VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vizualizace grafových struktur

Visualization of Graph Structures

2016 Lukáš Štefánek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Štefánek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Vizualizace grafových struktur
Visualization of Graph Structures

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je naimplementovat vizualizaci grafových struktur. Výsledkem aplikace bude umožňovat zobrazení a základní práci s velkými grafovými strukturami.

1. Seznamte se a popište současné trendy v oblasti 3D vizualizace a prezentace dat, a to s ohledem na aktuální možnosti OpenGL a případné hardwarové akcelerace.
2. Popište možné formáty vstupních dat, analyzujte potřeby a přidanou hodnotu vizualizace, navrhněte způsob vizualizace.
4. Navrhněte a vytvořte aplikaci umožňující vizualizaci data různým způsobem, a to na základě vstupních dat a volbě typu vizualizace.
5. Proveďte výkonnostní experimenty vytvořené případové studie.
6. Vyhodnocení experimentů.

Seznam doporučené odborné literatury:

[1] M. Pilgrim: HTML5: Up and Running, O'Reilly Media; 1 edition (August 24, 2010), ISBN-13: 978-0596806026
[2] D. Crockford: JavaScript: The Good Parts, O'Reilly Media; 1st edition (May 2008), ISBN-13: 978-0596517748

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Gajdoš, Ph.D.**

Datum zadání: 01.09.2015
Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2016 .....

Rád bych zde poděkoval svému vedoucímu bakalářské práce Ing. Petru Gajdošovi, Ph.D. za cenné rady a jeho trpělivost při vedení mé bakalářské práce.

Abstrakt

Bakalářská práce se zabývá vytvořením aplikace pro vizualizaci grafových struktur, která bude umožňovat zobrazení a základní práci s grafovými strukturami. První část této práce popisuje současné trendy v oblasti vizualizace a práce s daty a jejich prezentací. Druhá část je zaměřená na výběr algoritmů a implementaci aplikace. Třetí část je zaměřena na výkonnostní testy aplikace ve zpracování dat za použití ruzných prohlížečů a při různých nastaveních.

Klíčová slova: HTML5, JavaScript, Canvas, layout, svazování hran, graf, animace

Abstract

The bachelor thesis deals with create appliaction for visualization graph structures, which will allow display and basic work with graph structures. The first part of this work describes current trends in visualization and work with data and their presentation. The second part is focused to selection algorithms and implementation application. The third part is focused on performance testing application in data processing using different browsers and at different settings.

Key Words: HTML5, JavaScript, Canvas, layout, edge bundling, graph, animation

Obsah

Seznam použitých zkratek a symbolů

LCF	–	Lederberg-Coxeter-Frucht, stručný zápis pro 3-pravidelný Hamiltonovský graf
NP	–	Nedeterministicky polynomiální
HEB	–	Hierarchical Edge Bundles
FDEB	–	Force-Directed Edge Bundling
GBEB	–	Geometry-Based Edge Bundling
MINGLE	–	Multilevel Agglomerative Edge Bundling
JSON	–	JavaScript Object Notation

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Cílem této práce je vytvořit aplikaci na vizualizaci grafových struktur při aplikaci layout a edge bundling algoritmů. Součástí této práce je seznámení a popsání současně používaných algoritmů v oblasti vizualizace a práce s grafy. Tyto algoritmy jsou obsaženy v nejpoužívanějších aplikacích a knihovnách zaměřených na vizualizaci grafů, jako jsou např. knihovny Data-Driven Documents(D3.js), Chart.js, Sigma.js a Cytoscape.js. Tyto knihovny jsou používány v mnoha odvětvích pro vizualizaci různých dat, jsou to např. trasy letadel, migrace, vztahy na sociálních sítích a jakékoliv data reprezentovaná grafem.

Implementovaná aplikace bude nabízet načtení základního grafu z disku počítače ve formátu JSON, který bude uložen v daných třídách a graf bude procházet algoritmy, které budou postupně upravovat jeho strukturu. Tato úprava bude záležet na nastavení algoritmů pomocí výchozích nastavení a parametrů z grafického rozhraní, které má možnost uživatel měnit i za běhu aplikace. Mezi nastavitelné parametry v grafickém rozhraní budou patřit následující: delta, curviness, angle, neighbors, margin, alpha, layout iterations, animation time, solid color, line type, color type, dataset a recursion. Aplikace bude obsahovat následující funkce: zoomování, přesouvání grafu po plátnu, přesouvání vrcholů grafu, možnost zobrazení informací o vrcholu, počítání procentuálního pokrytí plátna barvou a možnost zachycení zobrazovaného grafu jako obrázku.

V poslední části práce budou provedeny experimenty a výkonnostní testování aplikace. Experimety budou zaměřené na vliv parametrů grafického rozhraní na výslednou vizualizaci grafu. Testování proběhne při spuštění aplikace v různých prohlížečích a bude se měřit čas výpočtu výsledného grafu při nastavení určitých parametrů.

2 Kreslení grafů

Kreslení grafů je oblast matematiky a informatiky, kombinuje metody z geometrické teorie grafů a vizualizaci informací k vyobrazení dvourozměrných nebo třírozměrných grafů z aplikací např. na sociální síťové analýzy, kartografie, lingvistiku a bioinformatiku.

Kresba grafu nebo diagramu sítě je vyobrazením vrcholů a hran grafu. Výsledek kresby by neměl být zaměnován s původním grafem. Velmi rozdílný výsledný vzhled grafu může odpovídat pouze jednomu původnímu grafu. V abstraktním grafu záleží jen na tom, které dvojice vrcholů jsou spojeny hranami. V pevném grafu však uspořádání vrcholů a hran ovlivňuje srozumitelnost, použitelnost, cenu zhotovení a estetiku. Problém se zhoršuje jestliže měníme graf v čase přidáváním a odebíráním hran(dynamické kreslení grafu).

Grafy jsou často kresleny jako uzel-spoj diagramy, ve kterých jsou vrcholy reprezentovány jako disky, krabice nebo textové štítky a hrany jsou reprezentovány jako úcečky, křivky nebo křivky v Euklidovském prostoru. Historie uzel-spoj diagramů sahá až do 13. století k práci Ramona Llull, který kreslil grafy tohoto typu pro úplný graf za účelem analyzovat všechny párové kombinace mezi sadami metafyzických konceptů.

V případě orientovaných grafů se někdy používají šipky k určení směru hrany. Nicméně uživatelské studie ukázaly, že nahrazení šipky zužující se hranou poskytne informaci směru efektivněji. Vzestupné rovinné kreslení grafu používá rozložení vrcholů tak, že šipky jdou směrem od spodních vrcholů k vrchním vrcholům, tudíž v tomto případě jsou šipky zbytečné. [5]

Zajištění kvality

Mnoho různých opatření v oblasti kvality bylo definováno pro kreslení grafů, ve snaze najít objektivní způsob hodnocení estetiky a použitelnosti. Na výběr je mnoho algoritmů na uspořádání jednoho původního grafu, některé z těchto algoritmů optimalizují tato opatření.

- Pojem počet křížení představuje v grafové terminologii počet křížení hran mezi sebou navzájem. V případě že je graf rovinný, je vhodné ho nakreslit bez křížení hran, v tomto případě grafové kreslení reprezentuje grafové vkládání. Nicméně neplanární grafy většinou vznikají v aplikacích, takže kreslící algoritmy musí obecně počítat s křížením hran.

- Plocha pro kreslení je jako nejmenší rámeček, který musí obsahovat všechny vrcholy grafu. Kreslení na větší plochu je obecně lepší než kreslení na menší plochu, protože to umožnuje lepší viditelnost detailů grafu než u měnší plochy a graf bude lépe čitelný. Důležitý je i poměr stran plochy na kreslení.

- Další problém je nalezení symetrických částí grafu, nalezení způsobu vykreslení výsledného grafu, tak aby byl co nejvíce symetrický. Některé algoritmy na úpravu grafů se automaticky snaží kreslit symetrické grafy, některé začínají tím že najdou symetrické struktury v grafu a využijí tuto konstrukci ke kreslení výsledného grafu.[2]

- Je důležité, aby hrany měly co nejjednodušší tvar, aby graf byl přehledný a bylo zřejmé kam hrana vede. Při kreslení hran jako křivky může být složitost měřena počtem řídících bodů křivek.
- Pro zlepšení kvality výsledného grafu je vhodné minimalizovat délky hran v grafu, může být výhodné, aby délky hran v grafu byly podobné, než aby se velmi lišily.
- Úhlové rozlišení je měřítkem nejostřejšího úhlu v grafu. V případě, že jsou v grafu vrcholy s vysokým stupněm, pak musí mít malé úhlové rozlišení, ale může být dole omezeno v závislosti na úhlu.

Algoritmy pro specifické kreslení grafů:
Force-based layout, Spectral layout, Orthogonal layout, Tree layout, Arc diagrams, Circular layout, Dominance drawning.

Software na kreslení grafů:
Cytoscape, BioFabric, Gephi, Graphviz, Mathematica, Microsoft Automatic Graph Layout, Tom Sawyer Software, Tulip, yEd, PGF/TikZ.

2.1 Datové struktury grafů

Datová struktura grafu se skládá z vrcholů (uzlů, bodů), neuspořádané páry těchto vrcholů pro neorientovaný graf a uspořádané páry pro orientovaný. Tyto páry jsou známé jako hrany(oblouky, křivky, linky), u orientovaného grafu jsou reprezentovány šipkou. Datová struktura grafu může obsahovat i jiné informace než pozice a propojení mezi vrcholy, například u vrcholů mohou obsahovat jméno, popis, barvu, tvar, geo-souřadnice a velikost, u hran to může být váha, barva a další informace o spojení dvou vrcholů. [18]

Seznam sousedů

Je jeden z několika běžně používaných reprezentací grafu, používaných v počítačových programech. Je kolekcí neuspořádaných seznamů použitých k reprezentaci konečného grafu. Každý seznam popisuje sousedy daného vrcholu. Výhodou této reprezentace je rychlost práce s grafem, časové složitosti operací jsou: uložení grafu - O(|V| + |E|), přidání vrcholu nebo hrany - O(1), odebrání vrcholu nebo hrany O(|E|).

Matice sousednosti

Matice sousednosti je čtvercová matice řádu $|V^2|$, která reprezentuje konečný graf. Elementy matice zaznamenávají, jestli jsou páry vrcholů spojeny hranou. Ve speciálních případech konečného jednoduchého grafu je matice sousednosti tvořena hodnotami 0 a 1, kde 0 jsou na diagonále matice. Pokud je graf neorientovaný, tak je matice symetrická. Oproti seznamu sousedů(Adjacency list) má tato struktura časově náročnější operace s grafem: uložení grafu - O($|V^2|$), přidání nebo odebrání vrcholu - O($|V^2|$), přidání nebo odebrání hrany O(1).[12]

Matice incidence

Je matice, kde je pro každý vrchol grafu jeden řádek a pro hranu grafu jeden sloupec. Pokud vrchol náleží hraně, je na dané pozici hodnota 1, pokud nenáleží 0. U orientovaného grafu se značí výchozí vrchol hodnotou -1. U této datové struktury je pro přidání a odebrání hrany, přidání a odebrání vrcholu, uložení grafu časová složitost O($|V| \times |E|$).

2.2 Layoutovací algoritmy

2.2.1 Force-Directed layout

Force-directed algoritmy jsou ze třídy Algoritmy pro kreslení grafů, které mají za účel zpřehlednit a esteticky upravit vzhled výsledného grafu. Algoritmus funguje na principu porovnávání vzdálenosti všech bodů a délky všech hran, kde spočítá síly(odpudivá síla a přitažlivá síla, atd.), pomocí kterých přemístí body po ploše.
Přitažlivá síla je počítána na základě Hookova zákona, počítá se mezi koncovými body hran, projdou se všechny hrany v grafu.
Odpudivá síla je počítána na základě Columbova zákona, počítá se mezi všemi body navzájem v celém grafu.

Force-directed layout může obsahovat i jiné síly než jen přitažlivé a odpudivé. Síla podobná gravitaci, může být použita k vytažení vrcholů směrem k nějakému bodu plátna, může být použita ke spojení nesouvislých grafů. Síla podobná magnetickému poli může být použita u orientovaných grafů. Odpudivá síla může být počítána jak na vrcholech, tak i na hranách, aby se ve výsledku zabránilo překrývání. V grafech tvořených křivkami mohou být síly použity na řídících bodech křivek, například za účelem zlepšení jejich úhlového rozlišení.

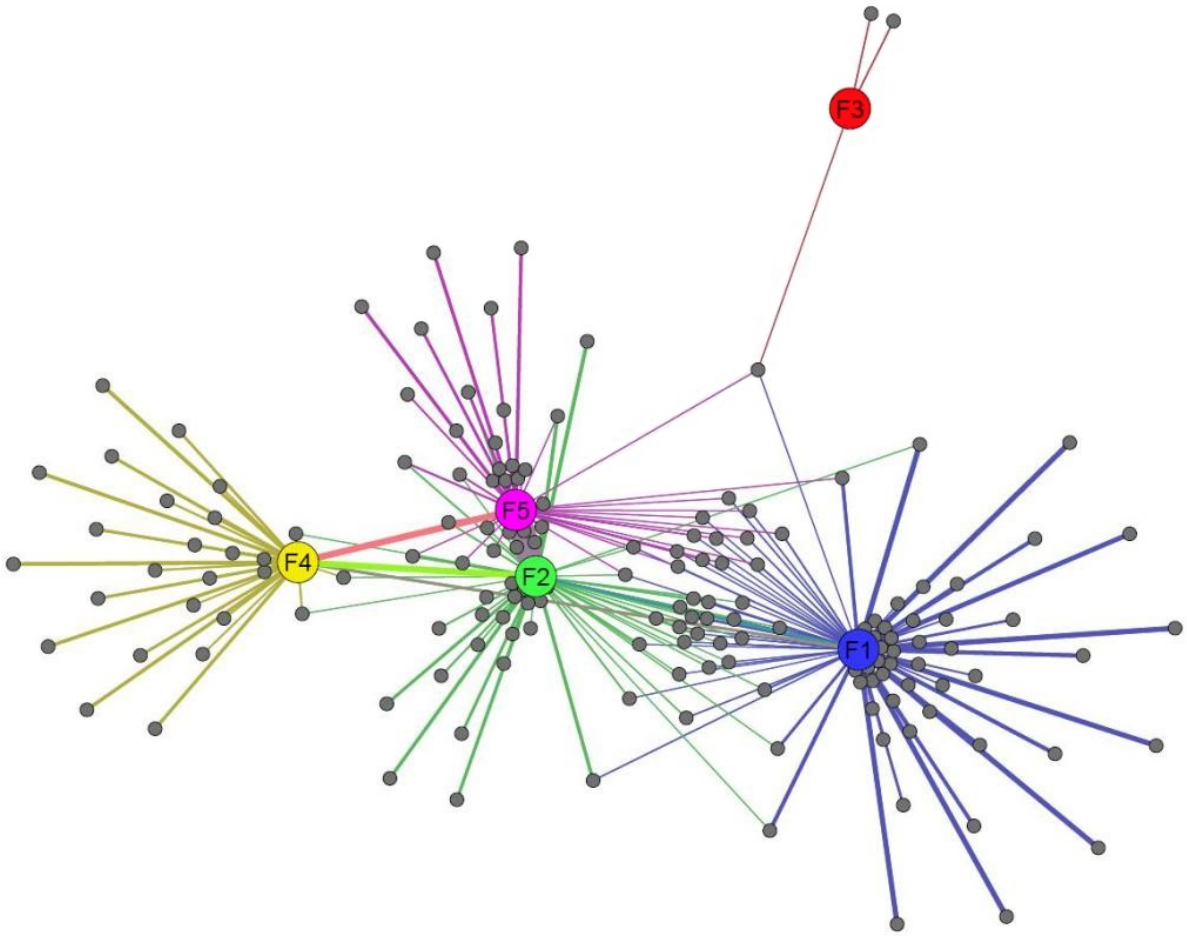
Poté, co se pro celý graf spočítaly všechny síly, aplikují se na vrcholy grafu. Tento celý proces bývá většinou opakován, dokud síly nejsou v rovnováze a pozice vrcholů grafu se nemění. Počet průchodů tímto algoritmem může být dán předem nebo dynamicky v běhu programu. [16]

Výhody:
Kvalitní výsledek - pro velikostně střední grafy(50 - 500 vrcholů), dobrý výsledek je hodnocen na základě následujících kriterii: stejné délky hran, rovnoměrné rozmístění vrcholů a zobrazovací symetrie.
Flexibilita - algoritmus lze snadno přizpůsobit nebo rozšířit podle dodatečných kriterií. Rozšíření jako 3D kreslení grafů, kreslení grafů po svazcích a dynamické kreslení.
Jednoduchost - force-directed algoritmy mohou být jednoduše iplementovány jen na pár řádků kódu, většina layout algoritmů je složitější.
Interaktivita - postupným kreslením průběhu algoritmu, může uživatel sledovat rozmísťování vrcholů po plátně od spleti vrcholů a hran až po dobře vypadající graf. V některých interaktivních nástrojích pro kreslení grafů si uživatel může zvolit skupinu vrcholů a naopak spočítat původní polohu vrcholů, to z těchto algoritmů dělá nejpoužívanější způsob pro kreslení dynamických a on-line grafů. [1]

Nevýhody:

Časová náročnost - typická časová naročnost algoritmů je O(n^3), kde n je počet vrcholů. Je to proto, že počet průchodů má časovou náročnost O(n), v každém průchodu se počítá odpudivá síla mezi všemi body s náročností O(n^2) a přitažlivá síla se složitostí O(n).

Slabé lokální minimum - algoritmus produkuje graf s minimální energií zejména tehdy, pokud je celková energie pouze lokální minimum, což má za následek nízkou kvalitu výsledku. Pro mnoho algoritmů ze třídy force-directed, zejména ty, které umožnůjí pouze "down hill"pohyby, konečný výsledek může být silně ovlivněn prvotním uspořádáním vrcholů. Problém špatného lokálního minima roste společně s počtem vrcholů grafu. Kombinování aplikací různých algoritmů může řešit tento problém. Například použitím Kamada–Kawai algoritmu k rychlému generování přiměřeného počátečního rozložení a poté použít Fruchterman–Reingold algoritmus pro zlepšení umístění sousedních vrcholů. Další technikou pro dosažení globálního minima je použití víceúrovňového přístupu. [3]



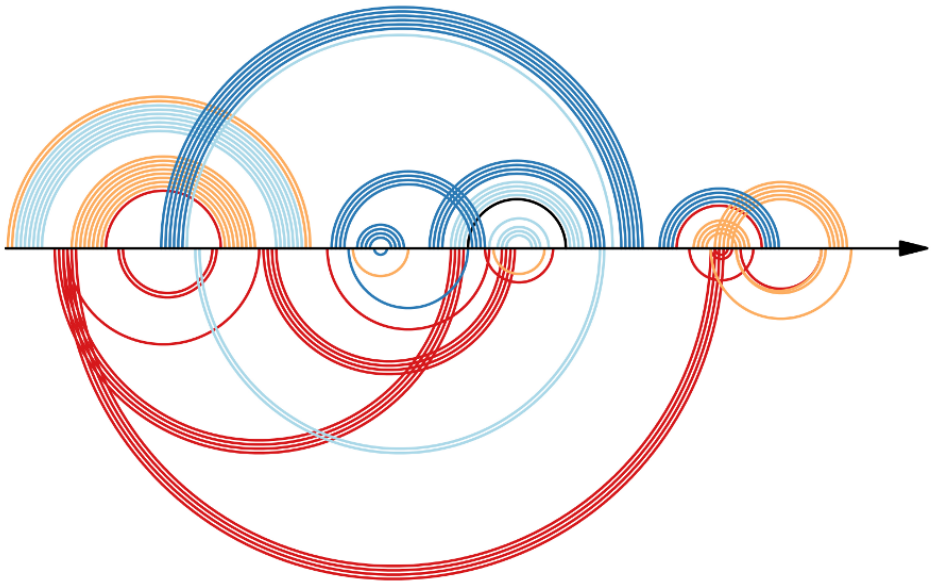
Obrázek 1: Force-directed layout. [20]

2.2.2 Arc diagram

Arc diagram je styl kreslení grafů, ve kterém jsou vrcholy grafu umístěny podél linie v Euklidovské rovině, přičemž hrany jsou nakresleny jako půlkruhy v jedné ze dvou polovičních rovin rozdělených linií, nebo jako hladké křivky, tvořené sekvencemi půlkruhů. V některých případech jsou segmenty linií brány jako hrany tak dlouho, dokud se nepropojí pouze s vrcholy, které jsou podél linie.

Použití výrazu obloukový diagram pro tento typ kreslení následuje použití podobného typu diagramu od Wattenberga(2002) pro vizualizaci opakování vzorů v řetězcích, pomocí oblouků spojuje páry stejných řetězců. Nicméně tento styl kreslení grafů je mnohem starší než jeho jméno, datuje se až k práci Saaty(1964) a Nicholsona (1968), kteří použili obloukové diagramy ke studiu počtu křížení v grafu. Starší, ale méně často užívaný název pro obloukové diagramy je lineární vkládání.

Nicholson zjistil, že každé vkládání grafu do roviny může být deformováno na obloukový diagram, aniž by se změnil počet křížení hran. Zejména každý rovinný graf má rovinný obloukový diagram. Nicméně toto vkládání může potřebovat více než jeden půlkruh pro některé z hran. Pokud je graf nakreslen bez křížení hran s použitím obloukového diagramu, ve kterém každá hrana je jeden půlkruh, pak je kreslení definováno jako dvoustránkové vkládání, které je možné použít u dílčích Hamiltonovských grafů, které jsou podmnožinou rovinných grafů. Například maximální rovinný graf má takové vkládání jen tehdy, pokud obsahuje Hamiltonovský cyklus. Z toho důvodu ne-Hamiltonovský maximální rovinný graf , jako například Goldner-Harary graf, nemůže mít rovinné vkládání jednoho půlkruhu na jednu hranu. Každý rovinný graf má obloukový diagram, ve kterém je každá hrana tvořena maximálně dvěma půlkruhy. [7]



Obrázek 2: Arc diagram. [17]

2.2.3 Circular layout

Circular layout je styl kreslení grafů, který rozmístí vrcholy grafu po kružnici, často jsou rozmístěny tak, že tvoří pravidelný mnohoúhelník. Tyto algoritmy jsou dobrou volbou pro kreslení komunikační sítové topologie, jako hvězdové nebo kruhové sítě a pro cyklické části metabolických sítí. Pro grafy s Hamiltonovským cyklem, kruhové algoritmy umožnují zobrazit cyklus jako kruh, a tím způsobem tvoří základ LCF notací pro Hamiltonovské kubické grafy.

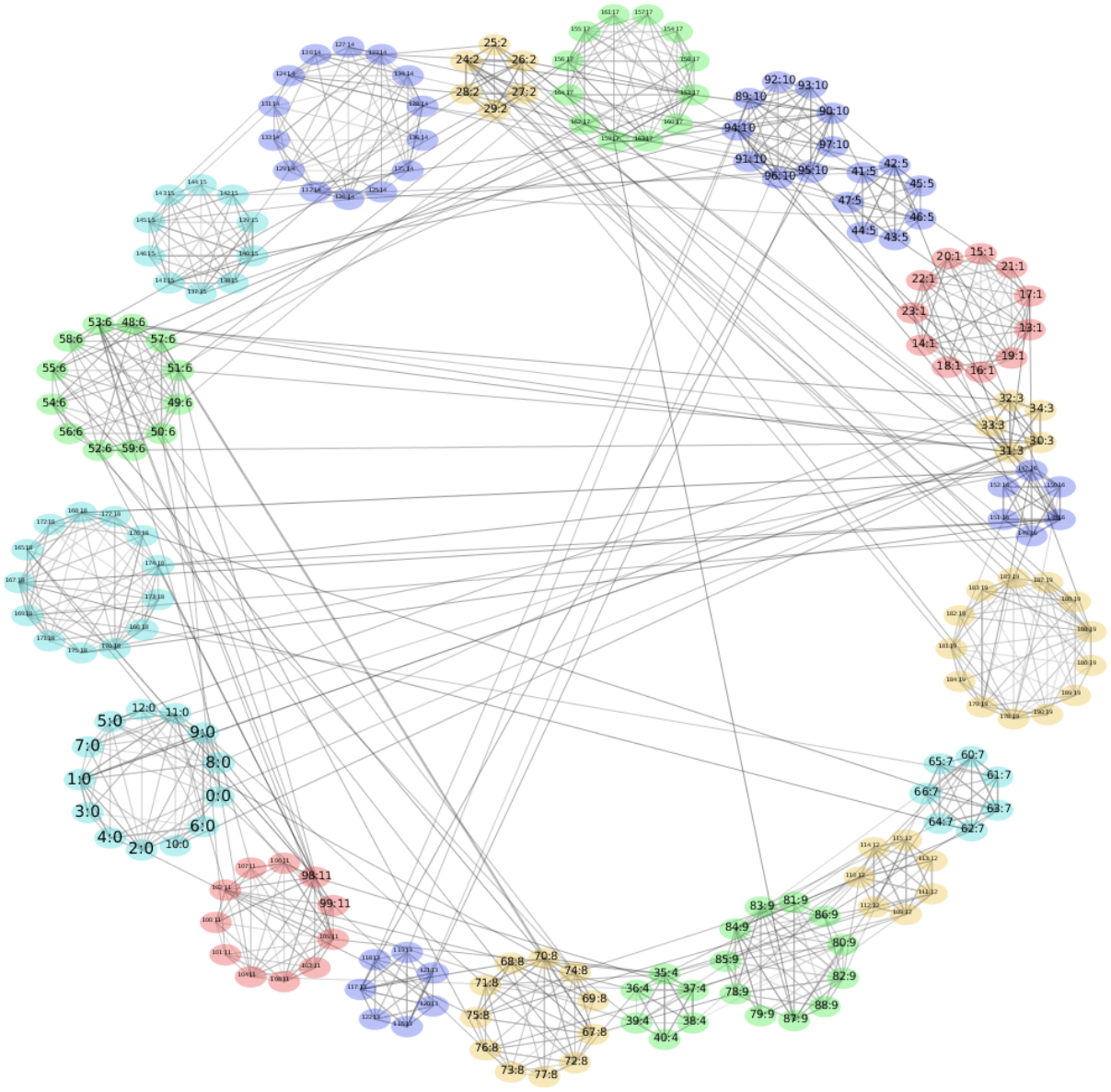
Tyto algoritmy lze použít pro kreslení celého grafu, ale také pro uspořádání menších skupin vrcholů v rámci kreslení velkého grafu, jako například pro grafy s více komponentami, shluky genů v genově interaktivním grafu, nebo podskupiny v sociálních sítích. Výhodou kruhového uspořádání v některých použitích, jako jsou bioinformatika nebo vizualizace sociálních sítí, je jeho neutralita: umístěním všech vrcholů stejně daleko od sebe a od středu plátna, žádný z těchto bodů nemá výhodnější pozici.

Hrany mohou být zobrazeny jako tětivy kružnice, kruhové oblouky, nebo jiný typ křivky. Vizuální rozdíl mezi vnitřní a vnější stranou kružnice může být použit k rozdělení dvou odlišných způsobů kreslení hran. Například algoritmus kruhového kreslení od Gansner and Koren (2007) používá ohýbání hran v kruhu spolu s několika hranami, které nejsou ohýbány a jsou kresleny mimo kruh.

Pro kruhové rozvržení pravidelných grafů, s hranami kreslenými uvnitř a vně jako kruhové oblouky, úhel dopadu jednoho z těchto oblouků je stejný na obou koncích oblouku. Je to vlastnost usnadňující optimalizaci úhlového rozlišení kresleného obrazu. [4]

Někteří autoři studovali problém hledání permutace vrcholů kruhového rozmístění, který minimalizuje počet křížení hran, když jsou všechny hrany kresleny uvnitř kruhu. Tento počet přechodů je nula pouze pro outerplanar grafy. Pro ostatní grafy může být optimalizován nebo redukován odděleně pro každou dvouspojenou komponentu grafu před sloučením, protože tyto komponenty mohou být vykresleny tak, že na sebe vzájemně nepůsobí.

Obecně platí, že snižováním počtu křížení hran je NP-kompletní, ale může se blížit k časové složitosti O(log2 n), kde n je počet vrcholů. Heuristické metody na snižování křížení hran byly rovněž založeny například na opatrném vkládání vrcholů a na lokální optimalizaci. Kruhové rozvržení grafu může být také použito pro maximalizování křížení hran v grafu. Zejména výběr náhodné permutace pro vrcholy způsobuje, že každé možné křížení hran nastane s pravděpodobností 1:3, takže očekávaný počet křížení hran je v rámci trojnásobného maximálního počtu křížení mezi všemi možnými rozvrženími. [15]



Obrázek 3: Circular Layout. [19]

2.3 Edge Bundling

První formální použití edge bundlingu je pravděpodobně v článku o HEB algoritmu od Dannyho Holdena, který byl publikován v říjnu 2006. Edge bundling je metoda, která se používá ke zpřehlednění grafů. Tyto algoritmy se používají pro data v mnoha odvětvích např. sítě, sledování dopravy, migrace, reprezentace uživatelů a jejich spojení ve virtuálním světě atp. Edge bundling algoritmy mohou být iplementovány jak ve 2D, tak i ve 3D. Když má graf vysoký počet hran, stává se, že uživatel z grafu nic nevyčte, reprezentace grafu je jen spletí stovek, tisíců hran, které jsou kresleny přes sebe a zahltí plátno na které je graf kreslen. Je vytvořeno mnoho technik na ohýbání hran v grafu, které mají za účel snížit nepořádek a zvýšit přehlednost výsledného grafu. Edge bundling algoritmy jsou navrženy pro grafy obsahující hierarchické struktury a vztahy popisující sousednosti hran. [8]

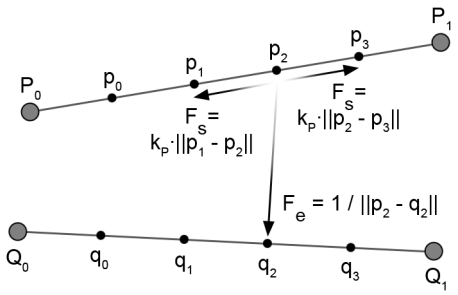
Používané Edge Bundling algoritmy:

- Hierarchical Edge Bundles (HEB)
- Multilevel agglomerative edge bundling (MINGE)
- Force-directed edge bundling (FDEB)
- Winding roads (WR)
- Graph Bundling by Kernel Density Estimation (KDEEB)
- Geometry-Based Edge Bundling (GBEB)
- Skeleton-Based Edge Bundling (SBEB)
- 3D Shift Edge Bundling
- 3D Density histograms for criteria-driven Edge bundling (3DHEB)

2.3.1 Force-directed edge bundling

Jednoduchý způsob jak sloučit hrany dohromady v obecném grafu, je nejprve vytvořit hierarchii a pak použít HEB algoritmus k sloučení hran. Nicméně vytvoření vhodné HEB hierarchie pro obecný graf není triviální. Svazky vzniklé pomocí hierarchie by měly věrně odrážet vysokou úroveň vzorů hran. Ale není zřejmé, jaký způsob hierarchického svazování nebo kostry grafu použít, aby to bylo pro daný graf vhodné. FDEB algoritmus má tu výhodu, že je snadno pochopitelný, protože vychází z jednoduchého modelu fyziky. Algoritmus může být implementován na několika řádcích kódu a snadno může být rozšířený o další požadavky na funkci algoritmu.

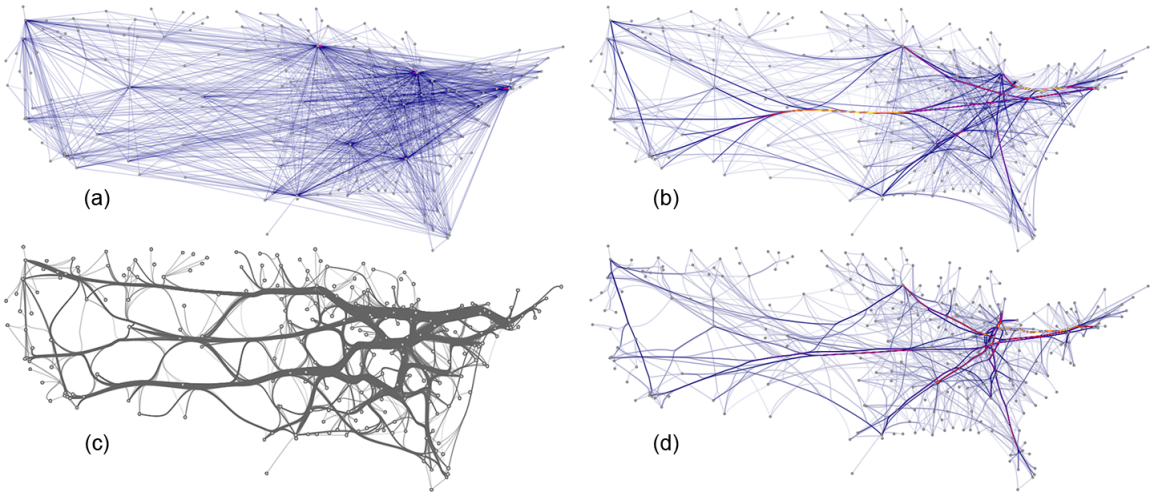
Vstupem FDEB algoritmu je lineární uzel-spoj diagram obecného grafu. Tento uzel-spoj diagram může být generovaný za pomocí jakékoliv technologie na vytaváření grafů. V případe, že vstupní graf představuje nějaké geografické informace, například migrace, letadlový provoz atp. jsou pozice bodů grafu dány podle geografických souřadnic. Chcete-li přímkám spojující vrcholy povolit měnit tvar v rámci svazování hran, je třeba je rozdělit na menší úseky. Na obrázku č.4 jsou dvě hrany P a Q, které jsou rozděleny pomocí 4 dělících bodů na jednu hranu, koncové pozice hran (P_0, P_1, Q_0, Q_1) zůstanou stejné.



Obrázek 4: Rozdělení hran. [9]

Pro každou hranu je přitažlivá síla F_s použita pro každý pár po sobě jdoucích bodů na hraně, které vznikly dělením hrany. Přitažlivá síla se rovná nule, pokud koncové body hrany mají od sebe nulovou vzdálenost. Konstanta globální tuhosti K je určena ke stanovení tuhosti ohýbaní hran v grafu. Vzhledem k tomu, že hrany mají různou délku a jsou rozděleny do segmentů, je konstanta Kp mezi body segmentů spočítána pro každý segment hrany P. k_p je stejná pro každý segment hrany P a vypočítá se $k_p = K/|P|(počet segmentů)$, kde |P| je celková délka hrany P.[9]

Elektrostatická přitažlivá síla F_e je použita mezi všemi páry podle dělení obou hran. Na obrázku č.4 jsou to páry $p_0 - q_0, p_1 - q_1, p_2 - q_2, p_3 - q_3$. Použitím inverzního čtvercového modelu oproti inverznímu lineárnímu modelu dostaneme lepší výsledky a více lokalizované ohýbaní hran. F_e by mohla být také vypočítána pro každou kombinaci párových bodů (p,q) kde $p \in P$ a $q \in Q$. To by zvýšilo výpočetní složitost z O(N) na $O(N^2)$ mezi páry bodů na hranách, kde N je počet bodů na hraně. Oba postupy jsou vizuálně málo rozlišné, tak je vhodné použít méně časově náročný postup. Během každého výpočtu kroku iterativní simulace se spočítá kombinace sil působící na body rozdělených částí hran. Pozice každého takového bodu je aktualizována malým posunem ve směru spočítané výsledné síly, která působí na tento bod. Pro bod segmentu p_i z hrany P, je kombinovaná síla F_{pi} působící na tento bod kombinací dvou sousedních přitažlivých sil F_s daných z bodů p_{i-1} a p_{i+1} a součtem všech elektrostatických sil F_e. Výpočet F_{p_i} je definován rovnicí: $F_{p_i} = k_p(||p_{i-1} - p_i|| + ||p_i - p_{i+1}||) + \sum_{Q \in E} \frac{1}{||p_i - q_i||}$, kde k_p je tuhost pro každý segment hrany P, E je množina všech vzájemně na sebe působících hran s vyjímkou hrany P. [14]

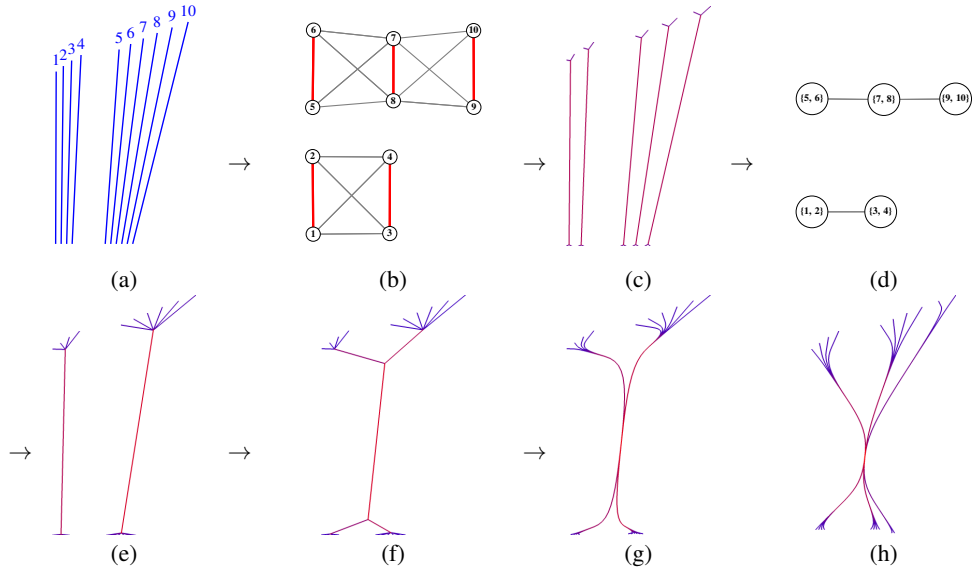


Obrázek 5: US Migrace. [14]
US migrace graf (1715 vrcholů, 9780 hran): (a) originální graf, (b) FDEB s inverzně-linearním modelem, (c) GBEB, (d) FDEB s inverzně-čtvercovým modelem

2.3.2 Multilevel Agglomerative Edge Bundling

Vstupem algoritmu je obecný graf G, který je dán vrcholy V a hranami E, popis algoritmu se bude týkat 2D kreslení, algoritmus je možno rozšířit pro práci v 3D prostředí.
1. Identifikace hran, které mají podobné počáteční a koncové body.
2. Sloučení podobných hran.
3. Kontrola, zda se nemůže další hrana připojit k už existujícímu svazku, nebo je potřeba vytvořit nový svazek.
4. Opakovat tento proces.

Nejprve musíme identifikovat podobné hrany, Holten a van Wijk představili 4 opatření s kompatibilitou hran v grafu. Pro každou hranu je třeba kontrolovat podobnost s ostatními hranami z toho vzniká E^2 operací. S takovou časovou složitostí je nereálné použití u velkých grafových struktur. Řešením je zbavení se kvadratické složitosti tak, že použijeme jednoduché opatření kompatibility, kde každá hrana představuje jeden bod v 4-rozměrném prostoru, hranová podobnost je dána metricky v tomto prostoru. Připoužitím tohoto postupu můžeme efektivněji vytvořit hranově proximitní graf. Hrany, které jsou dány v proximitním grafu jako sousedé k dané hraně, jsou kontrolovány k možnému sloučení. Hrany, které nejsou bezprostřední sousedé mohou být i přesto svázány z důvodu víceúrovňového přístupu, který je popsán v sekci 3.2.2.



Obrázek 6: Vyobrazení průchodu MINGLE algoritmem. [11]
(a) Originální hrany, Ink = 35.82. (b) Vytvoření proximitního grafu (k=3). (c) Po jednom aglomerativním svazováním. Ink = 19.31. (d)Vytvoření hrubého proximitního grafu. (e) Po víceúrovňovém aglomerativním svazováním. Ink = 12.00. (f) Po rekurzivním použití svazování. Ink = 10.94. (g) Vyobrazeno za použití křivek. (h) Se spojovacím úhlem <= 40° a renderování za pomocí křivek. Ink = 15.24.

Hranově proximitní graf

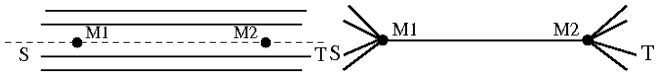
Každý vrchol u má pozici v 2D označenou jako x_u. Každá hrana (u, v)je prezentována jako 4D vektor (x_u, x_v). Dvě hrany jsou blízko, pokud je jejich euklidovská vzdálenost ve 4D prostoru malá. Rozklad prostoru může být např. za použití kd-tree, může být konstruován s využitím všech |E| 4D vektorů v čase $|E|\log(|E|)$.Tato datová struktura umožnuje najít k-nejbližší ($k << |E|$) sousedy v čase $k\log(|E|)$ za jednu hranu. Takže konstrukce hranově proximitního grafu Γ trvá $k|E|\log(|E|)$. Pořadí bodů ve 4D vektoru (x_u, x_v) ovlivňuje vzdálenost ve výsledku. Vyhnout se tomuto problému dá tak, že se vkládají obě souřadnice (x_u, x_v) a (x_v, x_u) do struktury a to bez ovlivnění časové složitosti. Vrcholy grafu Γ nejsou stejné s vrcholy grafu G, vrcholy grafu Γ jsou hrany grafu G.

Hranově proximitní graf Γ při konstrukci nemusí zahrnout všechny nejbližší sousedy, a to z důvodu omezeného počtu k, čímž je dán maximální počet sousedů. Také to může být z důvodu použití měření vzdálenosti v euklidovském 4D prostoru, namísto použití více detailního výběru, kde se porovnává délka, viditelnost, úhel a pozice. To nezpůsobuje významné problémy, protože po každém svázání hran se měří, zda svázání přineslo úsporu inkoustu. Navíc proces víceúrovňového svazování, svazuje nejen sousedy, ale i sousedy sousedů. A na konec rekurzivní způsob zvyšuje možnost spojení hran, které doposud nebyly spojeny. [22]

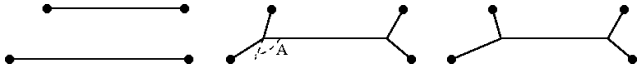
Aglomerativní svazování

Po zkonstruování grafu Γ, se svazují hrany. Pro každého souseda v vrcholu u v Γ, spočítáme úsporu inkoustu, která vznikde pokud sloučíme hrany v a u. Poté se vybere jeden ze sousedů, se kterým je největší úspora inkoustu. Někdy tento soused může být sloučený s jinými hranami, tudíž v reprezentuje celý svazek. Pokud $e(u)$ označuje hranu nebo hrany reprezentované vrcholem u v Γ,a pokud ink($e(u)$) označuje inkoust potřebný k nakreslení hrany nebo svazku hran reprezentující u, a $e(u) \bigcup e(v)$ hrana svazku tvořená sloučením hran u v. Pak množství ušetřeného inkoustu je dáno jako $ink(e(u)) + ink(e(v)) - ink(e(u)) \bigcup ink(e(v))$ Spočítáme přibližné množství inkoustu potřebné k nakreslení svazku za použití 1D optimalizační procedury od Gansner a Koren. $e(u) \bigcup e(v) = \{e_1 = (e_1^S, e_1^T),\ e_2 = (e_2^S, e_2^T),.....,\ e_k = (e_k^S, e_k^T)\}$. Předpokládejme, že konce hran jsou uspořádány tak, že tvoří dvě stejné velikostní sady, zdrojová sada S = $\{e_1^S, e_2^S, ..., e_k^S\}$ a cílová sada T = $\{e_1^T, e_2^T, ..., e_k^T\}$. Cílem postupu optimalizace inkoustu je najít dva styčné body M_1 a M_2, takže tyto hrany nechávají vrcholy v začátku S, odkud pokračuji do bodu M_1, pak do bodu M_2 odkud se rozdělí do původních cílových bodů T (Obrázek č.7). Celkový inkoust, který je potřeba k nakreslení těchto hran:
$f(\mathrm{S},\mathrm{T},M_1,M_2) = \sum_{x \in S} ||x - M_1|| + ||M_1 - M_2|| + \sum_{x \in T} ||M_2 - x||$.
Styčné body M_1 a M_2 jsou vybrány k minimalizováni celkového inkoustu:
$ink(e(u)) \bigcup ink(e(v)) = \min_{M1,M2} f(\mathrm{S},\mathrm{T},M_1,M_2)$. [11]

První centrální body z S a T jsou spočítány. Potom je potřeba je minimalizovat za pomoci procedury známé jako golden section search a najít body M_1 a M_2 na čáře spojující centrální body(Obrázek č.7)



Obrázek 7: Ilustrace minimalizování inkoustu. [11]



Obrázek 8: Úhel zakřivení hran. [11]
Levý: Hrany se vzdálenými body. Prostřední: Sloučení hran s velkým úhlem zakřivení. Pravý: Omezení úhlu zakřivení.

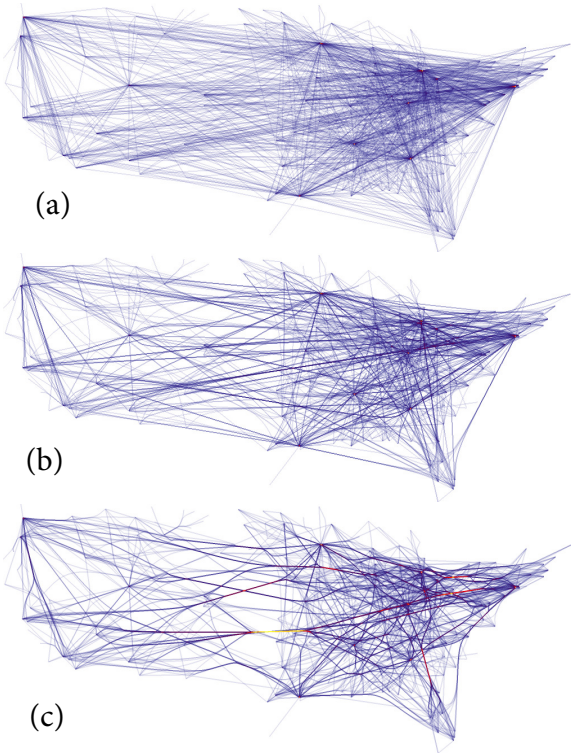
Někdy, když jsou daleko od sebe koncové body hran, potom sloučení těchto hran má za následek velký úhel zakřivení, a odpovídající křivky budou mít velké zakřivení. Lze to řešit dvěma způsoby. Za prvé, aby se docílilo maximálního uživatelem specifikovaného úhlu, omezit oba styčné body tak, aby se nepřekročil maximální daný úhel zakřivení. Za druhé, kompromis mezi úsporou inkoustu a menším úhlem zakřivení, výběr bodu M_1 a M_2 k minimalizování: $f(\mathrm{S},\mathrm{T},M_1,M_2)\left(1+\frac{\cos(A_{max})}{p}\right)$, namísto (1), s parameterm $p>1$, s vyjímnkou k maximálnímu úhlu zakřivení. A_{max} je maximální úhel zakřivení dán konkrétními styčnými body. Vyšší hodnota p zajistí vyšší úsporu inkoustu, přičemž nižší hodnota zajistí, aby úhly nebyly ostré.

Víceúrovňové aglomerativní svazování.

Poté, co jsou hrany sloučeny aglomerativním spojujícím procesem, zdrsníme hranově proximitní graf tím, že spojíme uzly (které představují okraje původního grafu), které jsou sloučeny. Nyní každý uzel z hrubého grafu může představovat svazek hran (Obrázek č.6 (d)). Pak opakujeme proces svazování hran, jak je popsané v předchozí části. Tento víceúrovňový proces je ukončen, když po průchodu nevznikla úspora inkoustu oproti poslednímu průchodu. Víceúrovňový proces umožňuje svázat i hrany, které mohou být v originálním proximitním grafu tak daleko, že by neměly šanci na svázání, toto platí pouze v případě, že výsledkem je úspora inkoustu. Obrázek č.6 (c),(e) ukazuje výsledek po první a po druhé aglomeraci. První průchod snižuje množství inkoustu z 35.82 na 19.31. Druhá úroveň redukuje hodnotu dále na 12.00. Žádná další úspora inkoustu nemůže být dosažena dalším průchodem algoritmu. [11], [10]

Rekurze

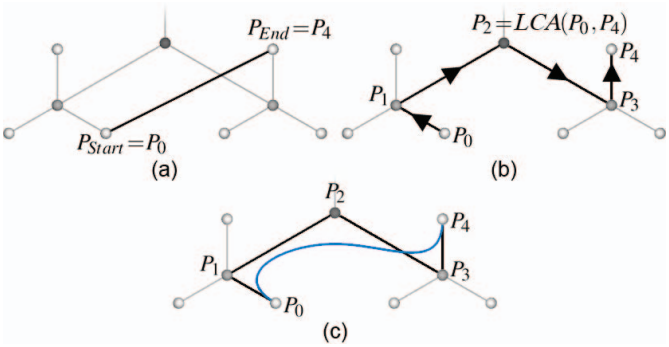
Po jednom průchodu svazovacím algoritmem, je každá hrana reprezentována křivkou s maximálně třemi segmenty. Obrázek č.7 (b) ukazuje výsledek na konci víceúrovňového procesu, při aplikaci na *airlines* graf. První průchod algoritmem redukuje nepřehlednost grafu oproti původnímu, ale stále zbývají hrany, které mohou být dále sloučeny. Proto je vhodné použití rekurzivního volání víceúrovňového procesu, za účelem sloučení svazků s hranami nebo dalšímy svazky. Obrázek č.6 (f) znázorňuje výsledek rekurze. Rekurze končí, pokud další průchod nepřinese úsporu inkoustu, v tom okamžiku je každá hrana reprezentována křivkou. Obrázek č.7 (c) ukazuje výsledek po rekurzivním průchodu.



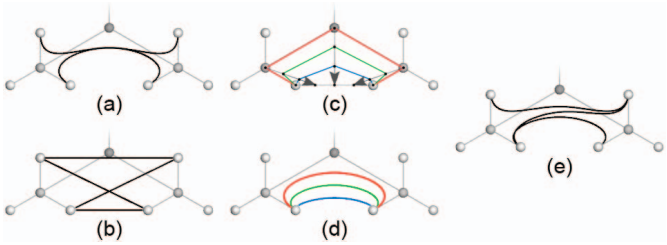
Obrázek 9: US Airlines. [11]
(a) Originální graf *US Airlines*. (b) Hrany ohýbány bez rekurze. (c) Hrany ohýbány s rekurzí.

2.3.3 Hierarchical Edge Bundles

Hierarchické svazování hran funguje na principu svazování hran podle hierarchického rozdělení - vhodné je použití stromových vizualizačních technik. Na obrázku č.10 je ukázáno, jak se vytváří křivka mezi dvěma body pomocí balloon tree layoutu. Tento přístup používa cestu hiearchií mezi dvěma vrcholy, které mají být spojeny a podle toho ohýbá křivku. Výsledná křivka je použita jako reprezentace vztahu mezi body. Kontrolní body P_i, které vytváří řídící polygon jsou podél cesty skrz hierarchii od P_{Start} skrz LCA(P_{Start}, P_{End}) do P_{End}, kde LCA(P_{Start}, P_{End}) je nejmenší společný předek bodů P_{Start} a P_{End}.



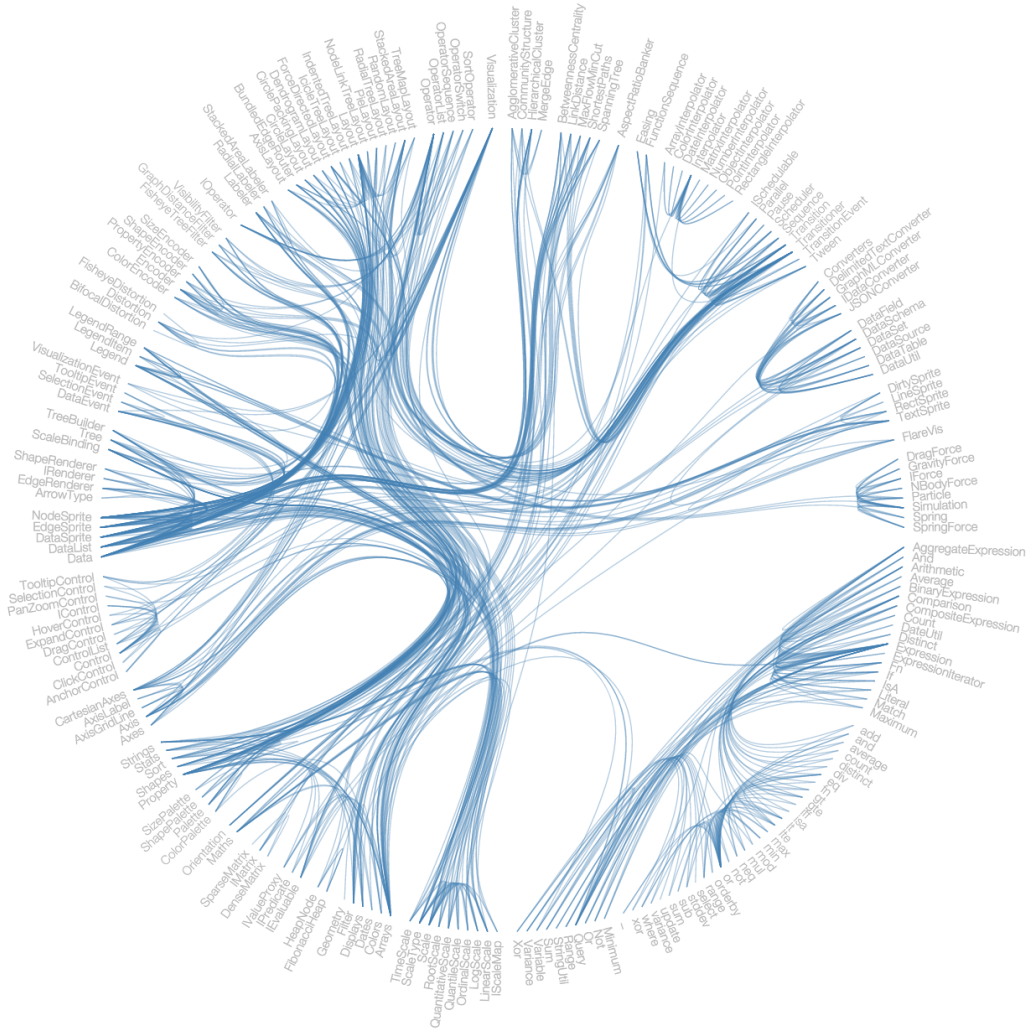
Obrázek 10: HEB boundling. [13]
(a) Přímka mezi P_0 a P_4. (b) Cesta skrz strom od bodu P_0 do P_4. (c) Křivka znázorňující spojení mezi body P_0 a P_4, která je ohýbána podle průchodu skrz strom.



Obrázek 11: Řešení problému dvojznačnosti. [13]
Výsledek na obrázku (a) může obsahovat každou hranu, která je znázorněna na obrázku (b).(c) a (d) znázornují rozdíl při jiné hodnotě β parametru (červená = 1, zelená = $\frac{2}{3}$, modrá = $\frac{1}{3}$). Poslední obrázek (e) ukazuje vyřešení nejednoznačnosti, kde parametr β je nastaven na hodnotu 0.8.

Pokud je tento přístup použit přímo ke slučování sousedních hran, může nastat problém dvojznačnosti, což je znázorněno na obrázku č. 11. Tyto problémy mohou být zredukovány snížením svazovací síly. Svazovací síla je řízena parametrem β, $\beta \in [0, 1]$, β učinně kontroluje sílu ohybu křivky. Obrázek č. 11 (d) zobrazuje vliv použití parametru, (e) zobrazuje řešení problému dvojznačnosti pomocí parametru β. [6]

Graf je kreslen po částech za pomocí kubických B-spline křivek, za pomocí těchto křivek je lepší lokální kontrola, což je nezbytné pro výrobu soudružných a rozdílných svazků, při zachování stupně a s nízkou výpočetní složitostí. Jednotný vektor uzlu řádu 4 (stupeň = 3), který má stejné hodnoty na každém konci je použit k interpolování začátku a konce kubické B-spline. Stupeň se automaticky snižuje na hodnotu 2 nebo 1, je nutné aby stupeň byl menší než číslo kontrolních bodů. [13]



Obrázek 12: Výsledný graf po použití HEB algoritmu. [21]

3 Implementace aplikace

3.1 Použité technologie

Aplikace je vivinuta v prostředí Microsoft Visual Studia 2015, je použito jen jako editor a pro zvýraznění syntaxe JavaScript, HTML, CSS a JSON kódu. Aplikace je spouštěna a testována v prohlížečích Google Chrome(verze 49.0) a Mozilla Firefox(vezre 45.0).
HLML5 a CSS3 je využito pro grafické zpracování aplikace, které umožňuje nastavení parametrů uživatelem, pro změnu vizualizace, vstupních dat a výpočtu.
Datový formát JSON je využit jako vstupní data pro výpočet, ze kterého se při spuštění aplikace načtou data grafu, na které má být aplikován algoritmus.
V programovacím jazyku JavaScript je napsána převážná část aplikace - načítání dat, výpočetní algoritmy, komunikace s grafickým rozhraním a knihovny usnadňující práci s daty.

První JavaScriptovou knihovnou, která je využita v rámci aplikace je knihovna jQuery[1]. Z této knihovny je použita pouze metoda ajax pro synchroní načítaní dat z JSONu, která jsou uložena do proměnné a následně do grafové struktury.
Další použitou knihovnou je PhiloGL[2], která je použita k animaci výsledného grafu v reálném čase. Animace je způsobena postupnou změnou parametru Delta, kde se v každém průchodu smyčkou inkrementuje tento parametr a poté je volána renderovací metoda.
Knihovna kdTree[3] je použita k získání nějbližších k-sousedů, kteří jsou pak použiti k vytvoření proximitního grafu, podle kterého jsou svazovány hrany.
Knihovna FruchterManReingoldLayout[4] je použita k výpočtu force-directed layoutu.
K výpočtu MINGLE algoritmu a vizualizaci svázaných grafů, jsou použity metody z projektu mingle[5].
V knihovně Transform je uchovávána transformační matice Canvasu, tato knihovna je použita i na výpočet transformací(translate, scale, rotate). Knihovna byla použita z důvodu jednoduché práce s maticí, knihovna byla rozšířena o další proměnné určené k výpočtu správných souřadnic myši, použité u zoomovaní a posunu grafu.
Poslední použitou JavaScript knihovnou je rangeSlider[6] poskytující metody ke CSS knihovně rangeSlider, která zajištuje vzhled posuvníků v grafickém rozhraní.

[1] jQuery
[2] PhiloGL
[3] kdTree
[4] Janoušek, Jan , Diplomová práce
[5] philogb/mingle
[6] rangeslider.js

3.2 Vstupní data

Vstupní data jsou uložena na disku v datovém formátu JSON. Data jsou načítána při startu aplikace, ale mohou být načtena i za běhu aplikace. Na obrázku č.13 je ukázka minimálních vstupních dat. ID hrany a coords je nutné uvést u každé hrany, aby byla zaručena správná funkčnost. Další parametry, které mohou být u hran uvedeny jsou: name, weight a color(RGB). Vstupní data jsou reprezentována pouze hranami grafu, bod začátku hrany a konec se použijí jako reprezentace vrcholů grafu. Pokud je jedna souřadnice dvou nebo více hran stejná bude to ve výsledném grafu reprezentováno jako spojení těchto hran v daném bodu grafu. Jako další vstupní data jsou načítána jména a geo-souřadnice měst, která jsou přiřazována vrcholům v grafu, struktura tohoto souboru je také JSON.

```
[
  {
    "id": 1,
    "data": {
      "coords": [25, 0, 60, 20 ]
    }
  },
  {
    "id": 2,
    "data": {
      "coords": [ 10, 10, 25, 0 ]
    }
  }
]
```

Výpis 1: Příklad vstupních dat - JSON.

3.3 Výběr algoritmů

Na rozmístění bodů byl vybrán Force-Directed layout, a to z důvodu jednoduchosti výpočtu a implementace algoritmu, dalším důvodem byl výsledek tohoto algoritmu z pohledu vizuálně pěkného rozmístění bodů grafu po plátnu canvasu. Jako vstupní data jsou používany grafy velikosti řádově stovek až několika tisíc hran, časová složitost tohoto algoritmu je $O(n^3)$, při použití na grafech této velikosti to není z časového hlediska velký problém.

Jako algoritmus na svazování hran byl vybrán Multilevel Agglomerative Edge Bundling(MINGLE), a to z důvodů vysoké rychlosti výpočtu, nízkých vstupních požadavků na grafovou strukturu, rekurzivního průběhu algoritmu a také pro vizuálně pěkný vzhled grafu po sloučení hran.

3.4 Třídy

SipleGraph je třída uchovávající jednoduchý graf, který je načten z JSONu. Graf v této třídě je reprezentován dvěmi poli(nodes, edges), v poli nodes jsou uchovávány informace o vrcholech, v poli edges jsou uchovávany informace o hranách a pro každou hranu dva ukazatele na instance vrcholů, které reprezentují, mezi kterými vrcholy je daná hrana. Tato třída obsahuje metodu centerGraph, která zajišťuje, aby střed grafu byl ve středu plátna a graf byl velikostně přes celé plátno, čehož je dosaženo pomocí transformací. Dále v této třítě jsou metody na vykreslování jednoduchého grafu, metody na vykreslování a správu informací k vrcholům, metody na práci s poli a jejich obsahem.

ProximityGraph je třída, ve které se vytváří proximitní graf, který je potřeba pro MINGLE algoritmus. Reprezentace grafu ve třídě je stejná jako u třídy SimpleGraph, jen instance hran mají navíc pole(adjacencies), ve kterém je uchováván seznam sousedních hran. Třída obsahuje metody na nalezení sousedních hran za pomocí knihovny kdTree a metody na práci s poli a jejich obsahem.

BundledGraph je třída uchovávající graf po aplikaci MINGLE algoritmu. Graf je reprezentován opět dvěmi poli(nodes, edges), kde pole nodes obsahuje stejné informace jako u třídy SimpleGraph, pole edges uchovává hrany, které obsahují informace o hraně, inkoustu, sousedních hranách, hranách ze kterých jsou složeny a další informace potřebné pro správný výpočet bundlingu. Třída obsahuje metody převážně pro renderování grafu podle parametrů, metody na správu grafu a podpůrné matematické metody a funkce.

Mingle je třída, v níž je implementován MINGLE algoritmus, data pro algoritmus jsou použita ze třídy BundledGraph. Každý průchod rekurzivní částí algoritmu a její výsledek je uložen do nové instance třídy BundledGraph z důvodu možnosti vizualizace každého průchodu.

Main je třída, v níž je řešena správa celé aplikace. Jsou zde uloženy instance všech předešlých 4 tříd, a parametry ovlivňující výpočet algoritmů a vizualizaci dat na plátno. V této třídě jsou metody na načítání dat ze JSON souborů, aktualizaci plátna a použití výpočetních algoritmů, metody řešící vstupní data z grafického rozhraní.

3.5 Spuštění aplikace

Aplikace je spustitelná v prohlížečích Google Chrome a Mozilla Firefox, verze těchto prohlížečů jsou uvedené výše. Před spuštěním aplikace v prohlížečích Google Chrome je nutné spustit prohlížeč s parametry --allow-file-access-from-files. Tento prohlížeč má z bezpečnostních důvodů jako výchozí nastavení zákaz načítání lokálních souborů. Spuštění prohlížeče lze provést například přes konzoli, příkaz do konzole by mohl vypadat následovně:
"chrome.exe --allow-file-access-from-files". Po spuštění prohlížeče s parametry je možné spustit apliaci(Edge Bundling.html). U prohlížeče Mozilla Firefox tento problém není, tudíž je možné spustit aplikaci jednoduše bez zadávání parametrů při spouštění prohlížeče. Po spuštění aplikace se ihned vypočte a zobrazí graf s výchozími parametry.

3.6 Ovládání aplikace

Vstupní parametry:

Parametr	**Min**	**Max**	**Popis**
Delta	0%	100%	Procentuální nastavení zakřivení hran.
Curviness	0%	100%	Nastavuje zakřivení hran kolem svazovacího bodu.
Angle	0.1	4	Ovlivňuje maximální úhel zakřivení směrem do svazku.
Neighbors	2	50	Maximalní počet hledaných sousedů.
Margin	0	25	Rozložení svazků na jednotlivé hrany.
Alpha	0	1	Nastavení průhlednosti.
Layout iterations	0	1000	Počet průchodu layout algoritmem.
Animation	1	5	Délka trvání animace.

Tabulka 1: Vstupní parametry.

Solid color - Rozsah(RGB 0 - 255), výběr barvy pro solid color.
Line type - Možnosti(Line, Bezier, Quadratic)
Color - Možnosti(Solid, Cluster, Edge), po výběru jedné z možností se překreslí graf.
Dataset - Možnosti(Star graph, Connected graph, Planar graph, Componnent graph), po výběru jedné z možností následuje výpočet(včetně layoutu).
Recursion - Po průchodu algoritmem se zde nabídnou jednotlivé průchody rekurzivní částí algoritmu.

Funkce aplikace:

Počítání inkoustu - Zobrazuje procentuální pokrytí plátna barvou, údaj se aktualizuje při každém vykreslování.

Zoom grafu - Pomocí kolečka myši je možné zoomovat, kolečkem nahoru pro přiblížení a dolů pro oddálení. Zoomuje se vždy směrem k pozici myši.

Posun grafu - Graf lze posunout tahem myši při stlačeném levém tlačítku, při stisknutí tlačítka myši musí mít pozici v plátně, ale mimo vrcholy grafu.

Přesouvání vrcholů - Vrcholy lze posunout tahem myši při stlačeném levém tlačítku, při stisknutí tlačítka myši musí mít pozici vrcholu. Graf se bude zobrazovat v průběhu tahu, hrany budou reprezentovány přímkami s nastavenou solid barvou, po uvolnění levého tlačítka v rámci plátna se na upravený graf aplikuje MINGLE algoritmus a zobrazí se podle nastavených parametrů.

Zobrazování informací o vrcholech - Informace o vrcholu lze zobrazit tak, že pozice myši bude na vrcholu, kde se poté zobrazí informace. Po přesunutí myši od bodu vrcholu po nějakém čase bublina s informací zmizí.

Přizpůsobení velikosti canvasu - Velikost plátna je dána podle velikosti okna prohlížeče. Layout algoritmus pracuje s velikostí plátna při rozmísťování bodů, takže při jiné velikosti okna prohlížeče může stejný graf vypadat jinak.

4 Experimenty s nastavením aplikace

Parametr	Hodnota
Delta	100%
Curviness	100%
Angle	3
Neighbors	10
Margin	0
Alpha	0.7
Layout iterations	200
Animation	3
Solid color	Orange
Line type	Quadratic
Color type	Cluster
Dataset	Star graph
Recursion	2. Recursion

Tabulka 2: Výchozí hodnoty (obrázek č.13).

Obrázek 13: Vizualizace podle výchozích hodnot.

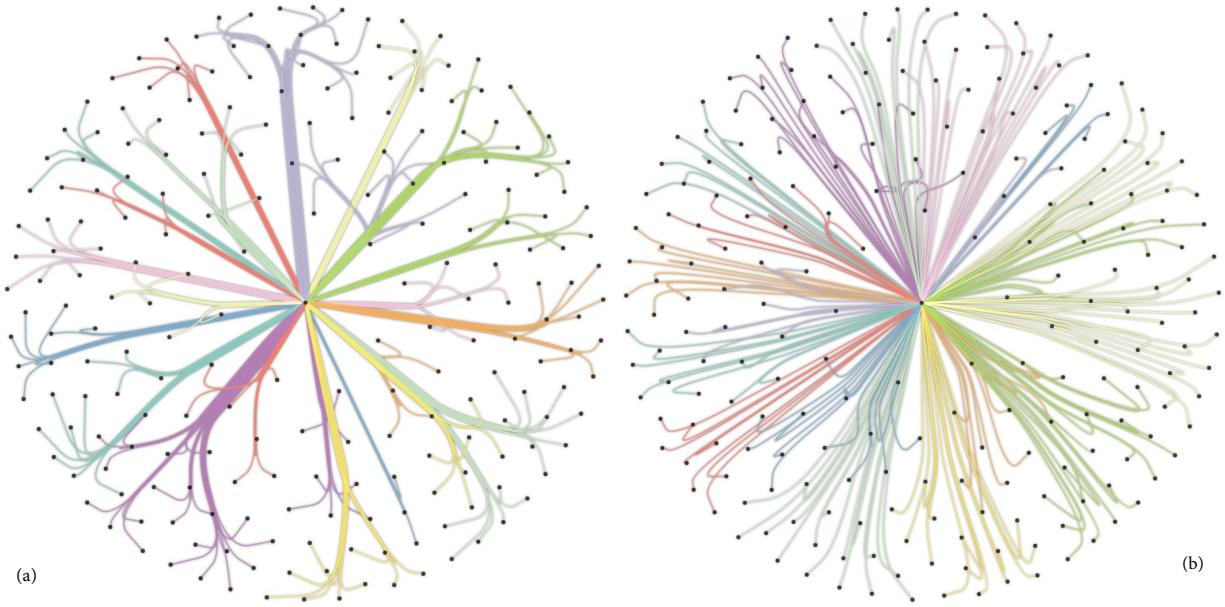
4.1 Vstupní parametry

4.1.1 Delta

Parametr	**Hodnota**	
	Výchozí	Upravená
Delta	100%	50%

Tabulka 3: Změna parametru delta vůči výchozímu nastavení.

Tento parametr má vliv pouze na vykreslování grafu, ostatní nastavení jsou shodná s výchozím jako u obrázku č.13, dle tabulky č.2. Pomocí delty se nastavuje interpolace mezi dvěma body křivky, pokud je delta 0, hrany jsou reprezentovány jako přímky, při inkrementování delty se hrany lineárně zakřivují.



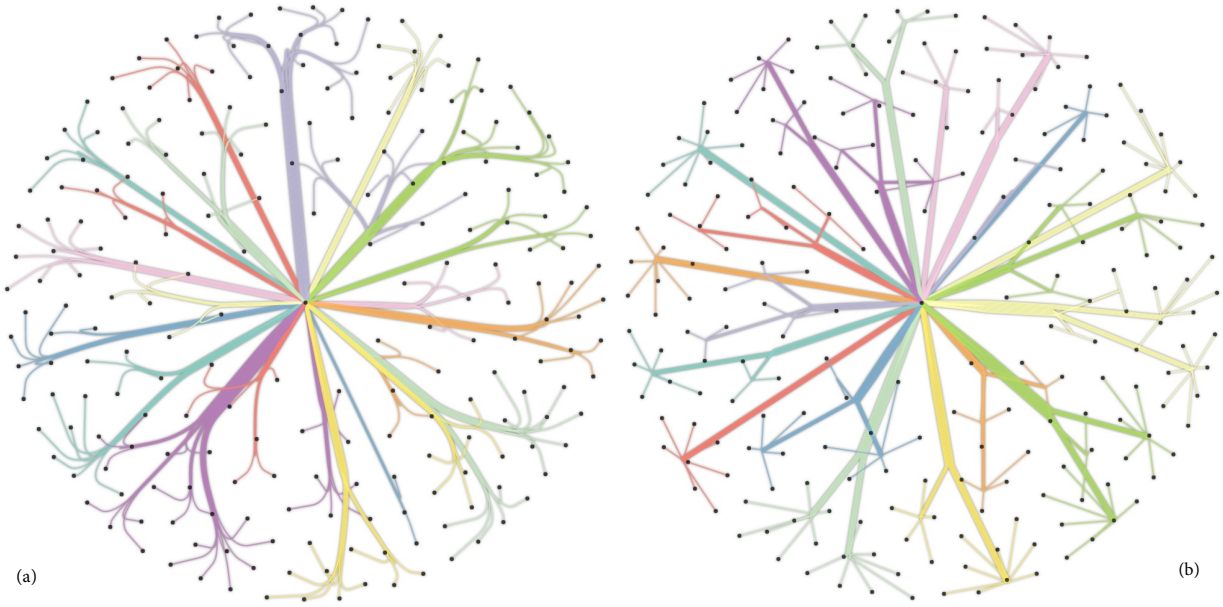
Obrázek 14: (a) Výchozí. (b) Upravený.

4.1.2 Curviness

Parametr	**Hodnota**	
	Výchozí	Upravená
Curviness	100%	0%

Tabulka 4: Změna parametru curviness vůči výchozímu nastavení.

Tento parametr má vliv pouze na vykreslování grafu, ostatní nastavení jsou shodná s výchozím jako u obrázku č.13, dle tabulky č.2. Pomocí parametru curviness se nastavuje zakřivení hran u středového bodu svazku, kde se slučují hrany do svazku. V případě nízké hodnoty jako na obrázku č.15 (B) je přechod ostrý, při vyšších hodnotách je přechod plynulejší a nejde vidět zlom hrany v bodu kde se hrany spojují.



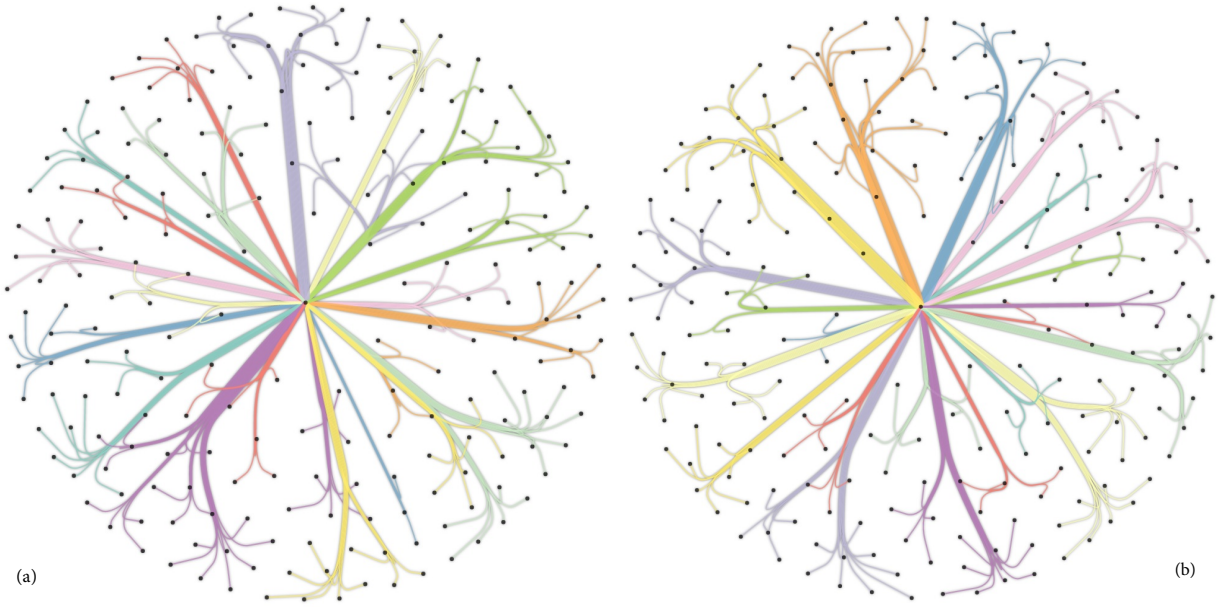
Obrázek 15: (a) Výchozí. (b) Upravený.

4.1.3 Angle

Parametr	Hodnota	
	Výchozí	Upravená
Angle	3	5

Tabulka 5: Změna parametru angle vůči výchozímu nastavení.

Tento parametr má vliv pouze na výpočet grafu, ostatní nastavení jsou shodná s výchozím jako u obrázku č.13, dle tabulky č.2. Pomocí parametru angle se ovlivňuje maximální úhel zakřivení hrany, který má vliv na rozhodování, zda se budou hrany svazovat. Při nastavení výššího úhlu je pravděpodobnější, že se sváže více hran, hlavně při nastavení parametru neighbors na výšší hodnoty.



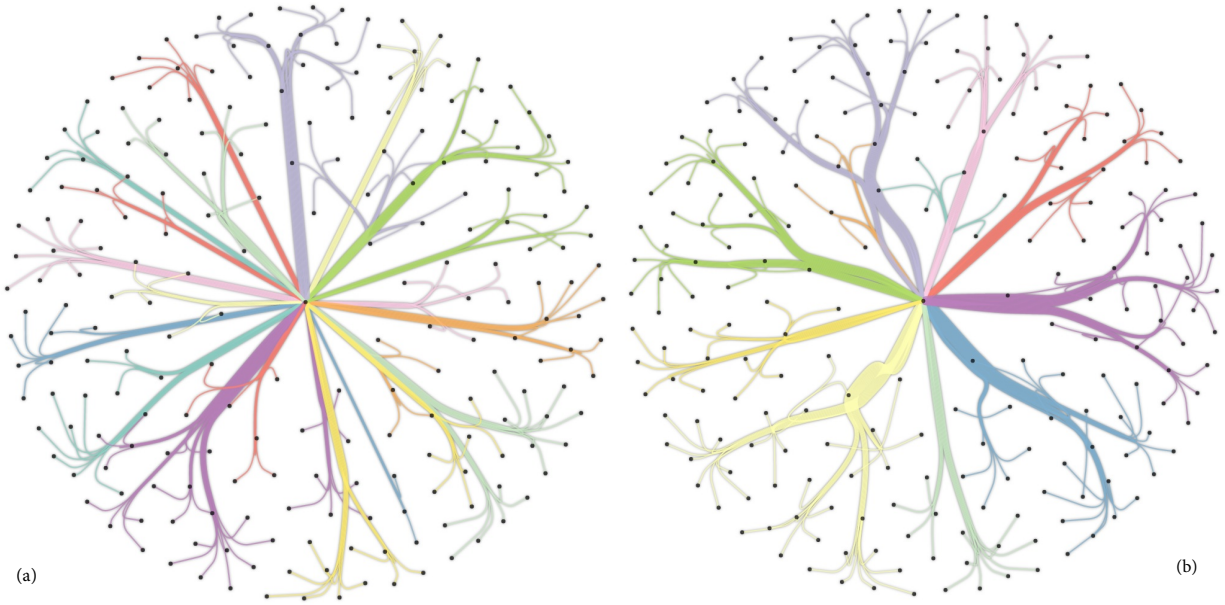
Obrázek 16: (a) Výchozí. (b) Upravený.

4.1.4 Neighbors

Parametr	**Hodnota**	
	Výchozí	Upravená
Neighbors	10	50

Tabulka 6: Změna parametru neighbors vůči výchozímu nastavení.

Tento parametr má vliv pouze na výpočet grafu, ostatní nastavení jsou shodná s výchozím jako u obrázku č.13, dle tabulky č.2. Pomocí tohoto parametru lze nastavit maximální počet hledaných sousedů, kteří budou hledáni ve struktuře kdTree.



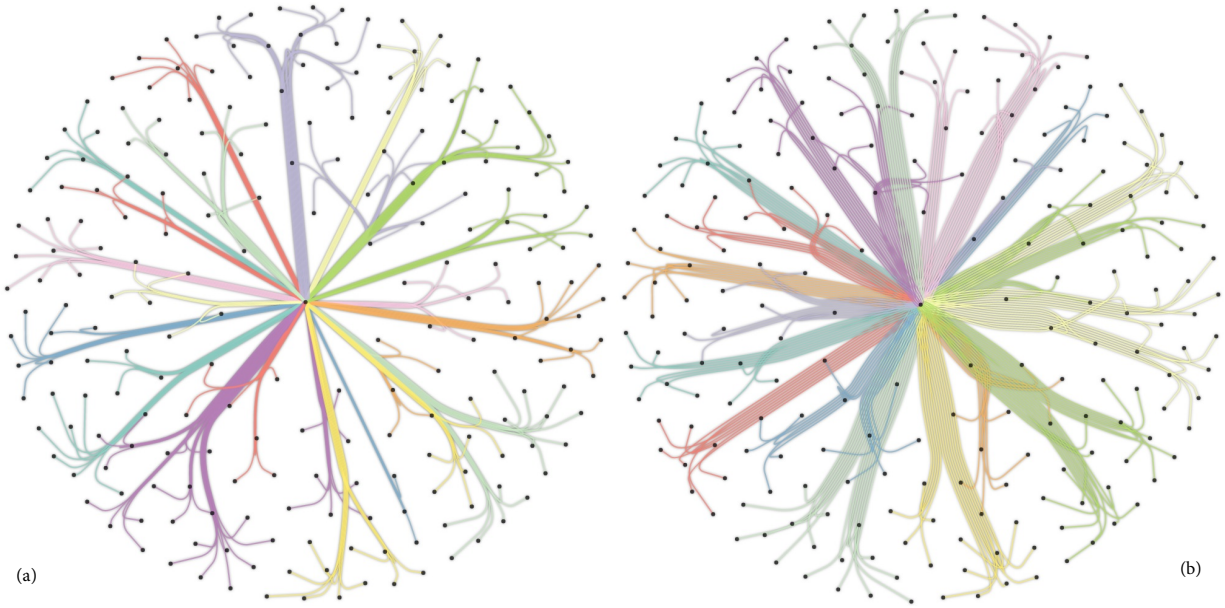
Obrázek 17: (a) Výchozí. (b) Upravený.

4.1.5 Margin

Parametr	Hodnota	
	Výchozí	Upravená
Margin	0	2.5

Tabulka 7: Změna parametru margin vůči výchozímu nastavení.

Tento parametr má vliv pouze na vykreslování grafu, ostatní nastavení jsou shodná s výchozím jako u obrázku č.13, dle tabulky č.2. Pomocí tohoto parametru lze vizuálně rozdělit svazek hran opět na hrany, které jsou reprezentovány kvadratickými křivkami.



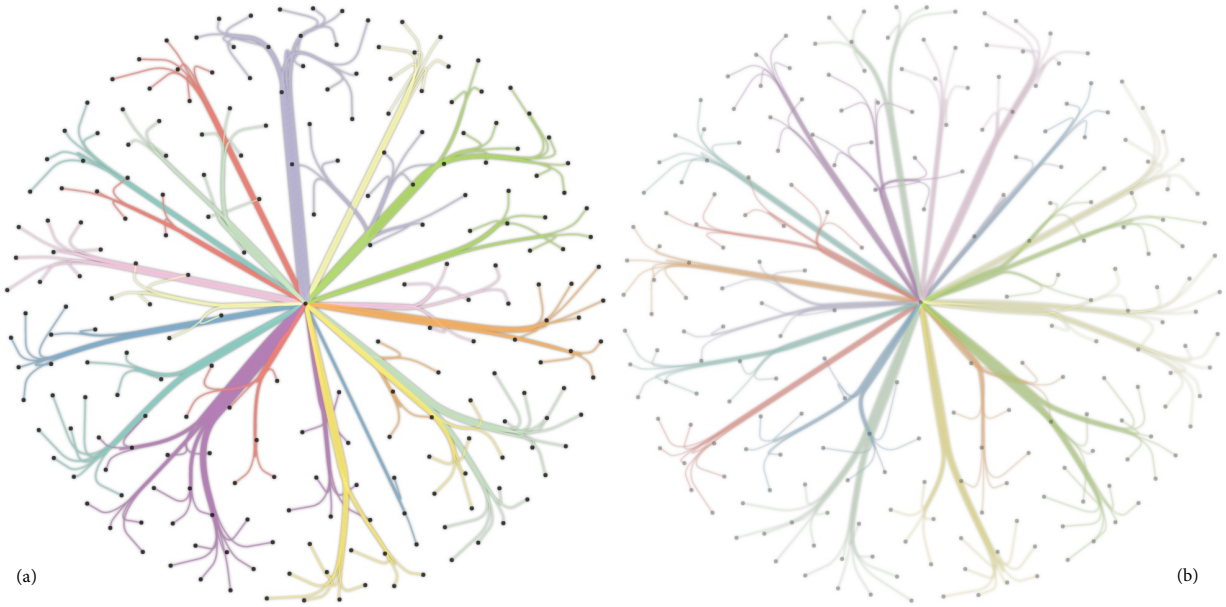
Obrázek 18: (a) Výchozí. (b) Upravený.

4.1.6 Alpha

Parametr	Hodnota	
	Výchozí	Upravená
Alpha	0.7	0.25

Tabulka 8: Změna parametru alpha vůči výchozímu nastavení.

Tento parametr má vliv pouze na vykreslování grafu, ostatní nastavení jsou shodná s výchozím jako u obrázku č.13, dle tabulky č.2. Tento parametr nastavuje průhlednost výsledného grafu.



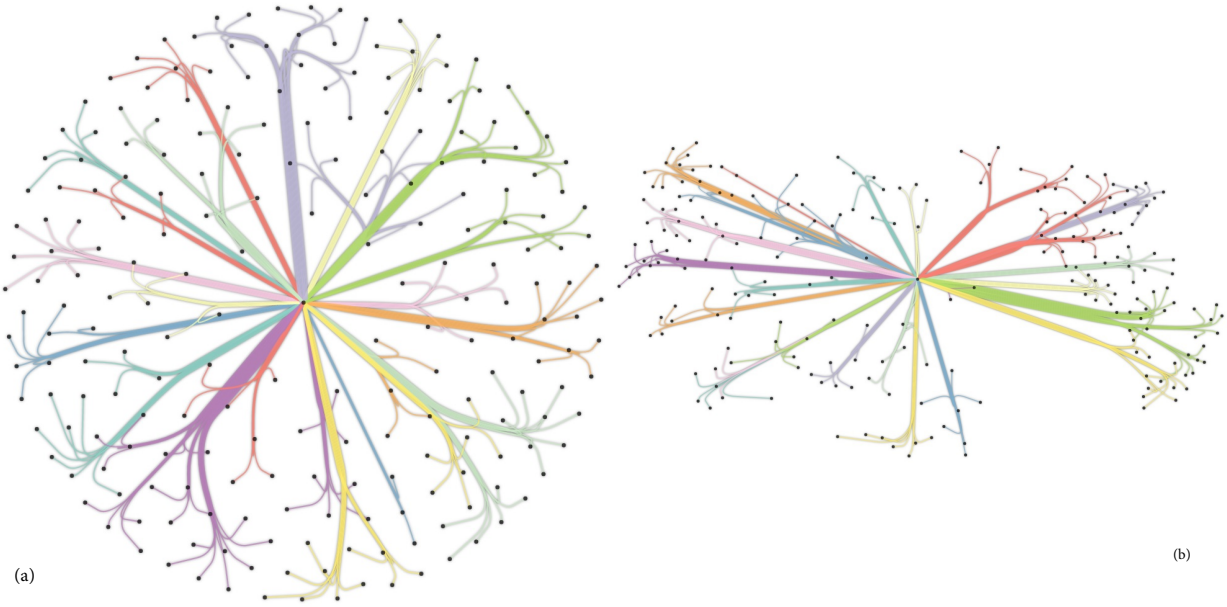
Obrázek 19: (a) Výchozí. (b) Upravený.

4.1.7 Layout iterations

Parametr	Hodnota	
	Výchozí	Upravená
Layout iterations	200	20

Tabulka 9: Změna parametru layout iterations vůči výchozímu nastavení.

Tento parametr má vliv pouze na výpočet grafu, ostatní nastavení jsou shodná s výchozím jako u obrázku č.13, dle tabulky č.2. Tento parametr určuje počet průchodu layout algoritmem, na obrázku lze vidět, že algoritmus nestačil srovnat body tak, aby výsledné síly v grafu byly nulové.



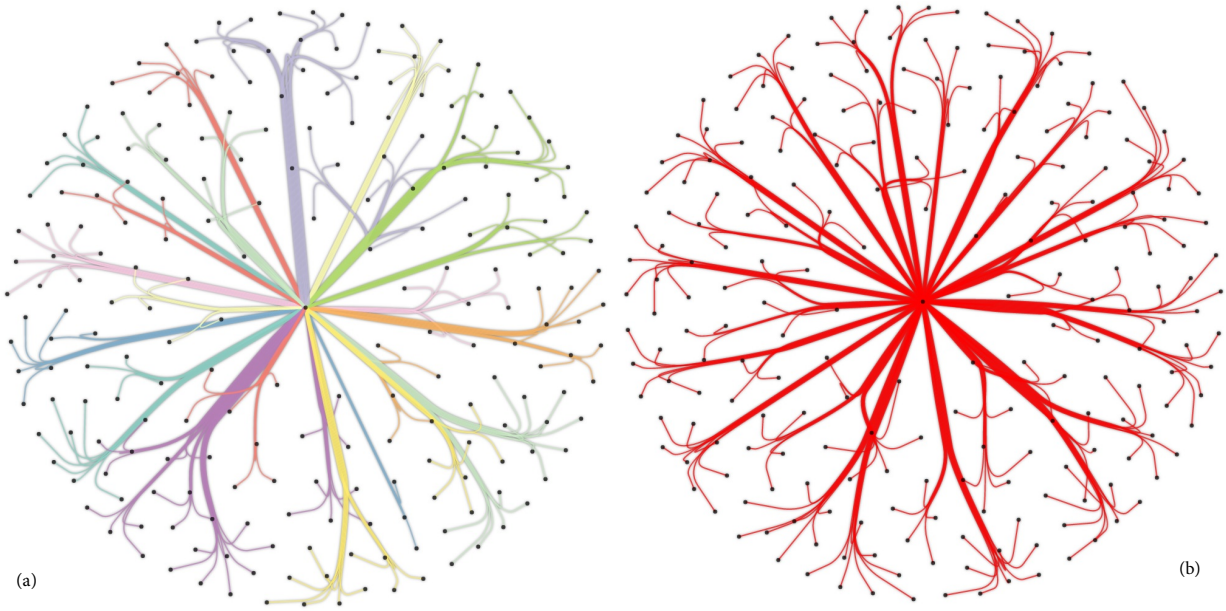
Obrázek 20: (a) Výchozí. (b) Upravený.

4.1.8 Solid Color

Parametr	Hodnota	
	Výchozí	Upravená
Solid Color	Orange	Red
Color type	Cluster	Solid

Tabulka 10: Změna parametru solid color a color type vůči výchozímu nastavení.

Parametr solid color má vliv pouze na vykreslování grafu, ostatní nastavení jsou shodná s výchozím jako u obrázku č.13, dle tabulky č.2. Pomocí tohoto parametru se nastavuje barva pro všechny hrany grafu při vizualizaci.



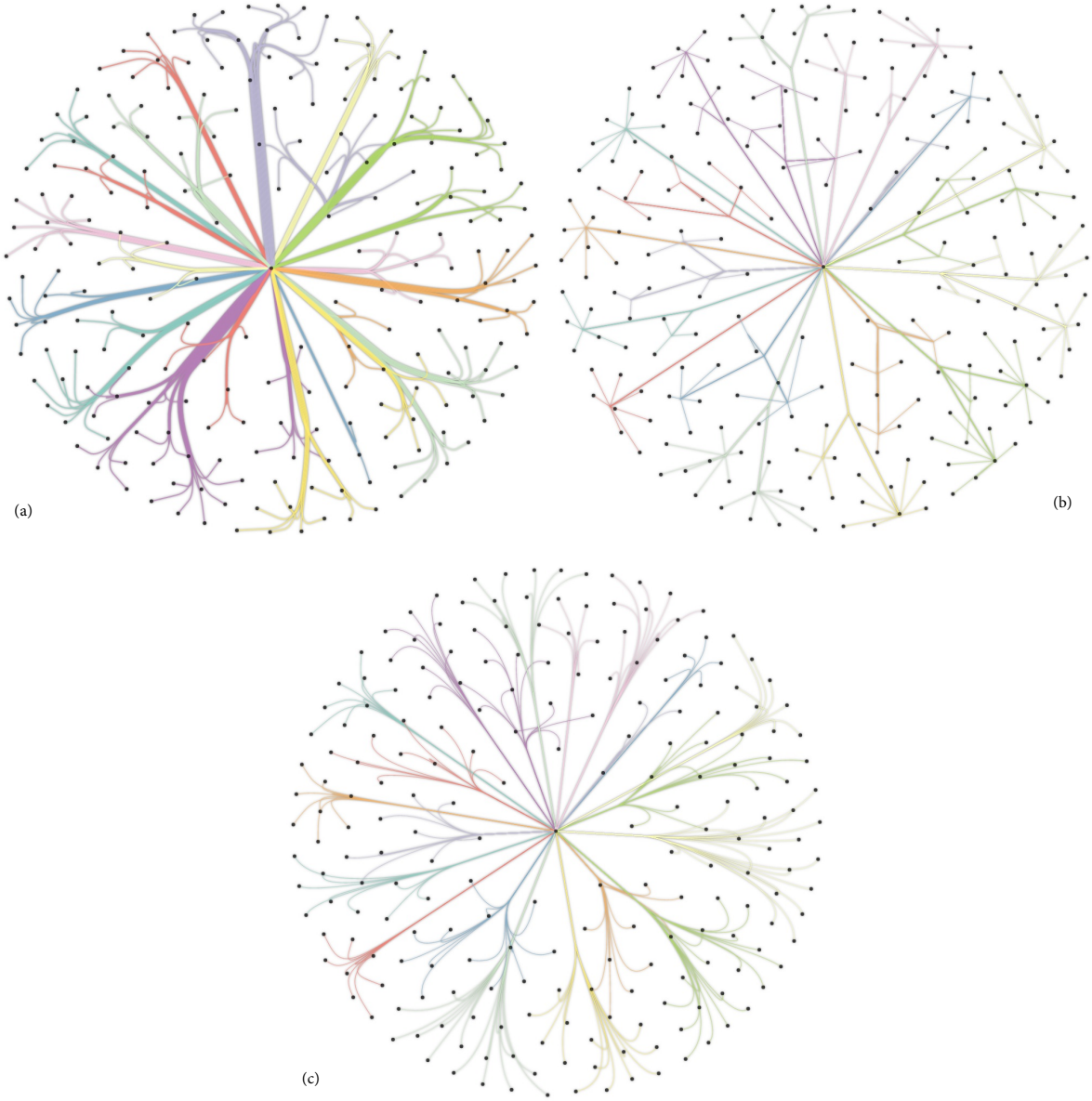
Obrázek 21: (a) Výchozí. (b) Upravený.

4.1.9 Line type

Parametr	**Hodnota**		
	Výchozí	Upravená-line	Upravená-bezier
Line type	Quadratic	Line	Bezier

Tabulka 11: Změna parametru line type vůči výchozímu nastavení.

Tento parametr má vliv pouze na vykreslování grafu, ostatní nastavení jsou shodná s výchozím jako u obrázku č.13, dle tabulky č.2.

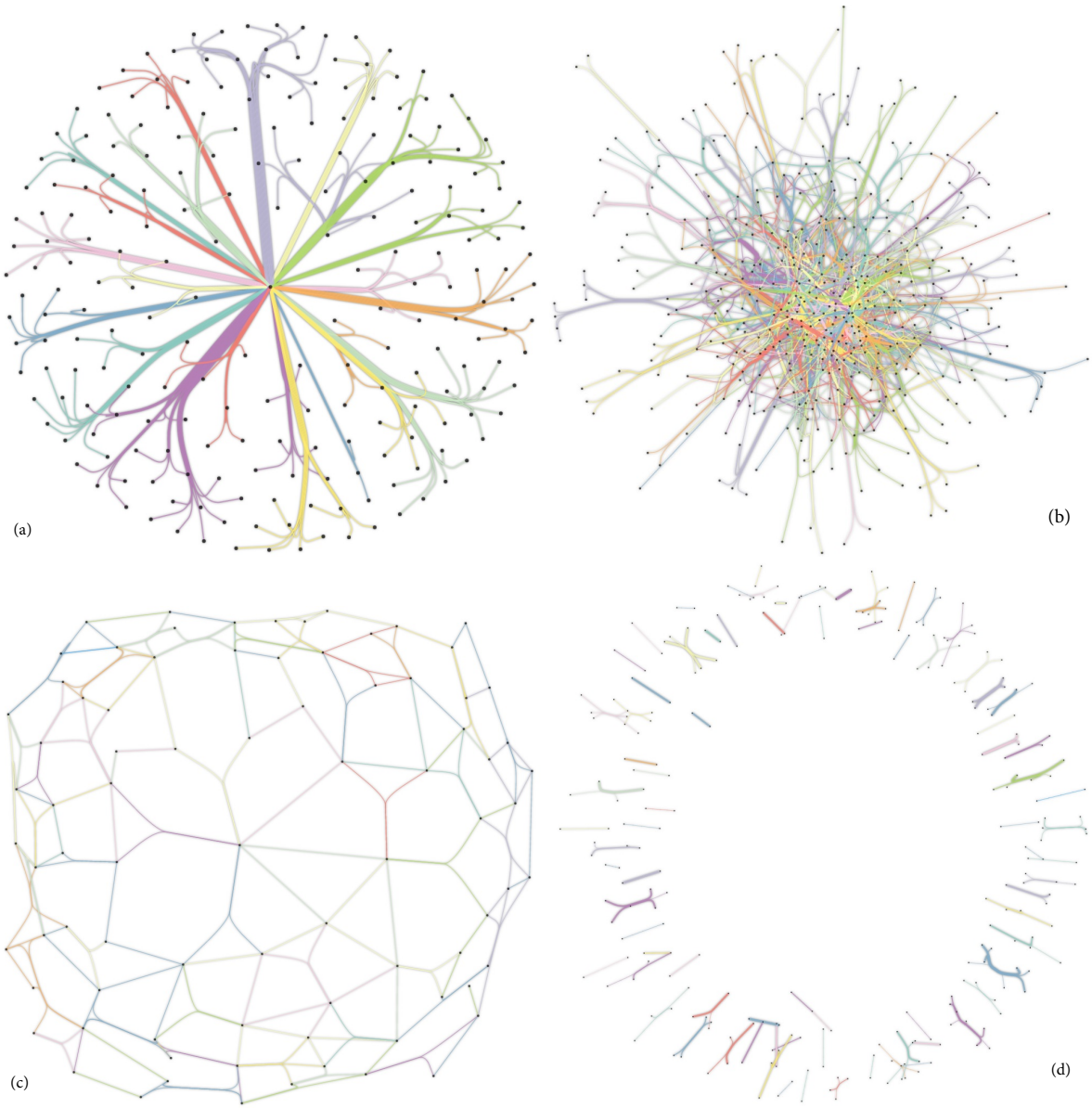


Obrázek 22: (a) Výchozí. (b) Upravený-line. (c) Upravený-bezier

4.1.10 Dataset

Parametr	**Obrázek**			
	(a)	(b)	(c)	(d)
Dataset	Star graph	Connected graph	Planar graph	Componnent graph
Neighbors	10	40	10	10

Tabulka 12: Grafy a jejich nastavení v obrázku č.23.



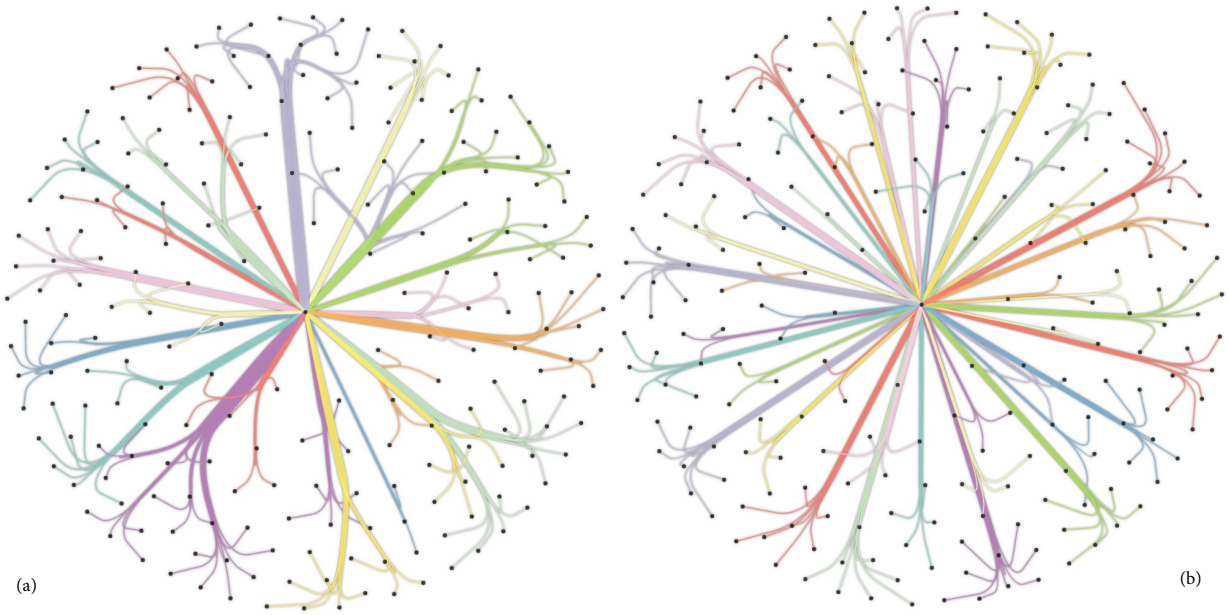
Obrázek 23: Různé typy grafů.

4.1.11 Recursion

Parametr	**Hodnota**	
	Výchozí	Upravená
Recursion	2.Recursion	1.Recursion

Tabulka 13: Změna parametru recursion vůči výchozímu nastavení.

Tento parametr má vliv pouze na výběr výstupních dat z algoritmu, ostatní nastavení jsou shodná s výchozím jako u obrázku č.13, dle tabulky č.2. Pomocí tohoto parametru lze vybrat, který výsledek průchodu algoritmem bude vykreslen.



Obrázek 24: (a) Výchozí. (b) Upravený.

4.2 Výkonnostní testování

4.2.1 Testovací sestava:

Procesor:	AMD Phenom II X4 965 BE, 4.0 GHZ
Základní deska:	Asus M4A88TD-V EVO
Paměť:	Crucial Ballistix Tactical Tracer DDR3 12GB
Grafické karty:	AMD Sapphire HD 6850 Toxic, AMD VT-X HD 6870
Disk:	Seagate Barracuda 3TB, 7200 rpm
Operační systém:	Windows 7 Ultimate 64bit

Tabulka 14: Testovací sestava

4.2.2 Testovací postup:

Testování bylo prováděno po restartu počítače, byl spuštěn pouze prohlížeč, ve kterém byla spuštěna pouze aplikace. Měření délky výpočtu bylo provedeno přímo v aplikaci za pomoci příkazu Date.now(), tato metoda vrací počet milisekund, které uplynuly od 1.ledna 1970 00:00:00 UTC.
Výpočet času vypadal následovně:

```
var begin = Date.now();

   //Mereny usek kodu

var end = Date.now();
console.log("Calculated in " + (end - begin) / 1000 + "secs");
```

Výpis 2: Část kódu použita k měření času.

Do konzole bude vypsán čas ve vteřinách, který strávil algoritmus výpočtem. Do měřeného úseku je zahrnuto: načtení dat z JSONu, načtení informací k vrcholům z JSONu, aplikování force-directed layoutu, vycentrování grafu, vytvoření proximitního grafu a spočítání výsledného grafu pomocí algoritmu MINGLE.

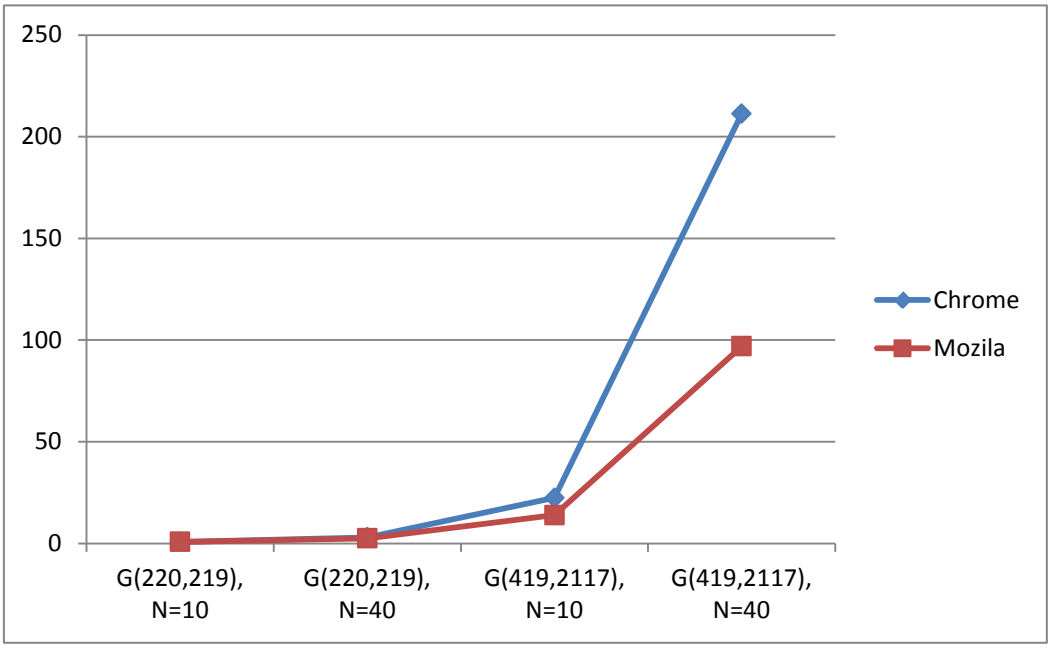
Měřeni bylo provedeno za použití dvou grafových struktur (Star graph, která má 220 vrcholů, 219 hran a Connected graph, která má 419 vrcholů, 2117 hran) za nastavení parametru Neighbors na hodnoty 10 a 40. Výsledky jsou porovnávány mezi prohlížeči Google Chrome a Mozilla Firefox. Po každém nastavení byl změřen čas několikrát a z daných časů byl spočítán aritmetický průměr, který je brán jako výsledek daného nastavení.

4.2.3 Výsledky testů:

Graf / Prohlížeč	Google Chrome	Rozdíl	Mozilla Firefox
G(220,219), N=10	0,769s	0,008s	0,761s
G(220,219), N=40	2,943s	0,482s	2,461s
G(419,2117), N=10	22,418s	8,5s	13,918s
G(419,2117), N=40	211,292s	114,3s	96,938s

Tabulka 15: Výkonnostní testy prohlížečů.

Při aplikaci algoritmu na větší objem data je rozhodně vhodné použít prohlížeč Mozilla Firefox, jak z hlediska rychlosti tak i podle stability výpočetního času. U prohlížeče Google Chrome jsou větší rozdíly mezi každým měřením při stejných podmínkách a parametrech. U prohlížeče Mozilla Firefox se tyto výsledky liší jen málo.
Na obrázku č.28 jsou zobrazeny výsledky testů v grafu, ze kterého je viditelné, že Mozilla Firefox je mnohem rychlejší při zvyšování velikosti vstupních dat nebo vyšší náročnosti výpočtu dle parametrů.



Obrázek 25: Výsledek testu zobrazený v grafu.

5 Závěr

Výsledkem práce je aplikace, která ukazuje možnosti vizualizace grafů za pomocí Force-Directed layout algoritmu a Multilevel Agglomerative Edge Bundling algoritmu, které jsou implementovány v programovacím jazyku JavaScript. Aplikace umožňuje nastavení parametrů jak pro výpočetní algoritmy, tak i pro algoritmy zabývající se vizualizací výsledného grafu. Aplikace pracuje s předem připravenými daty formátu JSON, které jsou načteny a zobrazeny po úpravě výpočetních a vizualizačních algoritmů. Aplikace poskytuje i funkce pro práci s výsledným grafem(přiblížení grafu, posun, přesouvání vrcholů, zobrazení informací o vrcholu). Experimentální část práce popisuje vliv nastavení parametrů na výsledný graf a také výkonnostní porovnání aplikace, která je spuštěna pod rozdílnými prohlížeči.

Aplikaci lze využít k vizualizaci a zpřehlednění vlastních dat, která jsou reprezentována jako grafová struktura. Podobných aplikací, knihoven a nástrojů na úpravu a vizualizaci grafů je mnoho, přičemž některé jsou zpoplatněné, bez zdrojových kódů, velmi složité knihovny nebo neposkytují potřebné funkce. Tato aplikace nabízí jednoduché aplikování uživatelských požadavků na vizualizaci vlastních dat a práci s nimi.

Práce na tomto projektu mě bavila a byla pro mě přínosná jak z hlediska získání nových zkušeností, tak i práce v novém prostředí a programovacím jazyku, se kterým jsem před projektem neměl zkušenosti. Vývoj tohoto projektu byl pro mě časově náročný, už jen z důvodu absence zkušeností s novým prostředím a programovacím jazykem. Výsledná aplikace splňuje zadané požadavky i mé vlastní cíle, které jsem si určil před začátkem práce na projektu.

Do budoucna bych se k této aplikaci chtěl vrátit a rozšířit její funkcionalitu a zkusit použití jiných technologií na výpočet a vizualizaci grafů.

Literatura

[1] Michael J Bannister, David Eppstein, Michael T Goodrich, and Lowell Trott. Force-directed graph drawing using social gravity and scaling. In *Graph Drawing*, pages 414–425. Springer, 2012.

[2] Di Battista, Peter Eades, Ioannis G Tollis, and Roberto Tamassia. *Graph drawing: algorithms for the visualization of graphs*. 1999.

[3] Christian Collberg, Stephen Kobourov, Jasvir Nagra, Jacob Pitts, and Kevin Wampler. A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 ACM symposium on Software visualization*, pages 77–ff. ACM, 2003.

[4] Hooman Reisi Dehkordi, Quan Nguyen, Peter Eades, and Seok-Hee Hong. *Circular graph drawings with large crossing angles*. Springer, 2013.

[5] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry*, 4(5):235–282, 1994.

[6] Peter Eades, Qing-Wen Feng, and Xuemin Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In *Graph drawing*, pages 113–128. Springer, 1996.

[7] Alon Efrat, Cesim Erten, and Stephen G Kobourov. Fixed-location circular-arc drawing of planar graphs. In *Graph Drawing*, pages 147–158. Springer, 2003.

[8] Qingwen Feng. *Algorithms for drawing clustered graphs*. PhD thesis, Citeseer, 1997.

[9] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[10] Pawel Gajer, Michael T Goodrich, and Stephen G Kobourov. A fast multi-dimensional algorithm for drawing large graphs. In *Graph Drawing'00 Conference Proceedings*, pages 211–221, 2000.

[11] Emden R Gansner, Yifan Hu, Stephen North, and Carlos Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *Visualization Symposium (PacificVis), 2011 IEEE Pacific*, pages 187–194. IEEE, 2011.

[12] Frank Harary. The determinant of the adjacency matrix of a graph. *Siam Review*, 4(3):202–210, 1962.

[13] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):741–748, 2006.

[14] Danny Holten and Jarke J Van Wijk. Force-directed edge bundling for graph visualization. In *Computer Graphics Forum*, volume 28, pages 983–990. Wiley Online Library, 2009.

[15] Juraj Hromkovič, Manfred Nagl, and Bernhard Westfechtel. *Graph-Theoretic Concepts in Computer Science: 30th International Workshop, WG 2004, Bad Honnef, Germany, June 21-23, 2004, Revised Papers*, volume 3353. Springer, 2005.

[16] Stephen G Kobourov. Spring embedders and force directed graph drawing algorithms. *arXiv preprint arXiv:1201.3011*, 2012.

[17] Daniel Lai. R-chie: a web server and R package for visualizing RNA secondary structures. `http://www.e-rna.org/r-chie/images/overlap.png`, 2012. [Online; Updated: March 18, 2014].

[18] Kurt Mehlhorn and Stefan Näher. *LEDA: a platform for combinatorial and geometric computing*. Cambridge university press, 1999.

[19] Quan Nguyen. The university of Sydney. `http://rp-www.cs.usyd.edu.au/~qnguyen/edgebundling/figures/dense_CC.png`, 2015. [Online; accessed: 2015].

[20] t a Augusts Bucket. PhotoBucket. `http://s1184.photobucket.com/user/t_a_august/media/FA_1.jpg.html`, 2016. [Online; accessed: 2016].

[21] Rowan Udell. DevOps, cloud, and code. `http://blog.rowanudell.com/content/images/2016/03/Screen-Shot-2016-03-13-at-11-18-52-AM.png`, 2015. [Online; accessed: 2015].

[22] Jianhua Yang, Vladimir Estivill-Castro, and Stephan K Chalup. Support vector clustering through proximity graph modelling. In *Neural Information Processing, 2002. ICONIP'02. Proceedings of the 9th International Conference on*, volume 2, pages 898–903. IEEE, 2002.

A Příloha

Příloha na CD.

Obsah CD:

- Složka "Classes"obsahuje zdrojové kódy a knihovny k aplikaci.
- Složka "Data"obsahuje vstupní data aplikace.
- Složka "Styles"obsahuje css styly.
- Dokument "Edge Bundling.html"je spouštěč aplikace.