholům, které po jejich přidání přispějí nejméně hranami, jejichž druhý konec by patřil do jiné části. Pokud je hodnota prohledané části alespoň přibližně rovna ideální hodnotě jedné části grafu, pak prohledávání končí a začne se vytvářet další část.

4.1.3 Zjemňování grafu

Poslední fází je zjemnění grafu společně s optimalizací řezu. Zjemnění grafu je docela přímočaré. Jednoduše se původní vrcholy přiřadí do té částí, ve které je hromadný vrchol vzniklý v rámci zhrubování grafu, do něhož původní vrchol patří. Následuje tzv. zpřesňovací algoritmus („refinement algorithm“), který již dobré dělení algoritmu ještě upraví tak, aby bylo dělení co nejlepší. Jako zpřesňovací algoritmus byl opět vybrán ten nejlépe hodnocený a to „Boundary Kernighan-Lin Refinement“, tedy hraniční Kernighan-Lin zpřesnění.

Tento algoritmus nejdříve spočítá pro všechny vrcholy, které mají alespoň jednu společnou hranu s vrcholem z jiné části, jejich rozdíl mezi váhou hran, které mají

Algoritmus 4.1 Zhrubování grafu

vstup: Silniční síť $G = (V, E, w)$.
výstup: Zhrubený graf $G_z = (V_z, E_z, w_z)$.

```
multiVertices ← empty_list
for all V do
    multiVertex ← v
    AddToMultiVertices(multiVertex)
end for

while |multiVertices| > 100 do
    sortedMultiVertices ← SortMultiVertices(multiVertices)
    matchedMultiVertices ← empty_list
    for all sortedMultiVertices do
        if sortedMultiVertex not in matchedMultiVertices then
            maxWeight ← −1
            maxVertex
            neighbours ← GetAdjacentVerticesAndValues(sortedMultiVertex)
            for all neighbours do
                if neighbour not in matchedMultiVertices then
                    if neighbourValue > maxWeight then
                        maxWeight ← neighbourValue
                        maxVertex ← neighbour
                    end if
                end if
            end for
            if maxVertex not empty then
                AddToMatchedMultiVertices(sortedMultiVertex)
                AddToMatchedMultiVertices(maxVertex)
                JoinMultiVertices(multiVertices,
                                  sortedMultiVertex, maxVertex, maxWeight)
            end if
        end if
    end for
end while
```

druhý konec v druhé části, a hran, jejíž druhý konec vede do stejné části jako právě zpracovávaný vrchol. Dále hledá nejvhodnější dvojici vrcholů, kde každý je z jiné části, na prohození podle vzorce 4.1:

$$g(v_1, v_2) = (ED[v_1] - ID[v_1]) + (ED[v_2] - ID[v_2]) - 2 * ED[v_1, v_2]\,, \qquad (4.1)$$

kde $g(v_1, v_2)$ je vhodnost prohození vrcholů v_1 a v_2, $ED[v_1]$ ($ED[v_2]$) je hodnota hran spojující vrchol v_1 (v_2) s druhou částí grafu, $ID[v_1]$ ($ID[v_2]$) je hodnota hran

spojující vrchol v_1 (v_2) s částí grafu, do které patří, a $ED[v_1, v_2]$ se rovná hodnotě hran spojujících v_1 a v_2. Pokud již nelze nalézt žádnou dvojici, která by snížila hodnotu řezu, dělení je hotové a algoritmus končí.

4.2 Algoritmus SParTSim

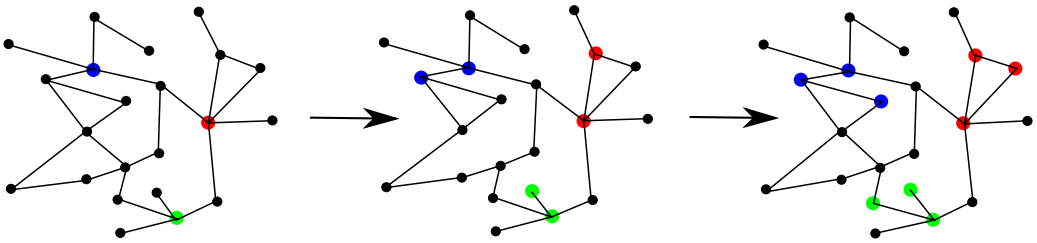
Druhým vybraným algoritmem je SParTSim. Algoritmus sice bere v potaz třídy silnic a přírodní dělící prvky (hory, řeky, atd.), ovšem je možné provést dělení i bez těchto doplňujících dat. Implementace algoritmu pro tuto práci bude napsána podle pseudokódu z [11]. Funkce uvedené v pseudokódu jsou v článku velice dobře popsány, proto by měla být implementace poměrně jednoduchá. Vstupem algoritmu je, kromě grafu a počtu částí, také tzv. přijatelné nevyvážení dělení, což je číslo, které udává maximální rozdíl mezi maximální a předpokládanou váhou a minimální a předpokládanou váhou. Z Alg. 4.2 je vidět, že algoritmus SParTSim se skládá ze čtyř hlavních částí: inicializace, růst oblastí, vyvážení dělení a zajištění spojitosti.

4.2.1 Inicializace

Inicializace spočívá ve vybrání nejlepších kandidátů pro počáteční vrcholy. Vhodnost kandidátů je hodnocena podle jejich stupně, tedy počtu hran s ním spojeným. Je tedy vybráno k vrcholů s nejvyšším stupněm, kde k je počet potřebných částí. [11]

4.2.2 Růst oblastí

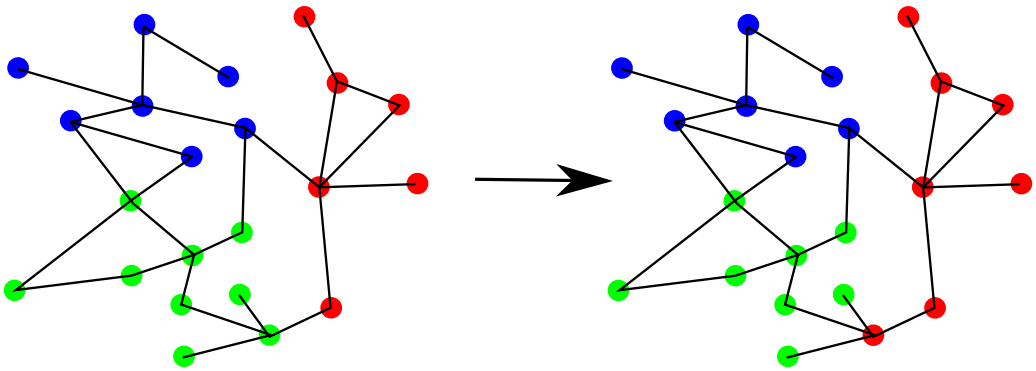
Následuje růst oblastí, který je graficky znázorněn na obrázku 4.2. Tato fáze může být implementována i paralelně a v článku tomu tak je, ovšem pro účely této práce je lepší sériová implementace z důvodu porovnatelnosti vybraných algoritmů. Jako nejlepší kandidát pro přidání do oblasti je vždy vybrán ten vrchol, který má s oblastí společné hrany s celkovou největší váhou [11].



Obrázek 4.2: Růst tří oblastí.

4.2.3 Vyvážení dělení

Po fázi růstu oblastí vznikne často nevyvážené řešení. Pokud není dodrženo přijatelné nevyvážení dělení, je nalezena část s maximální a část s minimální váhou. Mezi těmito částmi je nalezena nejkratší cesta a některé vrcholy na této cestě jsou následně přesunuty do části sousedící s těmito vrcholy (viz obrázek 4.3). Těmito posuny se postupně docílí vyvážení mezi maximální a minimální částí.



Obrázek 4.3: Vyvážení dělení.

4.2.4 Zajištění spojitosti

Vstupem této fáze je sice rozdělený graf, ovšem je možné, že v rámci vyvažování dělení se některé části „rozpadnou“, a tak vznikne více částí než bylo potřeba. V tomto posledním kroku je tudíž nutno spojit některé části tak, aby bylo splněné zadání; aby počet částí byl roven požadovanému počtu a zároveň, aby bylo dělení vyvážené. V článku není přesně uvedeno, jak tohoto dosáhnout. Proto byla v rámci této práce vytvořena vlastní metoda. Tato metoda spočívá v tom, že postupně snižuje počet přebývajících částí jejich spojováním. Spojování probíhá tak, že se nejdříve spojují části, jejichž součet hodnot se nejvíce blíží průměrné hodnotě jedné části grafu.

4.3 Algoritmus Inertial Flow

Posledním vybraným algoritmem je Inertial Flow (Inerciální tok) [17]. Tento algoritmus se vyznačuje svou originalitou a jednoduchostí. Vstupem tohoto algoritmu je nejen graf a počet požadovaných částí, ale také přímka l a vyváženost b, která je vyjádřena reálným číslem menším než 0,5. Algoritmus má celkem 5 hlavních kroků: ortogonální promítnutí všech vrcholů grafu na vstupní přímku, seřazení vrcholů podle pořadí na přímce, sjednocení $b * |V|$ prvních vrcholů do jednoho vrcholu s

a $b * |V|$ posledních vrcholů do jednoho vrcholu t, vypočítání maximálního toku mezi A a B a nalezení minimálního řezu k vypočtenému maximálnímu toku.

4.3.1 Ortogonální promítnutí vrcholů na přímku a jejich seřazení

První krok tohoto algoritmu je ortogonální promítnutí všech vrcholů vstupního grafu na vstupní přímku (viz Obr. 4.4). Souřadnice obrazu o jednotlivých vrcholů na přímce l se dají vypočítat takto:

$$x_o = \frac{(x_B - x_A) * p_1 - (y_A - y_B) * p_2}{(y_A - y_B)^2 - (x_A - x_B) * (x_B - x_A)} \tag{4.2}$$

$$y_o = \frac{(y_A - y_B) * p_1 - (x_A - x_B) * p_2}{(x_A - x_B) * (x_B - x_A) - (y_A - y_B)^2}, \tag{4.3}$$

kde

$$p_1 = -x_A^2 + x_A * x_B - y_A^3 + y_A^2 * y_B \tag{4.4}$$

$$p_2 = -(y_A - y_B) * x_v - (x_B - x_A) * y_v, \tag{4.5}$$

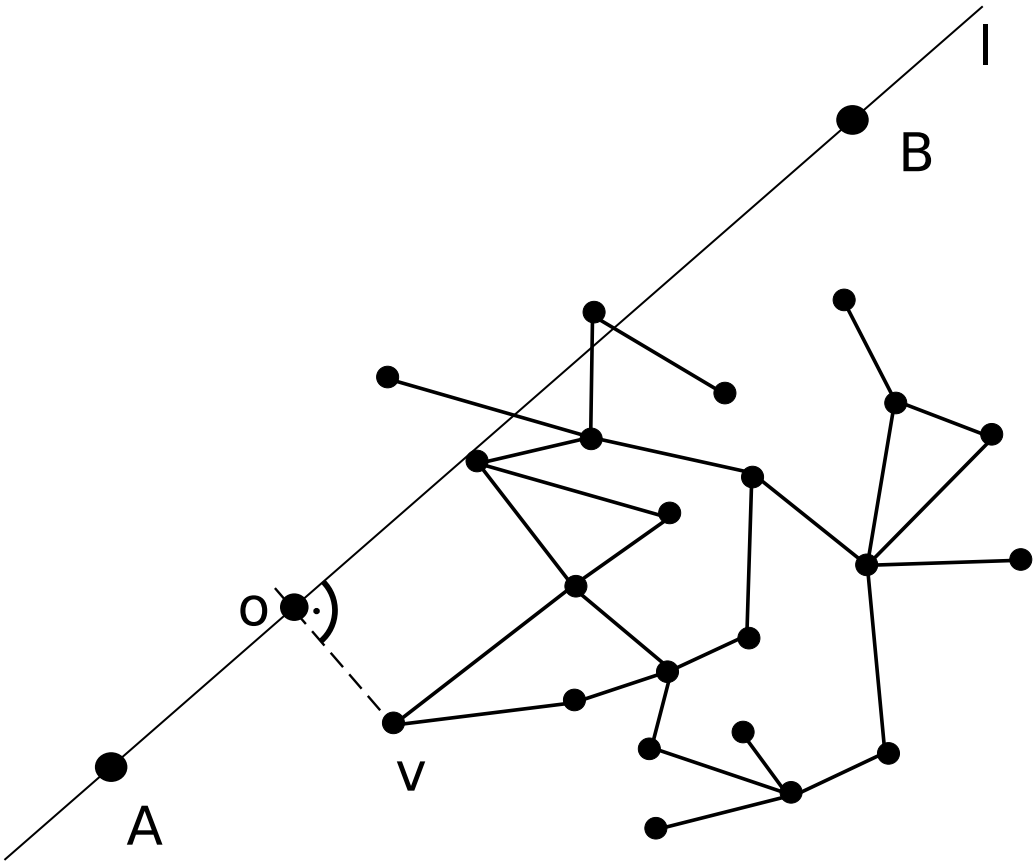
x_A a y_A jsou souřadnice bodu A ležícího na přímce l, x_B a y_B jsou souřadnice bodu B ležícího na přímce l a x_v a y_v jsou souřadnice promítaného vrcholu. Dalším krokem je seřazení vrcholů podle jejich pořadí obrazů na přímce. Pro toto řazení není v článku určen řadící algoritmus, pro tuto práci bude použito řazení vkládáním („insertion sort“). Toto řazení je velice dobře přizpůsobeno na vkládání nových prvků do pole, je tudíž možné promítat vrcholy po jednom a rovnou je dávat na správné místo v poli.

4.3.2 Sjednocení vrcholů a výpočet maximálního toku

Pro tento krok je vstupem seřazené pole (seznam) vrcholů z předchozího kroku. Prvních $b * |V|$ vrcholů z pole se sjednotí do jednoho nového vrcholu s a posledních $b * |V|$ vrcholů se sjednotí do jednoho nového vrcholu t. Dále se pomocí Dinicova algoritmu [21] vypočítá maximální tok mezi s a t. Výstup Dinicova algoritmu je pouze hodnota maximálního toku, která se rovná hodnotě minimálního řezu. Dinicův algoritmus hledá vždy nové cesty mezi s a t přes vrcholy, které ještě nebyly navštíveny, a hledá minimální ohodnocení hrany na této cestě. Toto ohodnocení se poté rovná maximálnímu toku v dané cestě.

4.3.3 Nalezení minimálního řezu

Metoda pro nalezení minimálního řezu podle maximálního toku není v článku příliš rozvedena. Spíše se očekává, že si zájemci o tento algoritmus vymyslí metodu vlastní.



Obrázek 4.4: Ortogonální promítnutí *o* vrcholu *v* na přímku *l*.

V této práci bude použita vlastní metoda pro nalezení minimálního řezu grafu podle hodnoty maximálního toku.

Tato metoda spočívá v uložení všech maximálních toků, které byly spočítány pro jednotlivé cesty během Dinicova algoritmu, a prohledáváním grafu do hloubky. Postupně se v grafu hledají hrany se stejným ohodnocením jako byl maximální tok jedné z cest a pokud jsou nalezeny všechny hrany s maximálním tokem a zásobník s dalšími možnými vrcholy podle prohledávání do hloubky je prázdný, toto dělení uložíme. Je-li hodnota prohledané části větší nebo rovna půlce hodnoty grafu, dělení končí, jinak prohledává dál. Může se stát, že po nalezení všech hran s maximálním tokem nebude zásobník prázdný, to znamená, že nalezený řez není minimální a musíme nalézt jiný. Metoda je podrobněji popsána v Alg. 4.3.

Algoritmus 4.2 Algoritmus SParTSim [11].

vstup: Silniční síť $G = (V, A, w)$; k počet požadovaných částí; ϵ přijatelná nevyváženost dělení.
výstup: $\pi(G, k)$ dělení silniční sítě.

```
stop ← new_array_with_size_k
for i ← 1, k do
    γ_i ← BestCandidateVertex()
    stop[i] ← 1
end for

while stop ≠ zeros do
    for i ← 1, k do
        if stop[i] ≠ 0 then
            hasGrown ← Grow(γ_i)
            if hasGrown then
                stop[i] ← 0
            end if
        end if
    end for
end while

balanced ← false
while not_balanced_or_not_enough_iterations do
    γ_i ← max(π(G, k))
    γ_j ← min(π(G, k))
    if |γ_i[V]| − ε < |G[V]|/n < |γ_j[V]| + ε then
        balanced ← true
    else
        Trade(γ_i, γ_j)
    end if
end while

for all γ do
    ComputeConnectedSubgraphs(γ_i)
end for

for all connectedComponents do
    Attach(connectedComponent_{γ_i}^j)
end for
```

Algoritmus 4.3 Nalezení minimálního řezu

vstup: Silniční síť $G = (V, A, w)$; `tempVertexList` - seznam se sjednocenými a nesjednocenými vrcholy; `flowList` - seznam hodnot hran v minimálním řezu (získané v rámci Dinicova algoritmu).
výstup: `oneHalfVertices` - vrcholy patřící do první části/poloviny.

```
oneHalfVertices ← empty_list
tempOneHalfVertices ← empty_list
stack ← empty_stack
minEdgeStack ← empty_stack
visitedVertices ← empty_list
value ← AddToOneHalfVertices(tempVertexList[first],
stack, oneHalfVertices, visitedVertices, 0)
tempValue ← 0
while stack not empty do
    firstVertexInStack ← PopStack(stack)
    for all GetStartingEdges(firstVertexInStack) do
        endPoint ← GetEndPoint(startingEdge)
        if endPoint not in visitedVertices then
            if EdgeNotInMinCut(startingEdge) then
                tempValue ← AddToOneHalfVertices(endPoint,
stack, tempOneHalfVertices, visitedVertices, tempValue)
            else
                PushStack(minEdgeStack, startingEdge)
                MarkEdgeValue(flowList)
            end if
        end if
    end for
    if stack is empty then
        if value + tempValue < |G|/2 then
            oneHalfVertices ← oneHalfVertices +
                                  tempOneHalfVertices
            value ← value + tempValue
            tempOneHalfVertices ← empty_list
            StartFromFirstMinCutEdge(...)
        end if
    else if AllEdgeValuesMarked(flowList) then
        UnmarkSomeValuesAndContinueSearching(...)
    end if
end while
```

5 Návrh aplikace pro testování funkčnosti a vlastností algoritmů pro dělení silniční sítě

V rámci této bakalářské práce bude vytvořena aplikace pro testování funkčnosti a vlastností algoritmů pro dělení silniční sítě.

5.1 Požadavky na aplikaci

Aplikace pro svou činnost nebude vyžadovat síťovou komunikaci. Bude primárně sloužit pro porovnání implementovaných případně dalších dodaných dělících algoritmů. Proto není nutné, aby se jednalo o webovou aplikaci, desktopová aplikace je dostačující.

Jelikož by bylo velice praktické, aby aplikace byla schopná uživateli výsledné dělení grafu ukázat, bude mít aplikace grafické uživatelské rozhraní. Nabízí se tedy použít pro implementaci aplikace tří-vrstvou architekturu. Aplikace tak bude rozdělena do tří částí: pohledy, kontrolery a výpočty/data.

Také je vhodné v aplikaci zautomatizovat testování dělících algoritmů. Například tím, že uživateli umožníme nastavit, kolik stejných testů chce provést, aby mohlo delší testování běžet i přes noc.

5.1.1 Programovací jazyk

Pro desktopové aplikace se často používá Java, Python nebo třeba C++. Ovšem jedním z požadavků na aplikaci je dát možnost uživateli přidat další dělící algoritmy do již hotového přeloženého programu. Jedním z běžných jazyků, kde se toho dá poměrně jednoduše dosáhnout (např. pomocí reflexe - viz Kap. 6.2) je Java. Navíc má tento jazyk k dispozici mj. platformu JavaFX pro tvorbu pěkného grafického

uživatelského rozhraní. V neposlední řadě je to jazyk, se kterým mám já i zadavatel práce nejvíce zkušeností.

5.1.2 Vstup a výstup aplikace

Účelem aplikace je dělení silniční sítě, přičemž vstup a výstup bude ze souboru a do souboru. Dlouhodobé uchování dat není potřeba, proto nebude využita žádná databáze.

Dále je nutné umožnit uživateli nahrát do aplikace vlastní graf/silniční síť. Je tedy nutné určit formát zápisu grafu do souboru. Měl by to být nějaký běžně používaný formát pro tento typ dat. Nabízí se například CSV, XML nebo JSON. Ovšem musíme brát v potaz také dostupná testovací data [6] a jejich formát. Některá testovací data jsou ve formátu GeoJSON. Tento formát je podobný JSONu, ale je přizpůsobený pro geografická data, která jsou podobná datům potřebným pro sestavení grafu či silniční sítě. Formát je tedy pro tento typ dat přehledný a dokonce existují i webové stránky, které dokáží tento formát zpracovat a zobrazit silniční síť na mapě [22].

S problémem formátu zápisu grafu souvisí fakt, že většina souborů v testovacích datech obsahuje jen vrcholy (křižovatky), nebo jen hrany (silnice). Bude tedy potřeba tyto soubory spojit do jednoho, který bude obsahovat celý graf (silniční síť). Řešením tohoto problému by mohl být built-in parser těchto souborů přímo ve vytvářené aplikaci, jehož výstup by byl jeden soubor s kompletní silniční sítí. Tento soubor už by byl dále použitelný pro zobrazení a dělení grafu.

5.2 Vložení vlastního dělícího algoritmu

Jak již bylo uvedeno výše, jeden z požadavků na aplikaci je možnost vložení dalších dělících algoritmů do už hotové přeložené aplikace. V Kap. 5.1.1 bylo rozhodnuto, že aplikace bude napsaná v jazyce Java. Java umožňuje přidat za běhu další externí třídy, které už jsou také přeložené, ale nejsou součástí aplikace. Této vlastnosti, které se říká reflexe, bude tedy využito pro přidání nových algoritmů.

Abychom mohli s novými třídami algoritmů manipulovat, je nutné vytvořit programově nějaký předpis, co musí tyto třídy splňovat. Pro tento účel bude v aplikaci vytvořeno rozhraní, které musí třída obsahující algoritmus implementovat, nebo třída, ze které musí třída s algoritmem dědit. Uživatel, který chce přidat nový algoritmus, zajistí,aby třída s dělícím algoritmem implementovala dané rozhraní, nebo dědila z dané třídy. Poté vytvoří JAR soubor, který pak aplikace otevře a najde v něm tuto třídu s algoritmem, ze které pomocí reflexe vytvoří novou instanci

5.3 Implementace grafu

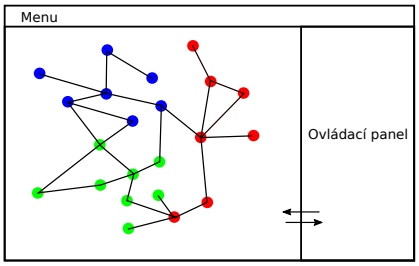
Samotný graf se dá implementovat různými způsoby. Nabízí se matice sousednosti, seznam sousednosti či plexová struktura (propojení hran a vrcholů přímými referencemi dle topologie grafu. Vzhledem k tomu, že grafy představující silniční síť mohou být poměrně velké (tisíce i desetitisíce vrcholů) a zároveň jsou poměrně řídké (většina vrcholů má maximálně 4 hrany), není implementace maticí sousednosti vhodná, obzvláště s ohledem na spotřebovanou paměť. Seznam sousednosti či plexová struktura jsou vhodnější. Protože plexová struktura lépe odpovídá skutečné topologii grafu, bude v práci využita. Pro snadný přístup k jednotlivým vrcholům a hranám budou tyto rovněž umístěny do samostatných seznamů.

5.4 Export jednotlivých částí grafu

V neposlední řadě bude také potřeba exportovat jednotlivé části grafu do souboru. Tento soubor bude ve formátu GeoJSON. Je ale nutné vyřešit zapisování společných hran částí. U každé takové hrany známe počáteční i koncový vrchol, její délku a kapacitu. Běžnou praxí je, že se hrany spojující dvě různé části dělí na polovinu. Tudíž by bylo dobré zapsat do souboru vždy polovinu této hrany, která bude uměle zakončena novým bodem ležícím v její jedné polovině. Podle těchto nových bodů by pak mělo být poznat, jak se má graf z jednotlivých částí složit, aby odpovídal původnímu grafu. Přidělíme-li stejné ID k oběma novým vrcholům dělícím hranu (každý vrchol v jedné části), můžeme tak snadno určit, které hrany spojit v jednu.

5.5 Návrh hlavního okna

Hlavní okno musí co nejlépe zobrazovat graf na co největší ploše, a zároveň by ovládací prvky grafu měly být vždy k dispozici. Bylo tedy navrženo hlavní okno (viz Obr. 5.1), kde na levé straně je prostor pro vykreslení grafu a na pravé je ovládací panel. Velikosti těchto dvou částí se dají upravovat.



Obrázek 5.1: Přibližný návrh hlavního okna.

6 Implementace testovací aplikace a vybraných algoritmů

Jak již bylo zmíněno v Kap. 5, aplikace byla napsána v jazyce Java (přesněji Java 11) a pro vytvoření uživatelského rozhraní byla použita softwarová platforma JavaFX.

6.1 Architektura aplikace

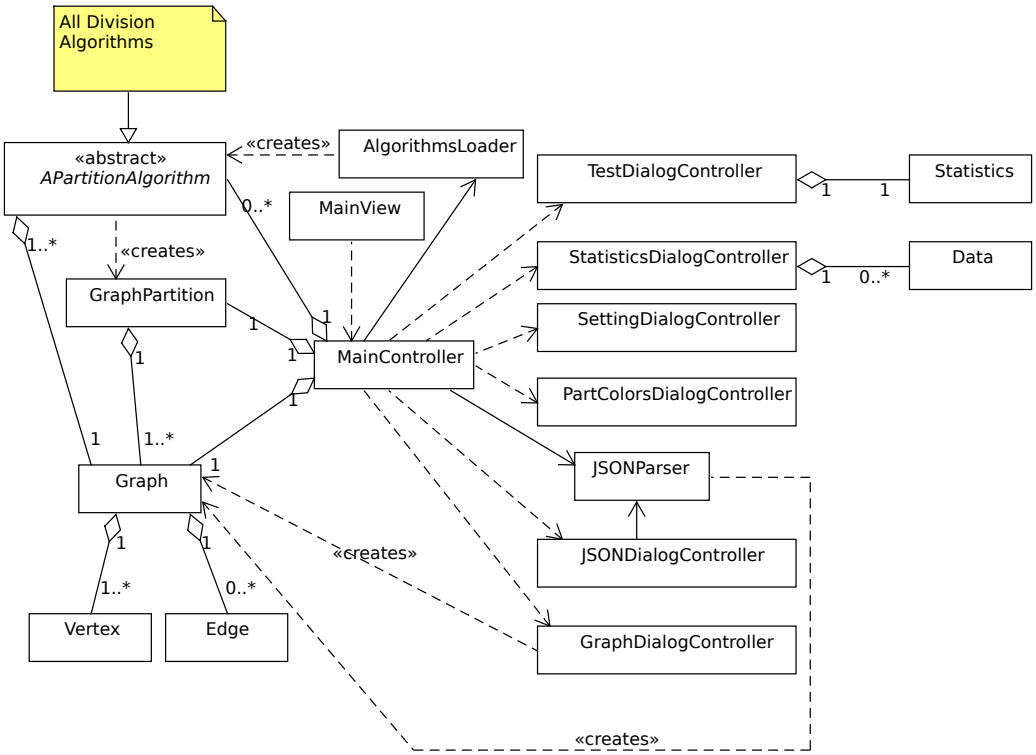
Aplikace je postavená na třívrstvé architektuře. Skládá se tedy s pohledů (Views), kontrolerů (Controllers) a modelu. Pohledy jsou pro přehlednost zapsány ve FXML. Platí, že ke každému oknu je napsán jeden pohled a každý pohled má vždy svůj kontroler. Přehled tříd je vidět v UML diagramu tříd 6.1. Aplikace má pouze jedno hlavní okno a 6 dialogových oken.

6.2 Přidání a načtení dostupných dělících algoritmů

Jedním z hlavních požadavků je přidání nových dělících algoritmů do již hotové aplikace. Tato funkčnost byla implementována pomocí balíku `java.lang.reflect` a je implementována ve třídě `AlgorithmsLoader`. Podmínkami pro načtení dělícího algoritmu je:

1. Jeho implementace musí být napsaná ve stejném programovacím jazyce.
2. Celá jeho implementace musí být zkompilovaná a její `.class` soubory musejí být v jednom JAR souboru a ten musí být ve složce `lib`.
3. V jeho implementaci se musí nacházet neabstraktní třída, která dědí od abstraktní třídy `APartitionAlgorithm`, která se nachází ve vytvořené aplikaci.

Veřejná metoda `findAlgorithms()` ve třídě `AlgoritmsLoader` otevře pomocí instance třídy `URLClassLoader` JAR soubor a získá z něj dostupné třídy.



Obrázek 6.1: UML diagram tříd.

Dále nalezne potomky třídy `APartitionAlgorithm` a vytvoří pomocí metody `getDeclaredConstructor()` ze třídy `Class<?>` instanci konstruktoru, který vytvoří novou instanci třídy `APartitionAlgorithm`. Metoda `findAlgorithms()` nakonec vrátí mapu s vytvořenými instancemi algoritmů.

6.3 Reprezentace grafu

Jak již bylo zmíněno v Kap. 5, graf je reprezentován pomocí vrcholů a hran. Byla tedy vytvořena třída `Graph`, která v sobě uchovává odkazy na instance tříd `Vertex` a `Edge`. Instance jsou rozděleny do dvou Map - `vertices` a `edges`. Klíčem v mapě je vždy ID instance. Třída `Vertex` dědí od knihovní třídy Point2D a uchovává v sobě odkazy na počáteční a koncové hrany pro rychlejší procházení grafem. Instance třídy `Edge` má zase v sobě uložen počáteční a koncový vrchol této hrany, také pro rychlejší procházení grafem.

6.4 Reprezentace částí grafu

Pro reprezentaci dělení grafu byla vytvořena třída `GraphPartition`, kde jsou uložené všechny části jednoho grafu rozděleného jedním dělícím algoritmem. Jedna část je reprezentována instancí třídy graf, ve které jsou pouze ty vrcholy z původního grafu, které patří do dané části. Všechny části jsou uložené v seznamu. Třída `GraphPartition` také obsahuje metody pro zjištění kvality a vlastností daného dělení.

6.5 Vykreslení grafu

Metoda `drawGraph()` pro vykreslení grafu je implementována v kontroleru hlavního okna `MainController`. Parametry této metody jsou instance knihovní třídy `Group`, graf a barva grafu. Tato metoda totiž slouží také pro vykreslení každé části grafu s určitou barvou. V metodě se podle každého vrcholu grafu vytvoří bod knihovní třídy `Circle` a pro každou počáteční hranu tohoto vrcholu se vytvoří úsečka třídy Line. Instance tříd `Circle` a Line se poté přidají do instance `Group`, jejíž obsah se zobrazí uživateli jako graf.

6.6 Umožnění vlastního nastavení dělících algoritmů

Může se stát, že bude některý přidaný dělící algoritmus potřebovat možnost nastavování některých hodnot, které používá, proto bylo v aplikaci počítáno i s tímto. Jelikož musí být u každého přidávaného algoritmu jedna třída, která dědí z `APartitionAlgorithm`, musí tedy i implementovat všechny abstraktní metody, a to jsou:

- `createGraphPartition()`,
- `getName()`,
- `getDescription()`,
- `getAllCustomParameters()`,
- `getAllCustomParametersDescription()`.

Poslední dvě uvedené metody právě umožňují definici vlastních parametrů, které se poté dají měnit v dialogu s nastavením jednotlivých algoritmů. Tyto vlastní parametry musí mít unikátní název a uživatel může zadat pouze text, ovšem `String` se dá převést třeba na `Integer`, parametrem tedy může být téměř cokoliv. Jedním

z příkladů by mohl být některý z dělících algoritmů, který používá ještě nějaké dodatečné informace, které nejsou z GeoJSON souboru získávány aplikací. Proto může být jako jeden z vlastních parametrů cesta k souboru, kde se tyto dodatečné informace nacházejí, a dělící algoritmus si je tak může ze souboru získat sám a dále použít pro další výpočty.

6.7 Hromadné testování dělení

Toto testování je implementováno v kontroleru testovacího dialogu. Tento typ testování je užitečný, pokud je potřeba otestovat dělící algoritmus se stejným grafem a stejným nastavením. Toto je především dobré, je-li algoritmus nedeterministický, nebo je-li potřeba zjistit co nejpřesněji dobu trvání jednoho dělení.

Testování vždy probíhá tak, že se postupně pro každý algoritmus udělá daný počet dělení a výsledky jednotlivých dělení se zaznamenávají do instance třídy `Statistics`. Po dotestování posledního algoritmu se výsledky zaznamenají do souboru, samozřejmě jen pokud je tato možnost zvolena. Dále se také mohou zapisovat do souborů i jednotlivá dělení, kde v jednom GeoJSON souboru je vždy jen jedna část grafu.

6.8 Implementace vybraných algoritmů

Vybrané algoritmy byly implementovány každý v novém modulu uvnitř hlavního modulu a byly označeny jako artefakty projektu. V každém modulu dělícího algoritmu jsou pouze jedna až dvě třídy a tyto moduly neobsahují hlavní třídu a tedy ani metodu `main`. Nejsou tedy určeny pro použití mimo testovací aplikaci.

6.8.1 METIS

Implementace METISu se skládá ze dvou tříd: `MetisAlgorithm` a `MetisVertex`. Ve třídě `MetisAlgorithm` se nachází samotná implementace algoritmu a třída `MetisVertex` vytváří instance vrcholů zhrubnutého grafu. Tento vrchol se skládá z několika vrcholů vstupního grafu. Váha tohoto hromadného vrcholu se rovná součtu všech obsažených vrcholů a hran mezi nimi původního grafu. Ve třídě `MetisVertex` jsou také metody pro získání všech vstupních a výstupních hran z tohoto hromadného vrcholu.

Třída `MetisAlgorithm` dědí od `APartitionAlgorithm` a obsahuje hlavní logiku METISu. Je zde především implementovaná metoda `createGraphPartition()`, ve které je provedeno hlavní dělení. Toto dělení je provedeno v cyklu `while`, ve kterém se graf dělí, dokud nedosáhne požadovaného počtu částí.

Pro každý graf nebo část grafu se znovu provádí zhrubování, zjemňování a samozřejmě dělení. Samotná metoda pro dělení grafu zkouší 4krát rozdělit graf a poté vybírá to nejlepší dělení nejen podle velikosti řezu, ale také podle vyváženosti dělení, jelikož se může stát, že dělíme-li graf na 4 části, tak nám může při prvním dělení vzniknout nespojitý graf. Při dělení nespojitého grafu se pak snadno může stát, že dělící metoda začne vytvářet dělení právě v mnohem menší nespojené komponentě nespojitého grafu.

6.8.2 SParTSim

Třída `SpartsimAlgorithm` obsahuje kompletní implementaci algoritmu SParTSim. Třída dědí od abstraktní třídy `APartitionAlgorithm` a implementuje metodu `createGraphPartition()`. V metodě `createGraphPartition()` není zapotřebí cyklus `while`, jako je tomu například u algoritmu Inertial Flow popsaného níže, jelikož algoritmus dělí graf přímo na požadovaný počet částí.

6.8.3 Inertial Flow

Implementace Inertial Flow obsahuje 3 třídy: `InertialFlowAlgorithm`, `IFVertex`, `IFEdge`. Třída IFVertex vytvoří instanci jednoho vrcholu pro algoritmus Inertial Flow. Tato instance obsahuje seznam vrcholů původního grafu patřící do tohoto vrcholu. Dále jsou zde metody pro získání počátečních hran, koncových hran, reverzní hrany a úrovně v grafu. Instance třídy `IFEdge` představují hranu spojující dvě instance `IFVertex` a mající vlastní kapacitu, tok a index v seznamu toků grafem použitého v rámci Dinicova algoritmu.

Třída `InertialFlowAlgorithm` dědí od `APartitionAlgorithm` a je zde hlavní implementace algoritmu. Je zde implementována metoda `createGraphPartition()` a použit podobný `while` cyklus jako v algoritmu METIS. Vstupní přímka je jen jedna, proto se promítnutí vrcholů a následné seřazení provádí mimo cyklus `while` pouze jednou a následně se vybírají vrcholy patřící do právě dělené části.

6.9 Jednotkové testování a testování podle scénářů

V rámci testování vytvořené aplikace byly použity JUnit5 testy. Vzhledem ke komplexnosti některých metod a obtížnosti vytvoření vstupu pro test a jeho očekávaného výstupu bylo napsáno pouze 6 jednotkových testů a bylo podstatně více využito testování podle scénářů. Jednotkovými testy byla otestována jen základní funkčnost aplikace. Zároveň byly testovány pouze neprivátní metody, kterých v aplikaci není mnoho. Jednotkovými testy bylo otestováno načtení dělících algoritmů do aplikace,

některé složitější metody ve třídě `Graph` a čtení souborů ve třídě `JSONParser`. Jednotkové testy odhalily 2 chyby.

Dále byla aplikace i vybrané algoritmy otestovány podle scénářů. Toto testování se ukázalo jako mnohem užitečnější. Různých scénářů bylo vytvořeno celkem 11 a téměř každý z nich lze zkusit vícekrát díky možnosti použití různých vstupním dat (ukázka - viz příloha A). Tímto typem testování byla odhalena většina chyb (cca 15 chyb).

Testování a porovnání vybraných algoritmů

7

Implementované metody byly pomocí vytvořené aplikace zevrubně otestovány a porovnány.

7.1 Popis testů

Testováno bylo dělení na 2, 4 a 6 částí. Každé dělení proběhlo 7krát se stejným nastavením, především kvůli měření času výpočtu - 2 extrémní hodnoty nebyly započítány do průměru, k výpočtu průměru tedy bylo použito 5 hodnot. U testování deterministických algoritmů budou všechny vlastnosti kromě času při stejném vstupu totožné.

Pro testování vybraných algoritmů byla použita vytvořená aplikace, proto byl samotný proces testování poměrně jednoduchý, z velké části automatizovaný a rychlý. Hodnoty vlastností každého dělení byly počítány aplikací a na konci každého testování zapsány do CSV souboru. Nastavení parametrů pro jednotlivá dělení bylo nastaveno na výchozí, kromě nastavení vyvážení u Inertial Flow, kde bylo v rámci této práce zjištěno, že hodnota 0,25 uvedená v [17], byla málo efektivní a byla tedy nahrazena hodnotou 0,45, se kterou bylo dosaženo lepších výsledků.

7.1.1 Testovací prostředí

Dělící algoritmy byly testovány na notebooku HP ZBook s procesorem Intel(R) Core(TM) i7-4810MQ CPU @ 2.80GHz, 2.80 GHz a RAM 16GB.

7.1.2 Silniční sítě použité k testování

Silniční sítě, které byly použity pro testování vybraných dělících algoritmů jsou uvedeny v Tab. 7.1. Záměrně byly vybrány sítě s různorodým počtem křižovatek a silnic - jedna malá, jedna střední a dvě velké sítě (viz počet křižovatek a silnic v Tab. 7.1). Silniční sítě byly získány z veřejně dostupného repozitáře [6] a jedná se o reálné silniční sítě. Název souboru odpovídá oblasti, kterou silniční síť reprezentuje.

Název	Počet vrcholů/křižovatek	Počet hran/silnic
`anaheim.geojson`	416	914
`goldcoast.geojson`	4783	11140
`chicagoregional.geojson`	12979	39018
`philadelphia.geojson`	13389	40003

Tabulka 7.1: Dělené silniční sítě [6]

7.1.3 Sledované vlastnosti dělení

Hodnoceny byly tyto vlastnosti dělení:

1. Čas - doba běhu aplikace měřená od začátku metody `getGraphPartition()` testovaného dělícího algoritmu do konce této metody.
2. Relativní odchylka od ideálního vyvážení - hodnota vypočtena podle vzorce 7.1:

$$RSD = \frac{\sqrt{\frac{\Sigma(x_i - \overline{x})^2}{k}}}{\overline{x}} * 100[\%] \,, \tag{7.1}$$

   kde x_i je hodnota části i, $\overline{x}$ je průměrná hodnota jedné části a k je počet částí.
3. Počet rozdělených silnic/hran - počet hran, které spojují dvě části.
4. Minimální počet sousedů - minimální počet částí sousedící s danou částí.
5. Maximální počet sousedů - maximální počet částí sousedící s danou částí.
6. Průměrný počet sousedů - průměrný počet částí sousedící s danou částí.

7.2 Porovnání vybraných algoritmů

Výsledky testování jednotlivých algoritmů jsou uvedeny v Tab. 7.2 , 7.3, 7.4 a 7.5.

7.2.1 Čas dělení

Z Tab. 7.2 , 7.3, 7.4 a 7.5 lze vidět, že čas dělení ve většině případů roste spolu s velikostí silniční sítě a požadovaným počtem částí u každého dělícího algoritmu. Ovšem toto nesplňuje ve dvou případech METIS, který v případě sítí chicagoregional.geojson a philadelphia.geojson má nižší časy při dělení sítě na 6 částí než při dělení na 4 části. Každopádně tvrzení, že čas roste podle velikosti sítě, platí u METISu také.

Dále je zřejmé, že nejrychlejším algoritmem je Inertial Flow, který dokáže dělit síť philadelphia.geojson na 6 částí za pouhých 2873 ms. Naopak nejpomalejší je SParTSim, který dokáže to samé za 2 399 351 ms (tedy více jak půl hodiny).

Algoritmus	Počet částí	Čas [ms]	Relativní odchylka [%]	Počet rozdělených hran	Min. počet sousedů	Max. počet sousedů	Prům. počet sousedů
Inertial Flow	2	1	12,5	69	1	1	1,0
Inertial Flow	4	2	18,1	223	1	2	1,5
Inertial Flow	6	2	54,1	355	1	3	2,0
SparTSim	2	29	12	225	1	1	1,0
SparTSim	4	407	25,6	269	3	3	3,0
SparTSim	6	491	26	331	2	5	4,0
METIS	2	140	0,62	162	1	1	1,0
METIS	4	216	4,7	378	3	3	3,0
METIS	6	232	34,2	520	3,8	5	4,6

Tabulka 7.2: Průměrné vlastnosti každého dělení silniční sítě `anaheim.geojson`.

7.2.2 Relativní odchylka od ideálního vyvážení

Relativní odchylka není již tolik závislá na velikosti sítě. V Tab. 7.2 , 7.3, 7.4 a 7.5 je vidět, že relativní odchylka sice v rámci jednoho dělícího algoritmu roste, ale tento růst je závislý spíše jen na počtu požadovaných částí. Výjimkou je SParTSim, u kterého odchylka při dělení sítě chicagoregional.geojson klesla z 16,7% na 11,27%. Relativní odchylku bude také určitě ovlivňovat rozložení ohodnocení grafu, hustota vrcholů v různých místech grafu a v některých případech i jeho vnější tvar.

Nejmenší relativní odchylka byla zjištěna u dělícího algoritmu METIS při dělení sítě chicagoregional.geojson na 2 části. Největší odchylka byla vypočtena také u METISu ale při dělení silniční sítě goldcoast.geojson na 6 částí.

7.2.3 Počet rozdělených hran

Počet rozdělených hran rostl především v závislosti na velikosti sítě a počtu požadovaných částí. Konzistentně nejmenší počet rozdělených hran měl vždy Inertial Flow. Největší počet rozdělených hran měl vždy METIS.

7.2.4 Minimální počet sousedů

V tomto ohledu byl nejlepší Inertial Flow, jelikož měl vždy minimálně pouze jednoho souseda. METIS a SParTSim na tom jsou v tomto případě podobně, oba měly v některých případech nejméně i 4 sousedy.

Algoritmus	Počet částí	Čas [ms]	Relativní odchylka [%]	Počet rozdělených hran	Min. počet sousedů	Max. počet sousedů	Prům. počet sousedů
Inertial Flow	2	188	2,03	127	1	1	1,0
Inertial Flow	4	230	15,9	375	1	2	1,5
Inertial Flow	6	229	39,33	567	1	2	1,67
SparTSim	2	1665	8,4	127	1	1	1,0
SparTSim	4	19008	14,2	559	3	3	3,0
SparTSim	6	26204	36	673	2	5	3,67
METIS	2	9870	0,71	548	1	1	1,0
METIS	4	13644	52,2	923	1	2,4	2,3
METIS	6	32619	73	1256	1,2	4	3,8

Tabulka 7.3: Průměrné vlastnosti každého dělení silniční sítě `goldcoast.geojson`.

Algoritmus	Počet částí	Čas [ms]	Relativní odchylka [%]	Počet rozdělených hran	Min. počet sousedů	Max. počet sousedů	Prům. počet sousedů
Inertial Flow	2	1312	2,3	347	1	1	1,0
Inertial Flow	4	1858	21,84	1011	1	2	1,5
Inertial Flow	6	2205	46,3	1585	1	2	1,67
SparTSim	2	21474	16,7	2585	1	1	1,0
SparTSim	4	1227741	11,27	4415	3	3	3,0
SparTSim	6	2762569	25	5683	3	5	4,0
METIS	2	547786	0,4	2648	1	1	1,0
METIS	4	719654	8,23	5019	3	3	3,0
METIS	6	515937	38,6	5753	4,2	5	4,8

Tabulka 7.4: Průměrné vlastnosti každého dělení silniční sítě `chicagoregional.geojson`.

7.2.5 Maximální počet sousedů

Zde byl opět nejlepší Inertial Flow, jelikož měl v jednom případě maximálně 3 sousedy a v ostatních případech 1 až 2 sousedy. METIS a SParTSim na tom jsou v tomto případě podobně, oba měly v některých případech maximálně i 5 sousedů.

7.2.6 Průměrný počet sousedů

Celkově byl v počtu sousedů nejlepší Inertial Flow, takže má nejmenší i průměrný počet sousedů. Úplně největší průměrný počet sousedů měl METIS při dělení sítě `chicagoregional` na 6 částí a to 4,8 sousedů.

7.2.7 Celkové hodnocení

Jelikož je tato práce hlavně zaměřená na dělení silniční sítě pro použití v distribuované simulaci dopravy, jsou některé vlastnosti dělení důležitější než jiné. Potřebný čas pro dělení má v našem případě jen malou důležitost, jelikož se dělení vybraným algoritmem bude provádět jen jednou a pak se již dá použít pro více simulačních běhů opakovaně. Naopak ostatní výše uvedené vlastnosti jsou pro simulaci mnohem více důležité.

Každý dělící algoritmus má svoje výhody a nevýhody. Například Inertial Flow se výborně hodí pro dělení, u kterého úplně není nutné vyvážení, ale je důležité, aby jednotlivé počítače/procesy měly mezi sebou co nejmenší potřebu komunikovat. Dělení SParTSimu je sice mnohem pomalejší než u Inertial Flow, nabízí ale kompromis mezi vyvážením částí a množstvím komunikace mezi počítači. METIS jako jediný vybraný nedeterministický algoritmus má pokaždé jiné výsledky, ovšem lze u něj po několika spuštěních dosáhnout opravdu malé relativní odchylky, což v některých případech může být také velice užitečné.

Algoritmus	Počet částí	Čas [ms]	Relativní odchylka [%]	Počet rozdělených hran	Min. počet sousedů	Max. počet sousedů	Prům. počet sousedů
Inertial Flow	2	1071	18,05	601	1	1	1,0
Inertial Flow	4	2346	22,26	1757	1	2	1,5
Inertial Flow	6	2873	56,72	2919	1	2	1,67
SparTSim	2	10554	0,8	1909	1	1	1,0
SparTSim	4	1132945	7,5	3579	3	3	3,0
SparTSim	6	2399351	13,19	4267	4	5	4,33
METIS	2	396015	0,44	2167	1	1	1,0
METIS	4	848867	3,54	4089	3	3	3,0
METIS	6	635248	50,03	5334	2,8	4,6	4,3

Tabulka 7.5: Průměrné vlastnosti každého dělení silniční sítě `philadelphia.geojson`.

Závěr 8

V rámci této práce byl vytvořen nástroj pro porovnání algoritmů pro dělení silniční sítě v jednotném prostředí a jednotnou sadou testů. Vytvořený nástroj je plně funkční a umožňuje opakované spuštění testů vybraného algoritmu pro dělení silniční sítě s vybranou dopravní sítí a dalšími parametry. V rámci práce byly rovněž implementovány tři různé algoritmy - METIS, SParTSim a Inertial Flow dle popisu jejich implementace ve vědeckých publikacích. Vlastnosti dělení sítě byly otestovány s využitím 4 různých dopravních sítích pro 3 různé počty částí. V rámci testů bylo sledováno několik parametrů dělení silniční sítě, např. vyvážení výsledných částí, počet sousedů, či počet rozdělených hran.

Jako nejstabilnější algoritmus s dobrým kompromisem mezi vyvážením a malým počtem rozdělených hran a malým počtem sousedů mezi částmi se ukázal SParTSim. Pro minimalizaci počtu rozdělených hran a počtu sousedů mezi částmi, s čímž právě souvisí množství komunikace mezi částmi, by se nejvíce hodil Inertial Flow, ovšem ten má často značně nevyvážené dělení. Závěrem lze tedy říci, že při volbě algoritmu, je dobré si ujasnit požadavky dělení a podle nich zvolit dělící algoritmus. Vyváženější dělení jde na úkor množství komunikace mezi částmi.

Je třeba zdůraznit, že implementace vybraných algoritmů nemusí být optimální. I když byly vybrány algoritmy, jejichž popis je dostatečně podrobný pro implementaci, některé části nebyly popsány zcela přesně a bylo nutné vytvořit vlastní implementaci těchto částí. Některé vybrané algoritmy navíc ve svém popisu obsahovaly více variant a pouze jedna varianta byla vybrána pro implementaci v rámci této práce. Dosažené výsledky tak neplatí pro zmíněné algoritmy obecně, ale pouze pro jejich implementovanou variantu. Provedené testování třech vybraných algoritmů tak hlavně demonstruje použitelnost vytvořeného nástroje. Možnost snadno dodat další algoritmy dělení, případně jejich varianty, umožňuje využití nástroje k dalšímu porovnání. Vytvořená aplikace i implementace vybraných algoritmů byly otestovány. Veškerý zdrojový kód, testovací data a spustitelná aplikace jsou k dispozici na Github.com [1], kde je možné si to stáhnout a otestovat vlastní dělící algoritmy.

[1] `https://github.com/Lulu1234/RoadNetworkPartitioning`

Ukázka testovacích scénářů

Ukázka 2 testovacích scénářů A.1 a A.2.

Krok	Akce	Předpokládaný výsledek	Odpovídá skutečný výsledek předpokládanému?
1	Otevřete z menu Graph -> New -> Generate Graph.	Zobrazeno dialogové okno s požadovanými údaji k vyplnění.	Ano
2	Zadejte následující hodnoty: Number of vertices horizontally: 20, Number of vertices vertically: 10, Length: 5.	Údaje jsou vyplněny.	Ano
3	Stiskněte tlačítko Create graph!.	Po stisku tlačítka se na levé straně UI objeví graf, který je pravidelný, mřížkový, s 20 vrcholy horizontálně, s 10 vrcholy vertikálně a jehož hrany mají délku 5.	Ano
4	Zopakujte tento scénář s různými hodnotami.	Po zopakování scénáře alespoň se třemi různými sadami hodnot je výsledek správný.	Ano

Tabulka A.1: Scénář 1 - Generování grafu

Krok	Akce	Předpokládaný výsledek	Odpovídá skutečný výsledek předpokládanému?
1	Otevřete z menu Parser -> Create GeoJSON File.	Zobrazeno dialogové okno s možností vložení souborů s hranami (sítí) a vrcholy (křižovatkami) a údaji k jejich zpracování.	Ano
2	Klikněte na tlačítko Upload file with graph edges.	Zobrazen výběr souborů.	Ano
3	Vyberte soubor s hranami grafu.	Po vybrání souboru a potvrzení výběru se v textovém poli pod tlačítkem zobrazí prvních 20 řádek souboru.	Ano
4	Doplňte údaje o souboru.	Údaje jsou vyplněny.	Ano
5	Klikněte na tlačítko Upload file with node coordinates.	Zobrazen výběr souborů.	Ano
6	Vyberte soubor s vrcholy grafu.	Po vybrání souboru a potvrzení výběru se v textovém poli pod tlačítkem zobrazí prvních 20 řádek souboru.	Ano
7	Doplňte údaje o souboru.	Údaje jsou vyplněny.	Ano
8	Klikněte na tlačítko Create GeoJSON File!	Ve složce s aplikací byl vytvořen nový soubor s příponou .geojson, který odpovídá sjednocení dvou vybraných souborů.	Ano
9	Zopakujte tento scénář s více různými soubory.	Po zopakování scénáře alespoň se třemi různými dvojicemi souborů je výsledek správný.	Ano

Tabulka A.2: Scénář 2 - JSON Parser

Uživatelská příručka B

Aplikace se spouští pomocí příkazové řádky příkazem:

```
java -jar RoadNetworkPartitioning.jar
```

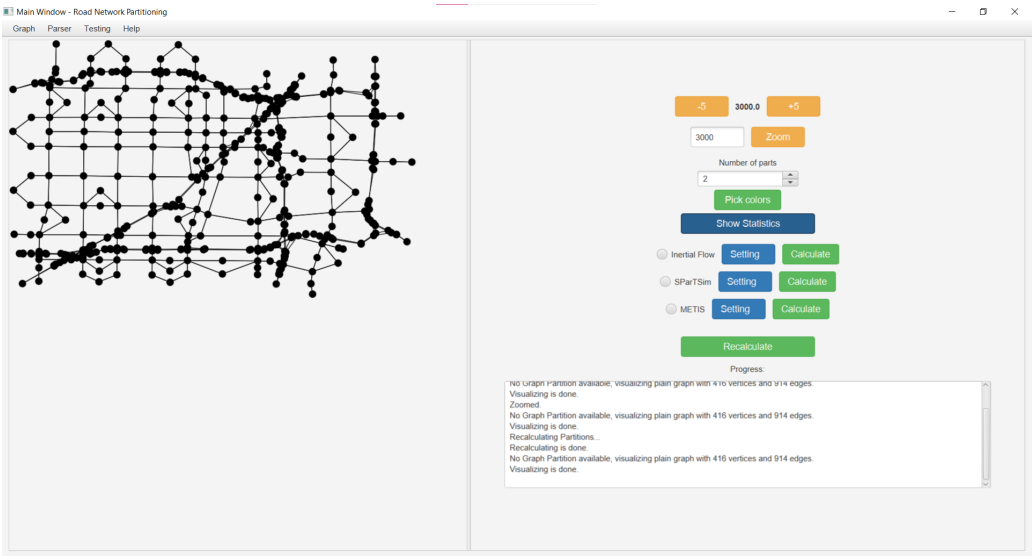
Uživatel musí mít nainstalovanou Javu 11 a výš. Po spuštění vypadá aplikace takto B.1. Po načtení nějakého grafu přes záložku Graph v menu se vykreslí graf viz B.2. Po zvolení nějakého dělícího algoritmu (kliknutí na tlačítko Calculate u jednoho z nich nebo na hromadné tlačítko Recalculate a zvolení příslušného radio buttonu) se rozdělený graf může zobrazit následovně B.3.

Pro hromadné testování dělení vyberte v menu Testing -> Test a zobrazí se dialog pro testování dělících algoritmů B.4. Pro přidání nových algoritmů do aplikace musí jejich implementace splnit následující podmínky:

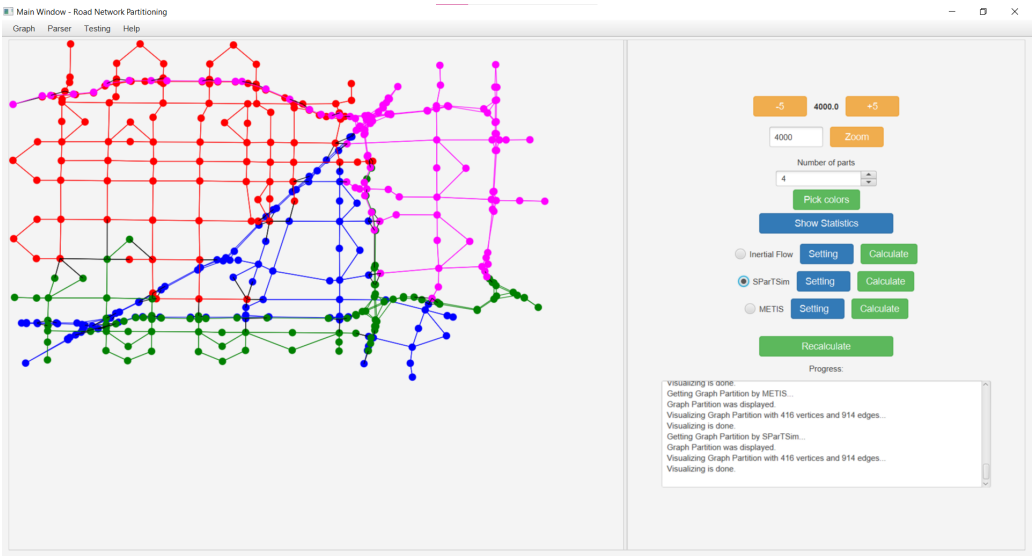
1. Jeho implementace musí být napsaná ve stejném programovacím jazyce.
2. Celá jeho implementace musí být zkompilovaná a její `.class` soubory musejí být v jednom JAR souboru a ten musí být ve složce `lib`.
3. V jeho implementaci se musí nacházet neabstraktní třída, která dědí od abstraktní třídy `APartitionAlgorithm`, která se nachází ve vytvořené aplikaci.

Více v elektronické příloze v souboru s názvem `testcases.pdf`, kde jsou uvedeny a popsány nejčastější případy užití.

Obrázek B.1: Úvodní obrazovka.



Obrázek B.2: Zobrazení grafu.



Obrázek B.3: Rozdělení grafu.

Obrázek B.4: Testovací dialog.

Bibliografie

1. KOVÁŘ, Petr. *Teorie grafů*. Petr Kovář, 2022. Dostupné také z: https://homel.vsb.cz/~kov16/files/skriptum_teorie_grafu_rozsirene.pdf.
2. KARYPIS, G.; AGGARWAL, R.; KUMAR, V.; SHEKHAR, S. Multilevel hypergraph partitioning: applications in VLSI domain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. 1999, roč. 7, č. 1, s. 69–79. Dostupné z DOI: 10.1109/92.748202.
3. ČADA, Roman. *Teorie sítí (velmi pracovní verze textu)*. Roman Čada, 2022. Dostupné také z: https://portal.zcu.cz/portal/studium/courseware/kma/tsi/prednasky.html.
4. SKIENA, Steven S. *The Algorithm Design Manual Second Edition*. New York, USA: Springer-Verlag London Limited, 2008. ISBN 978-1-84800-069-8.
5. KARYPIS, George; KUMAR, Vipin. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *Siam Journal on Scientific Computing*. 1999, roč. 20, č. 1, s. 359–392. Dostupné z DOI: 10.1137/S1064827595287997.
6. STABLER, Ben. *Transportation Networks for Research*. Transportation Networks for Research Core Team, 2023. Dostupné také z: https://github.com/bstabler/TransportationNetworks.
7. AÑEZ, J.; DE LA BARRA, T.; PÉREZ, B. Dual graph representation of transport networks. *Transportation Research Part B: Methodological*. 1996, roč. 30, č. 3, s. 209–216. ISSN 0191-2615. Dostupné z DOI: https://doi.org/10.1016/0191-2615(95)00024-0.
8. MARTÍNEZ, Luis M.; VIEGAS, José Manuel; SILVA, Elisabete A. A traffic analysis zone definition: a new methodology and algorithm. *Transportation*. 2009, roč. 36, s. 581–599. Dostupné z DOI: https://doi.org/10.1007/s11116-009-9214-z.
9. POTUŽÁK, Tomáš. Current Trends in Road Traffic Network Division for Distributed or Parallel Road Traffic Simulation. In: *2022 IEEE/ACM 26th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. 2022, s. 77–86. Dostupné z DOI: 10.1109/DS-RT55542.2022.9932112.

10. POTUŽÁK, Tomáš. Methods for Division of Road Traffic Networks Focused on Load-Balancing. *Advances in Computing*. 2012, roč. 2, s. 42–53. ISSN 2163-2944. Dostupné z DOI: 10.5923/j.ac.20120204.01.

11. VENTRESQUE, Anthony et al. SParTSim: A Space Partitioning Guided by Road Network for Distributed Traffic Simulations. In: *2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*. 2012, s. 202–209. Dostupné z DOI: 10.1109/DS-RT.2012.37.

12. DELLING, Daniel; GOLDBERG, Andrew V.; RAZENSHTEYN, Ilya; WERNECK, Renato F. Graph Partitioning with Natural Cuts. In: *2011 IEEE International Parallel Distributed Processing Symposium*. 2011, s. 1135–1146. Dostupné z DOI: 10.1109/IPDPS.2011.108.

13. XU, Yan; TAN, Gary. An Offline Road Network Partitioning Solution in Distributed Transportation Simulation. In: *2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*. 2012, s. 210–217. Dostupné z DOI: 10.1109/DS-RT.2012.38.

14. HENDRICKSON, B.; LELAND, R. A Multi-Level Algorithm For Partitioning Graphs. In: *Supercomputing '95:Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*. 1995, s. 28–28.

15. SANDERS, Peter; SEEMAIER, Daniel. Distributed Deep Multilevel Graph Partitioning. In: CANO, José; DIKAIAKOS, Marios D.; PAPADOPOULOS, George A.; PERICÀS, Miquel; SAKELLARIOU, Rizos (ed.). *Euro-Par 2023: Parallel Processing*. Cham: Springer Nature Switzerland, 2023, s. 443–457. ISBN 978-3-031-39698-4.

16. POTUŽÁK, Tomáš. Division of Road Traffic Network Based on Genetic Algorithm and Graph Coarsening. In: *2018 11th International Conference on Human System Interaction (HSI)*. 2018, s. 484–490. Dostupné z DOI: 10.1109/HSI.2018.8431334.

17. SCHILD, Aaron; SOMMER, Christian. On Balanced Separators in Road Networks. In: BAMPIS, Evripidis (ed.). *Experimental Algorithms*. Cham: Springer International Publishing, 2015, s. 286–297. ISBN 978-3-319-20086-6.

18. YU, Qing; LI, Weifeng; YANG, Dongyuan; ZHANG, Haoran. Partitioning urban road network based on travel speed correlation. *International Journal of Transportation Science and Technology*. 2021, roč. 10, č. 2, s. 97–109. Dostupné z DOI: https://doi.org/10.1016/j.ijtst.2021.01.002.

19. XIAOHUI LIN, Jianmin XU. Road network partitioning method based on Canopy-Kmeans clustering algorithm. *Archives of Transport*. 2020, roč. 54, č. 2, s. 95–106. ISSN 0866-9546. Dostupné z DOI: 10.5604/01.3001.0014.2970.

20. UMN-CSE, Karypis Lab @. *METIS*. Karypis Lab @ UMN-CSE, 2020. Dostupné také z: `https://github.com/KarypisLab/METIS`.

21. *Dinic's algorithm for Maximum Flow*. GeeksforGeeks, 2023. Dostupné také z: `https://www.geeksforgeeks.org/dinics-algorithm-maximum-flow/`.

22. *geojson.io*. mapbox, 2024. Dostupné také z: `https://geojson.io/#map=2/0/20`.

Seznam obrázků

Seznam tabulek