VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Hyperbolické prostory a vizualizace grafu

Hyperbolic Space and Graph Visualization

2012 Bc. Jakub Lojkásek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jakub Lojkásek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Hyperbolické prostory a vizualizace grafu
Hyperbolic Space and Graph Visualization

Zásady pro vypracování:

Cílem diplomové práce je navázat na práci o hyperbolických prostorech vytvořenou na katedře informatiky. Po prostudování problematiky hyperbolických prostorů se diplomant zaměří na problematiku vizualizace grafu pomocí hyperbolických prostorů. K vizualizaci může diplomant použít již existující algoritmy pro vizualizaci grafu a jejich výsledek následně modifikovat hyperbolickým prostorem.

Mimo teoretické části a části zabývající se hyperbolickými prostory se diplomant zaměří také na prostudování vyvíjeného standard HTML5 s návazností na vizualizaci grafu. Jako příklad může být SVG. Nedílnou součástí k zobrazení nějakých informací jsou samotná data. V rámci práce se bude jednat o data získaná pomocí Google API.

Jednotlivé cíle práce jsou:
1. Prostudovat problematiku hyperbolických prostorů.
2. Prostudovat technologie pro vizualizaci dat v rámci webového prohlížeče a detailně popsat vybranou technologii.
3. Implementovat stahování výsledků slovního vyhledání pomocí Google API s návazností na Google Cache.
4. Implementovat vizualizační prostředí využívající hyperbolické prostory.
5. Provedení experimentů s vizualizací s ohledem na kvalitu zobrazení a rychlost.
6. Zhodnocení výsledků.
7. Popsání možných dalších rozšíření vyvinutého vizualizačního algoritmu.

Seznam doporučené odborné literatury:

Zhang, Jin: Visualization for Information Retrieval, The Information Retrieval Series, Vol. 23, 2008, XVIII.

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

Vedoucí diplomové práce: **Ing. Jan Martinovič, Ph.D.**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

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

V Ostravě 4. května 2012 .....

Rád bych na tomto místě poděkoval vedoucímu mé diplomové práce Ing. Janu Martinovičovi Ph.D. za poskytnuté informace a věnovaný čas.

Abstrakt

Tato diplomová práce popisuje vizualizaci rozsáhlých grafů v hyperbolickém prostoru. Věnuje se modifikaci již existujících algoritmů pro vizualizaci grafu hyperbolickým prostorem. Je navrženo řešení implementace pro rozmístění vrcholů grafu v hyperbolickém prostoru za využití modelů pro vizualizaci hyperbolických prostorů a prostorového promítání. Mimo teorie o hyperbolickém prostoru se práce zabývá i problematikou vizualizace v internetovém prohlížeči pomocí SVG, algoritmy pro vizualizaci grafu a stahováním výsledků pomocí Google API.

Klíčová slova: Hyperbolický prostor, hyperbolická geometrie, hyperbolický layout, vizualizace grafu, Gephi, IKVM, C#, ASP, .NET, Java, JavaScript, D3.JS, JSON.

Abstract

This thesis describes the visualization of large graphs in hyperbolic space. This thesis shows modification of existing algorithms for graph visualization by hyperbolic space. There is proposed solution of implementation for layout graph vertices in hyperbolic space using models for visualizing in hyperbolic space and spatial projection. In addition to the theory of hyperbolic space, this work deals with problems of vizualization in the web browser using SVG, algorithms for the graph visualization and downloading results by Google API.

Keywords: Hyperbolic space, hyperbolic geometry, hyperbolic layout, graph visualization, Gephi, IKVM, C#, ASP, .NET, Java, JavaScript, D3.JS, JSON.

Seznam použitých zkratek a symbolů

$\mathbb{E}^3$	–	Třírozměrný euklidovský prostor
$\mathbb{H}^2$	–	Dvourozměrný hyperbolický prostor, hyperbolická rovina
$O()$	–	Časová složitost
API	–	Application Programming Interface – Rozhraní pro programování aplikací
DOM	–	Document Object Model – Objektový model dokumentu
GUI	–	Graphical User Interface – Grafické uživatelské rozhraní
HTML	–	HyperText Markup Language – Hypertextový značkovací jazyk
IKVMC	–	IKVM Compiler – IKVM Kompilátor
IL	–	Intermediate Language – Mezijazyk
JS	–	JavaScript – Objektově orientovaný skriptovací jazyk
JSON	–	JavaScript Object Notation – JavaScriptový objektový zápis
JVM	–	Java Virtual Machine – Virtuální stroj Java
LINQ	–	Language-Integrated Query – Integrovaný jazyk pro dotazování
SVG	–	Scalable Vector Graphics – Škálovatelná vektorová grafika
W3C	–	World Wide Web Consortium – Mezinárodní konsorcium celosvětové internetové sítě
XHTML	–	eXtensible HyperText Markup Language – Rozšiřitelný hypertextový značkovací jazyk
XML	–	Extensible Markup Language – Rozšiřitelný značkovací jazyk
REST	–	Representational State Transfer – Přenos reprezentačního stavu
HTTP	–	Hypertext Transfer Protocol – Protokol pro přenos hypertextových dokumentů
SOAP	–	Simple Object Access Protocol – Protokol pro přístup k jednoduchým objektům
MB	–	Megabyte – Megabajt = 2^{20} neboli 1 048 576 bajtů
UTF-8	–	UCS Transformation Format-8bit – 8bitový transformační formát UCS
UCS	–	The Universal Character Set – Univerzální znaková sada

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tato diplomová práce je zaměřena na vizualizaci rozsáhlých grafů v hyperbolickém prostoru. Existuje mnoho aplikací využívajících rozmístění vrcholů grafu v hyperbolickém prostoru (dále jen „hyperbolický layout“). V těchto aplikacích je převážně používán stromový graf (např. program HyperbolicSpace pro vizualizaci sociálních sítí do hyperbolických prostorů vytvořený v práci [23]). Takové řešení je nevyhovující pro komplexní síť. Díky stromové struktuře není v těchto aplikacích potřeba vytvářet layout grafu. Layout je tvořen postupným vykreslováním stromu do hyperbolického prostoru, kdy jsou vrcholy od sebe konstantně vzdáleny (viz [23]).

Rozmístění vrcholů rozsáhlého grafu v prostoru není triviálním úkolem. Hlavním cílem této diplomové práce je vytvořit hyperbolický layout, pomocí kterého by bylo umožněno přehledně vizualizovat rozsáhlé grafy v hyperbolických prostorech. Takový layout by měl odstraňovat výše zmíněný nedostatek a tím poskytovat detailní pohled na vybraný vrchol zachovávající propojení okolních vrcholů.

Dále je v práci popsána vybraná technologie pro vizualizaci dat v rámci webového prohlížeče. Jedná se o JavaScriptovou knihovnu D3.JS [1], která využívá dat ve formátu JSON a vektorové grafiky SVG.

1.1 Struktura práce

V kapitole 2 se budeme zabývat problematikou hyperbolických prostorů. Popíšeme Eukleidovu geometrii (sekce 2.1) a způsob, jakým vznikla geometrie hyperbolická (sekce 2.3). Následně si představíme modely hyperbolického prostoru (sekce 2.4) a možnosti jejich využití při vizualizaci grafu do hyperbolického prostoru (sekce 2.5).

Kapitola 3 popisuje algoritmy pro vizualizaci grafů se zaměřením na algoritmy založených na silách (sekce 3.1). Jsou zde popsány principy vybraných algoritmů a vztahy mezi nimi.

V kapitole 4 je popsáno, jak lze s grafem pracovat pomocí API programu Gephi za použití programovacího jazyka C#. Programu Gephi je představen v sekci 4.1. Funkce tohoto programu můžeme využít ve vlastních aplikacích pomocí Gephi Toolkitu (odstavec 4.1.1). Jelikož je Gephi Toolkit napsán v programovacím jazyce Java, využijeme program IKVM (sekce 4.2) abychom mohli Gephi Toolkit ovládat programovacím jazykem C#. Následuje popis importu a exportu grafu využitím programů Gephi Toolkit a IKVM (sekce 4.3).

Kapitola 5 se zabývá využitím JavaScriptu a vektorové grafiky SVG pro vizualizaci grafu v internetovém prohlížeči. Pro přenos dat o grafu je použit JavaScriptový objektový zápis JSON (sekce 5.1). Pro práci s tímto souborovým formátem byl vyvinut Framework Newtonsoft.Json (odstavec 5.1.1). V rámci vyvíjeného standardu HTML5 (sekce 5.2) byla integrována podpora vektorové grafiky SVG (sekce 5.3). Pro práci s touto grafikou byla vyvinuta JavaScriptová technologie – knihovna D3.JS (sekce 5.4). Způsob práce s touto knihovnou je popsán v odstavci 5.4.1.

V kapitole 6 je popsána webová služba Google API (sekce 6.2) a způsob, jakým lze pomocí této služby získat výsledky slovního vyhledávání ve vlastních aplikacích. S ná-

vazností na Google Cache je vytvořeno stahování výsledků (sekce 6.3) a jejich zdrojových kódů pomocí Google API.

Kapitola 7 se zabývá vizualizací grafu v hyperbolickém prostoru. V sekci 7.1 navrhneme možné řešení pro vytvoření hyperbolického layoutu. V sekci 7.2 je popsáno propojení jednotlivých komponent využitých při realizaci navrhovaného řešení. Vlastní implementací hyperbolického layoutu se zabýváme v sekci 7.3.

Experimenty s vizualizací s ohledem na kvalitu zobrazení a rychlost jsou uvedeny v kapitole 8. Zhodnocení výsledků získaných při experimentech je popsáno v sekci 8.1. Závěr je vysloven v kapitole 9 a v sekci 9.1 jsou uvedeny návrhy možného rozšíření vyvinutého vizualizačního algoritmu.

2 Hyperbolické prostory

Tato kapitola se zabývá problematikou hyperbolických prostorů. Je zde popsána Eukleidova geometrie (sekce 2.1) a způsob, jakým vznikla geometrie hyperbolická (sekce 2.3), která definuje hyperbolický prostor. Dále je popsán vztah Neeukleidovy geometrie (sekce 2.2) a absolutní geometrie (sekce 2.2.1) k hyperbolické geometrii. V sekci 2.4 jsou popsány modely pro vizualizaci v hyperbolických prostorech. Po srovnání těchto modelů je dále uvedeno možné využití hyperbolických prostorů pro vizualizaci grafu.

2.1 Eukleidova geometrie

Řecký matematik a geometr Eukleidés[1] (asi 365 - 280 př. n. l.) sepsal dílo nazvané „Základy"[15]. V této publikaci uvádí definice a axiomy, na základě kterých je založena nejstarší část geometrie a tou je právě Eukleidova geometrie.

Definice 2.1 *Axiom je tvrzení, které se předem pokládá za platné, a tudíž se nedokazuje.*

Pět základních axiomů definovaných Eukleidem[2][15]:

Axiom 1. Každými dvěma body lze vždy vést jedinou přímku.

Axiom 2. Úsečku lze neomezeně prodloužit.

Axiom 3. Z libovolného středu lze sestrojit kružnici o libovolném poloměru.

Axiom 4. Všechny pravé úhly jsou shodné.

Axiom 5. K dané přímce a bodu, který na ní neleží, lze sestrojit právě jednu přímku, která prochází daným bodem a s danou přímkou se neprotíná. Takové přímky nazýváme rovnoběžky.

Pátý axiom se formuluje též takto: dvě přímky v rovině, které protínají jinou přímku této roviny a tvoří s ní po jedné straně vnitřní úhly, jejichž součet je menší dvou pravých, se vždy protínají a to po té straně přímky, kde je součet menší.

Pátý Eukleidův axiom neboli postulát o rovnoběžkách sehrál v historii geometrie nejdůležitější roli. Tento axiom je výrazně složitější než zbylé axiomy, a proto se nejlepší světoví matematici snažili dokázat, že je tento axiom důsledkem prvních čtyř. I neúspěšné pokusy o důkaz přinášely užitek. Vznikl celý seznam vět, ekvivalentních s pátým axiomem, jako příklad můžeme uvést větu 2.1 nebo Pythagorovu větu 2.2.

Věta 2.1 *Součet vnitřních úhlů v trojúhelníku je roven dvěma pravým.*

Věta 2.2 *Obsah čtverce sestrojeného nad přeponou pravoúhlého rovinného trojúhelníku je roven součtu obsahů čtverců nad jeho odvěsnami.*

[1] Eukleidés též Euklides nebo Euklid

[2] `http://aleph0.clarku.edu/~djoyce/java/elements/bookI/bookI.html#posts`

2.2 Neeukleidova geometrie

Významným pokrokem byly pokusy o důkaz pátého axiomu sporem. Přijetím negace pátého axiomu byly odvozeny některá tvrzení geometrie, která byla odlišná od geometrie dosud studované[18]. Mnozí matematici své objevy odvozené z negace pátého axiomu zavrhli, jelikož příliš odporovaly zkušenostem z reálného světa. Matematici se také báli nepochopení a ztráty svého postavení. V roce 1829 ukončil ruský matematik Nikolaj Ivanovič Lobačevskij (1792-1856) všechny pokusy o důkaz pátého axiomu, když přijal myšlenku existence jiné geometrie a sestrojil geometrii hyperbolickou, v níž pátý axiom neplatí.

Geometriím, ve kterých neplatí pátý axiom rovnoběžnosti, říkáme neeukleidovské geometrie. Neeukleidovská geometrie je tvořena geometrií absolutní. Existuje mnoho typů neeukleidovské geometrie jako například již zmíněná hyperbolická geometrie, dále eliptická geometrie nebo sférická geometrie.

2.2.1 Absolutní geometrie

Absolutní geometrie je axiomatický systém, který je tvořen prvními čtyřmi Eukleidovy axiomy a všemi větami Eukleidovské geometrie, které lze dokázat bez použití 5. axiomu. Absolutní geometrií je tvořeno jádro všech geometrií, které jsou dále specifikovány přidáním nových axiomů.

2.3 Hyperbolická geometrie

Hyperbolická geometrie je tvořena neeukleidovskou geometrií, k níž je přidán Lobačevského axiom. Jde tedy o zavedení nového pátého axiomu. Pro tento axiom se používá mnoho jeho ekvivalentních tvrzení. Nejčastěji bývá používána věta 2.3. Tato geometrie bývá též nazývána Lobačevského neeukleidovská geometrie.

Axiom 5. Lobačevského V rovině prochází bodem neležícím na dané přímce alespoň dvě různé s danou přímkou se neprotínající přímky.

Věta 2.3 *V rovině prochází bodem mimo danou přímku nekonečně mnoho přímek, které danou přímku neprotnou.*

Jak již bylo zmíněno, už v dávné historii matematici své objevy ohledně neeukleidovské geometrie odmítali, protože příliš odporovaly dosavadním poznatkům geometrie a myšlení reálného světa. Pro člověka je velmi těžké a nepřirozené si tento hyperbolický prostor představit a uvažovat v něm. Avšak existují mnohé modely vizualizace hyperbolických prostorů:

1. Dvoudílný hyperboloid,
2. Jednodílný hyperboloid,
3. Hyperbolický paraboloid (sedlo),

4. Pseudosféra,
5. Klein–Beltrami model,
6. Polokulový model,
7. Half–Plane model,
8. Poincaré Half–Plane model a
9. Poincaré Disk model.

Nevýhodou prvních třech zmíněných modelů může být, že jejich zakřivení plochy není konstantní. Příkladem plochy se záporným konstantním zakřivením je čtvrtý model. Další modely jsou pak postupně odvozeny z modelu prvního a různými úpravami vznikají modely další. Například Klein–Beltrami model je odvozen od Dvoudílného hyperboloidu, Half–Plane model je odvozen od Klein–Beltrami modelu a Poincaré Half–Plane model i Poincaré Disk model jsou odvozeny od Half–Plane modelu. Definováním hyperbolické geometrie a jejich modelů říkáme, jak bude hyperbolický prostor vypadat.

2.4 Modely hyperbolického prostoru

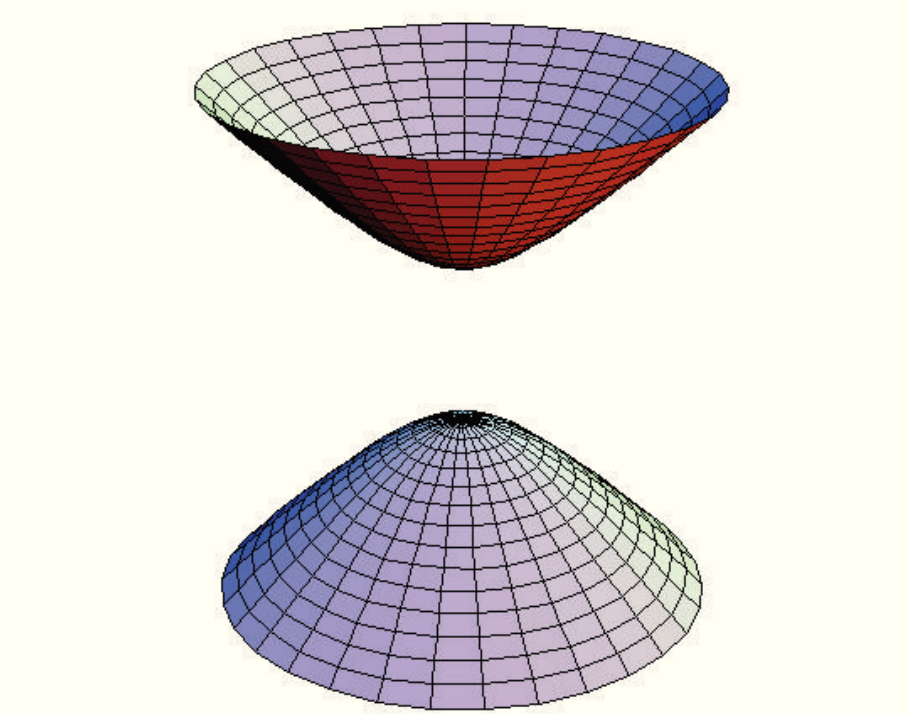
V následujících odstavcích jsou popsány jednotlivé modely hyperbolických prostorů a způsob jejich vzniku. V závěru je uvedeno srovnání vybraných modelů.

2.4.1 Hyperboloid model

Jako model dvourozměrného hyperbolického prostoru $\mathbb{H}^2$ budeme uvažovat povrch dvoudílného hyperboloidu v prostoru $\mathbb{E}^3$.

U dvoudílného hyperboloidu jsou ztotožněny každé dva body, které jsou symetrické podle středu hyperboloidu. Dále budeme uvažovat pouze jednodílný hyperboloid.

Ukázka dvoudílného hyperboloidu je vidět na obrázku 1.



Zdroj: http://upload.wikimedia.org/wikipedia/commons/9/91/HyperboloidOfTwoSheets.png

Obrázek 1: Hyperboloid model

2.4.2 Klein–Beltrami model

Klein–Beltrami model[3] lze odvodit z předchozího Hyperboloid modelu. Vznikne středovým promítáním hyperboloidu z jeho středu do libovolné roviny π, v níž není obsažen střed promítání a která je kolmá k jeho hlavní ose.

Definice 2.2 *Klein–Beltrami model pro $\mathbb{H}^2$ prostor je disk $\{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \prec 1\}$, ve kterém bod je stejný jako bod v Eukleidovském prostoru a přímka je definována jako spojnice dvou bodů disku (bez krajních bodů)[23].*

Disk neboli kružnici v $\mathbb{H}^2$ prostoru představuje asymptotická kuželová plocha, tedy nevlastní body hyperboloidu. Kružnice v rovině π představuje limitní nevlastní body roviny $\mathbb{H}^2$. Tato základní kružnice tedy určuje jakousi hranici prostoru, kde tato hranice je rovna nekonečnu. Body ležící na povrchu hyperboloidu se promítnou dovnitř této kružnice jako Eukleidovské body. Hyperbolická rovina $\mathbb{H}^2$ je tedy pouze vnitřek disku.

2.4.3 Polokulový model

Polokulový model lze odvodit z předchozího Klein–Beltrami modelu. Vznikne umístěním kulové plochy nad jeho základní kružnicí v $\mathbb{E}^3$ tak, aby tato kružnice byla hlavní kružnicí kulové plochy. Hyperbolickou rovinou $\mathbb{H}^2$ je v tomto modelu takto vzniklá sféra bez základní kružnice. Geometrie modelu je odvozena kolmým průmětem bodů z předchozího modelu na tuto plochu.

2.4.4 Poincaré Half–Plane model

Poincaré Half–Plane model vznikne středovým promítáním polokulového modelu. Předchozí model je promítán na rovinu α kolmou k rovině π základní kružnice se středem promítání S, který je nejvzdálenějším bodem od roviny α a leží na základní kružnici.

2.4.5 Poincaré Disk model

Poincaré Disk model vznikne středovým promítáním polokulového modelu na rovinu π základní kružnice z pólu S jeho doplňkové polosféry.

Definice 2.3 *Poincaré Disk model pro $\mathbb{H}^2$ prostor je disk $\{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \prec 1\}$, ve kterém bod je stejný jako bod v Eukleidovském prostoru a přímka je jednou z následujících[23]:*

1. *průměr jednotkové kružnice (bez hraničních bodů)*
2. *výsek (bez koncových bodů) kružnice, která protíná jednotkovou kružnici ve dvou bodech*

[3] Klein–Beltrami model též Beltrami–Klein model, projektivní model, Klein Disk model nebo Cayley–Klein model

2.4.6 Srovnání modelů

Na obrázku 2 je vidět jak vypadá stejný hyperbolický prostor zobrazený v modelech Klein–Beltrami, Poincaré Disk a Poincaré Half–Plane. Výhodou modelu Klein–Beltrami je, že přímky v tomto modelu se podobají přímkám v Eukleidovské geometrii, ale nevýhodou je, že toto zobrazení nezachovává velikost úhlů. Velkou výhodou modelu Half–Plane je, že úhly zachovává, na druhou stranu však jednou z nevýhod tohoto modelu je, že přímky v tomto modelu, se nemusí podobat přímkám z Eukleidovské geometrie. Taktéž v modelu Poincaré Disk model je výhoda zachování velikosti úhlů, přičemž se také přímky nemusí podobat přímkám z Eukleidovské geometrie. To, že se přímky nepodobají těm z Eukleidovské geometrie znamená, že přímky v tomto prostoru jsou části kruhu nebo jsou přímkami, pokud ale prochází středem disku.

Další způsoby projekce, jakým tyto modely vznikají, jsou vidět na obrázku 3, který ukazuje rozdílnou projekci. Liší se jak v posunutí roviny π, tak i v posunutí středu promítání.

2.5 Využití hyperbolických prostorů pro vizualizaci grafu

Hyperbolický prostor v našem případě představuje povrch hyperboloidu, který je neomezený. Hranici prostoru v modelech Klein–Beltrami i Poincaré Disk tvoří kružnice, která představuje asymptotickou kuželovou plochu hyperboloidu, což je jinými slovy nekonečno. Vrcholy na hyperboloidu se poté promítnou dovnitř kružnice a to tak, že vrchol našeho zájmu bude ve středu této kružnice a každý další vrchol se promítne dovnitř kružnice v závislosti na vzdálenosti od vybraného vrcholu. Čím více je vrchol vzdálen, tím více se přibližuje k hranici prostoru. Jelikož ale hranice představuje nekonečno, nikdy jej vrchol nedosáhne. Tímto vzniká jakýsi efekt roztažení grafu v prostoru, kdy se vrcholy vzdálené středu seskupují u hranice prostoru a vrcholy blízké středu jsou přehledně zobrazeny uvnitř kružnice.

Pro vizualizaci grafu v hyperbolickém prostoru tedy bude potřeba realizovat následující kroky:

1. Vykreslení hranice prostoru.
2. Určit vrchol našeho zájmu.
3. Na všechny další vrcholy aplikovat promítnutí dle zvoleného modelu.
4. Vykreslit hrany mezi vrcholy.

2.5.1 Hranice prostoru

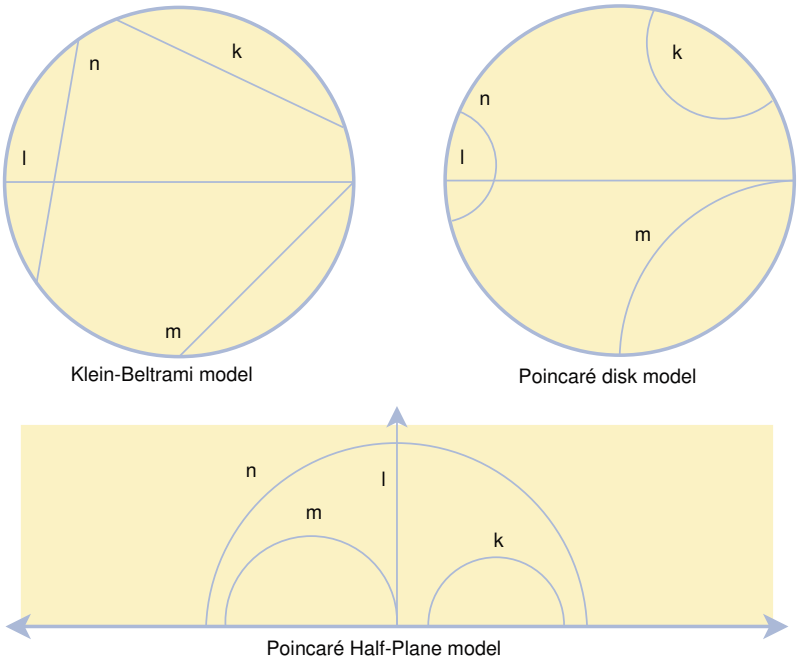
Pro hranici prostoru vykreslíme kružnici se středem uprostřed monitoru a poloměr zvolíme v závislosti velikosti monitoru tak, aby byla kružnice dostatečně velká a aby byla vidět celá. Tato kružnice bude tedy představovat hranici prostoru — nekonečno.

2.5.2 Vrcholy

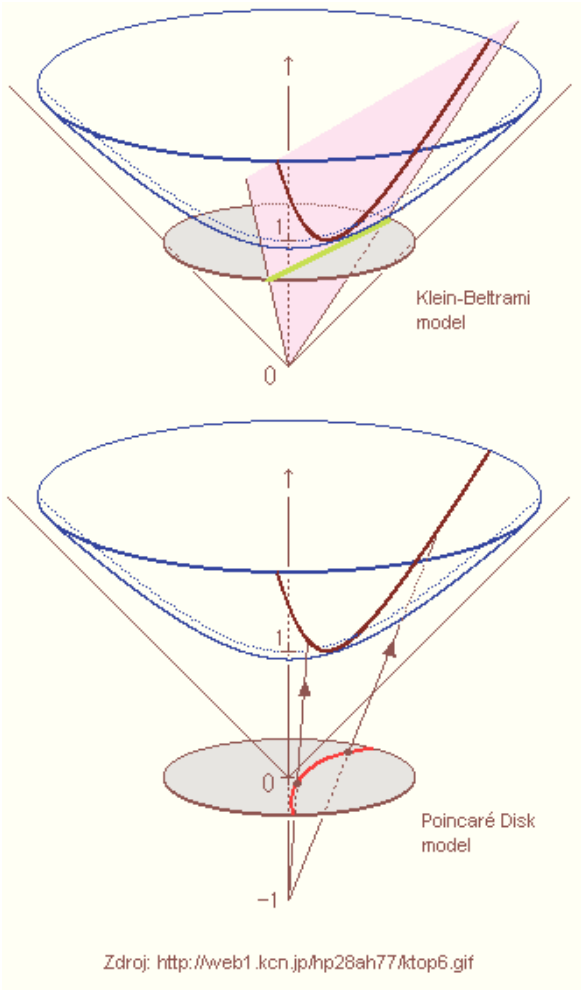
Podle zvoleného vrcholu a modelu hyperbolického prostoru vypočítáme souřadnice zobrazení bodů. Na vypočtené souřadnice poté vykreslíme vrchol, který může být zobrazen jako různé geometrické tvary jako čtverec, trojúhelník, kruh anebo může mít podobu obrázku.

2.5.3 Hrany mezi vrcholy

Jelikož se v našem případě bude jednat o grafy rozsáhlé, budou se hrany mezi vrcholy počítat na desetitisíce. Z důvodu jejich mnohonásobného křížení dochází k situaci, kdy hrany navzájem splývají a tvoří nepřehledný celek. Z tohoto důvodu nebude počítáno s jejich zakřivením a hrany budou vykresleny jako přímky spojující jednotlivé vrcholy. Ohodnocením hrany bude ovlivněna tloušťka přímky představující danou hranu.



Obrázek 2: Srovnání hyperbolických modelů



Obrázek 3: Vznik hyperbolických modelů

3 Algoritmy pro vizualizaci grafů

Pomocí algoritmů, pro vizualizaci grafů, je řešena otázka vhodného rozložení vrcholů tak, aby byl graf co nejvíce přehledný. Při výpočtu layoutu grafu by se měl zobrazovací algoritmus snažit především: minimalizovat křížení hran, minimalizovat úhly svírané jednotlivými hranami a rozložit vrcholy tak, aby byl graf co nejvíce symetrický. Je téměř nemožné optimalizovat layout grafu tak, aby co nejvíce vyhovoval všem požadavkům současně. Pro další práci byly zvoleny zobrazovací algoritmy založené na silách[4]. Při zohlednění estetických kritérií grafu mají tyto algoritmy kvalitní výsledky.

3.1 Algoritmy založené na silách

V případě algoritmů, patřících do této třídy, je s grafem pracováno jako s fyzikálním systémem. Pomocí fyzikálních zákonů (Hookův zákon, Coulombův zákon, gravitační zákony nebo magnetismus) jsou modelovány vztahy mezi vrcholy[19]. Jedná se o tzv. přitažlivé a odpudivé síly. Díky fyzikálním zákonům a fyzikální simulaci je řešen optimalizační problém, kde je hledán takový stav, kdy má celý systém nejmenší energii.

Zástupci této třídy algoritmů se většinou skládají ze dvou hlavních částí. První část implementuje vhodný silový či energetický model. Druhá část obsahuje optimalizační algoritmus, který hledá v modelu jeho optimum[24].

3.1.1 Pružinový algoritmus

Tento algoritmus publikoval v roce 1984 Petr Eades (pod názvem Spring algorithm nebo také Spring Embedder)[4]. Princip spočívá v náhodném rozmístění vrcholů grafu, nad kterými se poté vytvoří fyzikální model systému. V tomto systému jsou vrcholy reprezentovány kuličkami a hrany mezi vrcholy pružinami. Součástí modelu by mělo být modelování tlumení pohybu vrcholů tak, aby se model nerozkmital. Simulací vytvořeného modelu je hledán takový stav, kdy je energie celého systému minimální. Jelikož jsou vrcholy grafu na počátku náhodně rozmístěny, nemusí být výsledek nad stejnými daty vždy stejný.

Pružinový algoritmus používá pro výpočet působících sil vlastní vztahy a nerespektuje tak Hookův zákon, který popisuje chování pružiny. Na jednotlivé vrcholy působí buď přitažlivá (force attractive) nebo odpudivá (force repulsive) síla. Vše závisí na délce hrany v porovnání s délkou pružiny v klidové poloze. Pokud je délka hrany větší, působí na daný vrchol přitažlivá síla. Naopak pokud je délka hrany menší působí na daný vrchol síla odpudivá.

Pružinový algoritmus tedy pracuje v cyklu, dokud nejsou síly každého vrcholu v rovnováze nebo v nějakém tolerovaném rozptylu. V každém cyklu je pro každý vrchol počítána síla, kterou na něj působí okolní připojené vrcholy. Výsledná síla je rozložena na složky x a y. Hodnoty těchto složek jsou poté přičteny k původním souřadnicím vrcholu a na tyto nové souřadnice je daný vrchol přesunut. Složitost jednoho takového cyklu je $O(n^2)$, kde n je počet vrcholů.

[4] Anglicky Force–based nebo také Force–directed

Problémem pružinového algoritmu je, že může uváznout v lokálním minimu, což má dopad na kvalitu výsledků. Vrcholy, které nejsou vzájemně propojeny hranou, na sebe nepůsobí žádnou silou. Tímto může být způsobeno, že se vrcholy dostanou do velké blízkosti či se dokonce vzájemně překryjí.

3.1.2 Algoritmus Kamada–Kawai

Tomihisa Kamada a Satoru Kawai publikovali v roce 1989 dokument „An algorithm for drawing general undirected graphs"[16], ve kterém popsali algoritmus pro vykreslování obecných neorientovaných grafů, nazývaný algoritmus Kamada–Kawai. Tento algoritmus vychází z původního pružinového algoritmu. Odstraňuje však hlavní nedostatek pružinového algoritmu, kterým je absence sil působících mezi vrcholy, které nejsou vzájemně propojeny hranou.

Oproti pružinovému algoritmu je respektován Hookův zákon. Pro výpočet síly působící mezi vrcholy, které jsou vzájemně propojeny hranou, využívá vztahu $F = -k \times x$, kde k je tuhost pružiny a x je výchylka pružiny z klidového stavu. Díky tomuto vztahu tak mohou vrcholy, které jsou vzájemně propojeny hranou a jsou velmi blízko u sebe, na sebe působit i odpudivě. Vztah pro výpočet odpudivé síly zůstal stejný jako u pružinového algoritmu. Nově se ale používá pro výpočet odpudivých sil mezi všemi vrcholy navzájem.

Dále již algoritmus Kamada–Kawai pracuje podobně jako pružinový algoritmus. Opět pracuje v cyklu, ve kterém počítá pro každý vrchol grafu sílu, kterou na něj působí všechny okolní vrcholy. Na základě těchto sil se určí nové souřadnice vrcholů. Zároveň se počítá energie celého systému, která se s každou další iterací snižuje. Tento cyklus je ukončen, až rozdíl energií mezi iteracemi poklesne pod stanovenou hranici. I v tomto algoritmu je potřeba modelovat tlumení pohybu vrcholů tak, aby se model nerozkmital. Jedna iterace, ve které se počítá přitažlivá a odpudivá síla, má časovou složitost $O(|V|^2 + |E|)$, kde $|V|$ je počet vrcholů (Vertices) a $|E|$ je počet hran (Edges).

3.1.3 Algoritmus Fruchterman–Reingold

Thomas M. J. Fruchterman a Edward M. Reingold publikovali v roce 1991 dokument „Graph Drawing by Force–directed Placement"[6], ve kterém tento algoritmus popsali. Částečně vychází jak z pružinového algoritmu, tak i z algoritmu Kamada–Kawai. Pro výpočty je využívána tzv. optimální vzdálenost mezi vrcholy.

Vztah pro výpočet přitažlivé síly pružinového algoritmu byl nahrazen výpočetně jednodušším vztahem. Vztah pro výpočet odpudivé síly je stejný jako v algoritmu Kamada–Kawai.

Díky využívání principů simulovaného žíhání je algoritmus odolnější vůči uváznutí v lokálním minimu. Simulované žíhání navíc poskytuje tlumení modelu, které zabraňuje rozkmitání.

Princip metody simulovaného žíhání je založen na simulaci žíhání oceli. Žíhání se v metalurgii provádí k odstranění vnitřních defektů v tělese. Provádí se zahřátím na vysokou (žíhací) teplotu a následným velice pomalým ochlazováním. Zahřátím získávají

atomy tělesa energii, která jim umožní překonávat lokální energetické bariéry a dostat se do rovnovážných poloh. Díky postupnému snižování teploty se rovnovážné polohy atomů stabilizují.

Algoritmus tak může přijmout do další iterace i horší řešení. Tím je zabráněno uváznutí v lokálním minimu. V algoritmu se místo snižování teploty snižuje maximální dovolený posun vrcholu. Každý vrchol je posunut o vypočtenou vzdálenost, nejvýše však o maximálně dovolenou.

Nevýhodou tohoto algoritmu je vysoká výpočetní náročnost. Pro urychlení je často používáno předzpracování grafu jinými, méně náročnými algoritmy. Provádí se tak méně iterací. V jedné iteraci se pro každý vrchol počítá síla odpudivá o složitosti $O(|V|^2)$ a pro každou hranu síla přitažlivá o složitosti $O(|E|)$. Pro každý vrchol se sčítají síly působící na daný vrchol. Podle výsledných velikostí těchto sil jsou souřadnice vrcholů posunuty, nejvýše však o maximální dovolený posun. Složitost této části je $O(|V|)$. V závěru každé iterace je snížen maximální dovolený posun vrcholu (snížení teploty). Výsledná složitost jedné iterace je $O(|V|^2 + |E|)$.

Délka cyklu je obvykle určena délkou posunu (teplotou). Jakmile klesne pod určitou mez, je cyklus ukončen. Například pokud je maximální posun menší než jeden pixel. V tomto případě již další iterace nemají smysl. Lepších výsledků může být dosaženo opětovným spuštěním cyklu algoritmu.

3.1.4 Algoritmus Yifan Hu

Tento algoritmus[9, 10, 11] je efektivní a vysoce kvalitní. Kombinuje více úrovňový přístup, který účinně překonává lokální minima díky technice kvadrantového stromu Barnes–Hut, která aproximuje krátké a dlouhé rozsahy sil. Dále je navrženo adaptivní chladící schéma pro silové algoritmy a obecný model odpudivých sil, který vznikl na základě analýz modelu Fruchterman–Reingold.

4 Práce s grafem pomocí Gephi a C#

Aby mohl být graf vizualizován, musí být známy přesné souřadnice jednotlivých vrcholů grafu. Otázku rozmístění vrcholů grafu (layout grafu) v prostoru řeší celá řada algoritmů pro zobrazování grafů, tzv. layout algoritmů. V programu Gephi[7] je implementováno několik vizualizačních algoritmů včetně algoritmu Yifan Hu. S jehož pomocí můžeme vypočítat layout i rozsáhlým grafům.

4.1 Gephi

Gephi je interaktivní vizualizační platforma pro všechny typy grafů, jako například: síťových, komplexně systémových, dynamických a hierarchických. Grafy mohou být orientované, neorientované i smíšené. Jedná se o software určený pro průzkumnou datovou analýzu, který je open–source a je poskytován zdarma. Podporuje operační systémy Windows, Linux a Mac OS. Je napsán v programovacím jazyce Java. Velikost grafů by neměla překročit 1 milión vrcholů a hran.

S pomocí programu Gephi lze například: vizualizovat v reálném čase, vytvářet layout grafům nebo provádět dynamickou analýzu sítí (Dynamic Network Analysis – DNA). Dále program poskytuje Framework pro statistiku a metriku, který nabízí nejčastější metriky pro analýzu sociálních sítí (Social Network Analysis – SNA). Lze také využít dynamické filtrace v reálném čase na základě struktury grafu nebo dat.

Gephi podporuje následující formáty souborů, ze kterých lze načíst graf: GEFX, GDF, GML, GraphML, Pajek NET, GraphViz DOT, CSV, UCINET DL, Tulip TPL, Netdraw VNA, Spreadsheet.

V programu Gephi jsou naimplementovány tyto vizualizační algoritmy:

- Yifan Hu;
- Yifan Hu Proportional (Yifan Hu úměrný);
- Force Atlas;
- Force Atlas 2;
- Fruchterman Reingold;
- Yifan Hu's Multilevel (víceúrovňový YifanHuv).

4.1.1 Gephi Toolkit

Vývojáři Gephi vyvinuli kromě samotného programu také Gephi Toolkit. Gephi Toolkit obsahuje základní moduly (Graph, Layout, Filtry, IO, . . .). Je to standardní Java knihovna, která může být použita v jakémkoliv Java projektu. Tato knihovna se skládá pouze z jednoho souboru JAR, který může kdokoliv využít v Java aplikacích k dosažení úkolů, které lze provést v Gephi automaticky nebo z příkazové řádky programu. Možnost používat Gephi funkce v jiných Java programech je velice užitečná.

v Gephi Toolkitu jsou zabudovány funkce pro import všech podporovaných souborových formátů a pro export navíc PDF a PNG. Jednou z možností využití je vytvoření layout programu, kde je vstupem graf. Nad grafem je vytvořen layout a výstupem je soubor s grafem, který navíc obsahuje souřadnice vrcholů. Vývojáři Gephi uvádí, že nejlepší pro tuto funkčnost jsou formáty GEXF nebo GDF.

4.1.2 GDF

GDF je souborový formát používaný systémem GUESS[5]. Je postaven jako databázová tabulka nebo soubor, v kterém je čárka používána jako oddělovač. Jsou podporovány atributy vrcholů a hran. Standardní soubor je rozdělen do dvou oddílů, jeden pro vrcholy a druhý pro hrany. Každý oddíl obsahuje záhlaví, ve kterém jsou popsány názvy sloupců. Jednotlivé prvky (tj. vrcholy nebo hrany) jsou reprezentovány jedním řádkem, ve kterém jsou hodnoty atributů odděleny čárkou. Formát GDF je proto velmi snadno čitelný.

Ve výpisu souboru 1 je vidět minimální GDF soubor, který podporuje Gephi importér souborů. Popis vrcholů je definován prvním řádkem (záhlaví) prvního oddílu souboru. Název vrcholu představuje první sloupec a popisek vrcholu představuje sloupec druhý. Po definici vrcholů následuje druhý oddíl. Popis hran je definován prvním řádkem (záhlaví) druhého oddílu. Vrcholy, které jsou spojeny danou hranou, jsou uvedeny v prvním a druhém sloupci.

```
nodedef>name VARCHAR,label VARCHAR
s1,Site  number 1
s2,Site  number 2
s3,Site  number 3
edgedef>node1 VARCHAR,node2 VARCHAR
s1,s2
s2,s3
s3,s2
s3,s1
```

Výpis 1: GDF soubor – minimální

Ve výpisu souboru 2 je ukázka, jak může vypadat soubor s přidanými atributy. Je zde navíc barva vrcholu a jeho velikost. Za povšimnutí stojí definice obarvení jednotlivých vrcholů. U hran jsou přidány atributy označující tloušťku, orientovanost a barvu. Číselné atributy jsou zadávány v jednotkách DOUBLE.

```
nodedef>name VARCHAR,label VARCHAR,color VARCHAR, width DOUBLE
26,26,'128,128,128',10
4,4,'128,128,128',10
98,98,'128,128,128',10
179,179,'128,128,128',10
229,229,'128,128,128',10
edgedef>node1 VARCHAR,node2 VARCHAR,weight DOUBLE,directed BOOLEAN,color
    VARCHAR
26,5,1,False ,'0,0,0'
26,66,1,False ,'0,0,0'
```

[5]Systém pro zkoumání grafů – `http://graphexploration.cond.org/`

```
26,144,1,False ,'0,0,0'
26,161,1,False ,'0,0,0'
26,67,1.4142135624,False,'0,0,0'
4,408,2,False ,'0,0,0'
4,68,2.4142135624,False,'0,0,0'
```

Výpis 2: GDF soubor – rozšířený

4.2 IKVM.NET

IKVM.NET[5] je implementací Javy pro Mono[6] a Microsoft .NET Framework[7]. Obsahuje následující komponenty:

- Virtuální stroj Javy implementovaný v .NET.
- .NET implementaci Java tříd a knihoven.
- Nástroje umožňující interoperabilitu[8] Javy a .NET.

Mezi možné využití IKVM patří:

- **Drop–in JVM** – (Spuštění aplikace ve virtuálním stroji Java) – tato ikvm aplikace obsažená v distribuci IKVM.NET je .NET implementací virtuálního stroje Java. Ke spuštění aplikace stačí napsat `ikvm -jar myapp.jar`
- **Použití Java knihoven v aplikacích .NET** – V distribuci IKVM.NET je obsažena aplikace ikvmc (IKVM Kompilátor). Touto aplikací je realizována funkce Java bytecode překladače.
- **Vývoj .NET aplikací v Javě** – Pomocí výše uvedené aplikace lze nejen převádět jednotlivé Java knihovny do prostředí .NET, ale i celé Java aplikace.

4.2.1 IKVM compiler

IKVMC slouží pro překlad Java bytecode do .NET IL. Pokud máme Java knihovnu, kterou bychom chtěli použít v .NET aplikaci, spustíme příkaz `ikvmc -target:library mylib.jar` k vytvoření mylib.dll. Typ výstupního souboru je určen pomocí parametru „target“:

- **exe** – generuje spustitelný soubor v příkazové řádce;
- **winexe** – generuje .exe pro GUI aplikace;
- **library** – generuje .dll;

[6] Multiplatformní open–source .NET vývojový Framework – `http://www.mono-project.com/Main_Page`

[7] `http://msdn.microsoft.com/cs-cz/netframework/`

[8] interoperabilita — schopnost systémů vzájemně si poskytovat služby a efektivně spolupracovat

- **module** – generuje .netmodule.

Takto vytvořený soubor můžeme použít v .NET aplikaci přidáním reference na něj. Zároveň musíme přidat knihovny programu IKVM a v referencích se odkázat na jeho jádro *IKVM.OpenJDK.Core.dll*.

4.3 Export a import struktury grafu

Import struktury grafu je v Gephi Toolkit řešen třídou ImportController, kterou je načten vybraný soubor. Formát souboru je automaticky rozpoznán. Dále je graf přiřazen k zvolenému pracovnímu prostoru (workspace), kde na něm lze provádět požadované úkony, například vytvořit layout. Příklad importu provedený Gephi Toolkitem v prostředí .NET, volaný pomocí IKVM, je vidět ve výpisu 3.

```
// Inicializace projektu a workspace
ProjectController pc = (ProjectController)Lookup.getDefault().lookup(new ProjectControllerImpl().
    getClass());
pc.newProject();
Workspace workspace = pc.getCurrentWorkspace();
ImportController importController = (ImportController)Lookup.getDefault().lookup(new
    ImportControllerImpl().getClass());

//Import souboru/grafu
org.gephi.io.importer.api.Container container = ((ImportContainerFactoryImpl)Lookup.getDefault().
    lookup(new ImportContainerFactoryImpl().getClass())).newContainer();
try
{
    File file = new File(FileName);
    container = importController.importFile(file);
    container.getLoader().setEdgeDefault(EdgeDefault.UNDIRECTED);
}
catch (java.lang.Exception ex)
{
    throw ex;
}
finally
{
    container.closeLoader();
}

// Připojení importovaných dat k workspace
importController.process(container, new DefaultProcessor(), workspace);
```

Výpis 3: Import grafu

Výpočtem layoutu jsou přiřazeny jednotlivým vrcholům grafu konkrétní souřadnice a hranám mezi body jejich ohodnocení. Takto zpracovaný graf může být dále exportován. Export je řízen třídou ExportController. Bez další specifikace je graf exportován ve stejném formátu, jako byl formát vstupní. Pokud ve vstupním souboru byly u vrcholů definovány atributy *x, y* a u hran *weight* (její ohodnocení), tak se při exportu tyto atributy aktualizují. Pokud ovšem definovány nebyly, tak se při exportu potřebné sloupce

přidají. Díky vlastnostem formátu GDF, případně GEFX, není problém přidávat jakékoliv atributy jak k vrcholům, tak ke hranám. Typ výstupu lze ovšem konkrétně specifikovat. Příklad exportu provedený Gephi Toolkitem v prostředí .NET, volaný pomocí IKVM, je vidět ve výpisu 4, kde je ukázán jak defaultní export, tak export do formátu PNG i PDF. U každého typu výstupu lze definovat mnoho volitelných atributů, například: šířku, výšku, okraj, velikost stránky, průhlednost, apod.

```
// Export
ExportController ec = (ExportController)Lookup.getDefault().lookup(new ExportControllerImpl().
    getClass());

PNGExporter pngExporter = (PNGExporter)ec.getExporter("png");
pngExporter.setHeight(4000);
pngExporter.setWidth(4000);
pngExporter.setTransparentBackground(false);
pngExporter.setWorkspace(workspace);

PDFExporter pdfExporter = (PDFExporter)ec.getExporter("pdf");
pdfExporter.setPageSize(com.itextpdf.text.PageSize.A0);
pdfExporter.setWorkspace(workspace);

try
{
    ec.exportFile(new File(FileName + "YifanHuProportional_vystupni.gdf"));
    ec.exportFile(new File(FileName + "YifanHuProportional.png"), pngExporter);
    ec.exportFile(new File(FileName + "YifanHuProportional.pdf"), pdfExporter);
}
catch (IOException ex)
{
    throw ex;
}
```

Výpis 4: Export grafu

Na vestavěných třídách řídících import a export v Gephi Toolkitu však nemusíme být závislí. Je zde další třída GraphModel, ve které jsou obsaženy veškerá data o grafu, jeho vlastnostech a podrobná data o každém vrcholu či hraně a jejich atributech. Na základě všech těchto údajů, získaných z třídy GraphModel, lze sestavit vlastní algoritmus pro export grafu do souboru, v požadované podobě a struktuře. Například v souborovém formátu JSON, který lze dále využít pro vizualizaci v internetovém prohlížeči

5 Využití JavaScriptu a SVG pro vizualizaci

V této kapitole je popsán souborový formát JSON a dále Framework určený pro práci s tímto souborovým formátem. Následují informace o vyvíjeném standardu HTML5 a o vektorové grafice SVG[9], určené pro vizualizaci v internetovém prohlížeči. Jak lze pomocí JavaScriptu a knihovny D3.JS vytvářet SVG grafiku s propojením na data ve formátu JSON je popsáno v odstavci 5.4.

5.1 Json

JSON neboli JavaScript Object Notation je v překladu JavaScriptový objektový zápis. Jedná se o formát určený pro výměnu dat. Tento formát se vyznačuje tím, že je pro člověka jednoduše čitelný i zapisovatelný a strojově snadno zpracovatelný i generovatelný. Formát JSON je odvozen ze skriptovacího jazyka JavaScript pro reprezentaci jednoduchých datových struktur, polí a objektů. I přes jeho vztah k JavaScriptu je JSON textový, na jazyce zcela nezávislý formát. Díky tomu je pro výměnu dat opravdu ideálním jazykem.

Do souboru formátu JSON můžeme ukládat následující objekty: textový řetězec, číslo, logickou hodnotu, hodnotu null a pole, které je složeno z dalších objektů. Složitost a hierarchie skládání a vnořování objektů není nijak omezena. JSON je založen na dvou strukturách:

- kolekce párů název/hodnota a
- uspořádaný seznam hodnot (pole).

Tyto struktury jsou realizovány pomocí následujících konstrukcí:

- Objekt je množina párů název/hodnota. Objekt je ohraničen znaky { (levá složená závorka) a } (pravá složená závorka). Každý název je následován znakem : (dvojtečka). Jednotlivé páry název/hodnota jsou odděleny znakem , (čárka).
- Pole je kolekcí hodnot. Pole je ohraničeno znaky [(levá hranatá závorka) a] (pravá hranatá závorka). Hodnoty pole jsou odděleny znakem , (čárka).
- Hodnotou rozumíme: řetězec, číslo, logickou hodnotu **`true`** nebo **`false`**, nepřiřazenou hodnotu **`null`**, objekt nebo pole. Tyto struktury mohou být libovolně vnořovány.
- Řetězcem je libovolný (i nulový) počet znaků kódování Unicode, které jsou uzavřeny do dvojitých uvozovek a využívají tzv. únikových sekvencí (escape sequence) s použitím zpětného lomítka.
- U čísel není používán oktalový ani hexadecimální zápis.

[9] `http://www.w3.org/TR/SVG/`

objekt	{ } { členové }
členové	pár pár , členové
pár	řetězec : hodnota
pole	[ ] [elementy]
elementy	hodnota hodnota , elementy
hodnota	řetězec číslo objekt pole **`true`** (pravda) **`false`** (nepravda) **`null`** (nepřiřazená hodnota)

Tabulka 1: Definice objektů formátu JSON

V tabulce č. 1 je vidět skladba jednotlivých prvků a objektů formátu JSON. Ve výpisech číslo 5 a 6 je vidět ukázka grafu ve formátu GDF a jemu odpovídající graf ve formátu JSON.

U formátu souboru GDF je struktura vrcholů i hran definována prvním řádkem (záhlavím oddílu) příslušného oddílu. Vrcholy či hrany jsou popsány každým dalším řádkem. Zápis parametrů, jejich pořadí a tvar, je definován právě záhlavím. Oproti tomu u formátu JSON jsou objekty a jeho hodnoty definovány dle již zmíněného formátu. Celý dokument je tvořen jedním objektem obsahujícím dva páry název/hodnota. Vrcholy jsou zde uloženy jako hodnota prvku s názvem *nodes* (vrcholy) ve formě pole. Stejně je tomu tak i v případě hran, které jsou hodnotou prvku s názvem *edges* (hrany). V jednotlivých polích jsou obsaženy objekty představující vrcholy nebo hrany. Atributy konkrétních vrcholů i hran jsou ve formátu JSON definovány pro každý prvek zvlášť. Nezáleží na pořadí atributů. Pomocí páru název/hodnota je určen název atributu a jeho hodnota.

Způsob vytvoření grafu ve formátu JSON zpracováním grafu ve formátu GDF je podrobněji popsán v odstavci 7.3.1 na straně 40.

```
nodedef>name VARCHAR,label VARCHAR,color VARCHAR, width DOUBLE
26,26,'128,128,128',10
4,4,'128,128,128',10
98,98,'128,128,128',10
179,179,'128,128,128',10
229,229,'128,128,128',10
261,261,'128,128,128',10
edgedef>node1 VARCHAR,node2 VARCHAR,weight DOUBLE,directed BOOLEAN,color
    VARCHAR
26,5,1,False ,'0,0,0'
26,66,1,False ,'0,0,0'
```

```
26,128,1,False ,'0,0,0'
26,27,1,False ,'0,0,0'
26,31,1,False ,'0,0,0'
26,125,1,False ,'0,0,0'
26,144,1,False ,'0,0,0'
26,161,1,False ,'0,0,0'
26,204,1,False ,'0,0,0'
26,289,1,False ,'0,0,0'
26,300,1,False ,'0,0,0'
26,67,1.4142135624,False,'0,0,0'
```

Výpis 5: Ukázka grafu ve formátu GDF

```
{
  "nodes" : [
    { "name" : "26" , "group" : -1 , "x" : 275.077728 , "y" : 556.015137 },
    { "name" : "4" , "group" : -1 , "x" : 623.3768 , "y" : 843.255432 },
    { "name" : "98" , "group" : -1 , "x" : 744.041443 , "y" : 588.1283 },
    { "name" : "179" , "group" : -1 , "x" : 722.3707 , "y" : 588.4702 },
    { "name" : "229" , "group" : -1 , "x" : 712.802856 , "y" : 574.7805 },
    { "name" : "261" , "group" : -1 , "x" : 618.7286 , "y" : 696.7201 }
    ],
  "links" : [
    { "source" : 2 , "target" : 1 , "value" : 1 },
    { "source" : 3 , "target" : 1 , "value" : 1 },
    { "source" : 3 , "target" : 2 , "value" : 5 },
    { "source" : 4 , "target" : 1 , "value" : 1 },
    { "source" : 4 , "target" : 2 , "value" : 7 },
    { "source" : 4 , "target" : 3 , "value" : 11 },
    { "source" : 5 , "target" : 1 , "value" : 1 },
    { "source" : 5 , "target" : 2 , "value" : 2 },
    { "source" : 5 , "target" : 3 , "value" : 1 },
    { "source" : 5 , "target" : 4 , "value" : 1 },
    { "source" : 6 , "target" : 1 , "value" : 1 },
    { "source" : 6 , "target" : 2 , "value" : 6 }
    ]
}
```

Výpis 6: Ukázka grafu ve formátu JSON

5.1.1 Newtonsoft.Json

James Newton, student z Nového Zélandu, vytvořil vysoce výkonný JSON Framework pro .NET nazvaný JSON.NET. Základní služby Frameworku, pro práci s formátem JSON, jsou implementovány v třídách, které jsou poskytovány jmenným prostorem Newtonsoft.Json. Výčet funkcí a vlastností tohoto Frameworku:

- flexibilní JSON serializér pro konvertování objektů mezi .NET a JSON;
- LINQ v JSONu – slouží pro manuální čtení a zápis JSONu;
- vysoká výkonnost – větší než JSON serializér obsažený v .NET;

.NET JSON – Srovnání výkonu serializace		
	Serializace	Deserializace
Json.NET 4.0	84ms	218ms
DataContractJsonSerializer	1669ms	231ms
JavaScriptSerializer	292ms	809ms

Zdroj: http://james.newtonking.com/pages/json-net.aspx

Tabulka 2: .NET JSON – Srovnání výkonu serializace

- členité zapisování – snadno čitelný JSON;
- převod JSON na XML a zpět;
- podpora .NET 2, .NET 3.5, .NET 4, Silverlight a Windows Phone.

Ve výpisu 7 je vidět ukázka použití serializace a deserializace a ve výpisu 8 je vidět ukázka použití LINQ v JSONu. V tabulce 2 je uvedeno srovnání výkonu serializace a deserializace.

```
Produkt produkt = new Produkt();
produkt.Nazev = "Jablko";
produkt.Platnost = new DateTime(2008, 12, 28);
produkt.Cena = 3.99M;
produkt.Velikosti = new string[] { "Malý", "Střední", "Veliký" };

string json = JsonConvert.SerializeObject(produkt);
//{
//  "Nazev": "Jablko",
//  "Platnost": new Date(1230422400000),
//  "Cena": 3.99,
//  "Velikosti": [
//    "Malý",
//    "Střední",
//    "Veliký"
//  ]
//}

Produkt deserializovanyProdukt = JsonConvert.DeserializeObject<Produkt>(json);
```

Výpis 7: Ukázka použití JSON serializace a deserializace

```
string json = @"{
  ""Nazev"":""Jablko"",
  ""Platnost"": new Date(1230422400000),
  ""Cena"":3.99,
  "" Velikosti "": [
    ""Malý"",
    ""Střední"",
    ""Veliký""
  ]
}";
```

```
JObject o = JObject.Parse(json);

string nazev = (string)o["Název"];
// Jablko

JArray velikosti = (JArray)o[" Velikosti " ];

string nejmensi = (string) velikosti [0];
// Malý
```

Výpis 8: Ukázka použití LINQ v JSON

Json.NET knihovnou jsou poskytovány třídy umožňující jednoduché a bezpečné čtení a zápis objektů JSON z prostředí .NET. Používání tříd pro zápis a čtení (JsonReader, JsonTextReader, JsonWriter, JsonTextWriter), stejně jako v .NET prostředí, je známo většině .NET vývojářů.

Použitím třídy *JsonTextWriter* lze postupným zapisováním jednotlivých objektů JSON vytvořit soubor formátu JSON. Použitím třídy *JsonTextReader* je umožněno ze souboru číst jednotlivé JSON objekty. Práce s těmito třídami je velice podobná práci s třídami *XmlReader* a *XmlWriter*.

5.2 HTML5

HTML5 je pátou revizí HTML standardu (HTML standard byl vytvořen v roce 1990 a standardizován jako HTML4 v roce 1997), která je stále ve vývoji (duben 2012) organizací W3C. HTML5 je jazyk pro strukturování a prezentování obsahu na internetu. Hlavním cílem vývojářů standardu je vylepšení jazyka o podporu nejnovějších multimédií při zachování snadné čitelnosti člověkem a správnému zpracování počítači a zařízeními. V HTML5 je zahrnuto kromě HTML4 také XHTML 1 a DOM Level 2 HTML.

HTML5 přináší řadu nových funkcí. Ty byly navrženy tak, aby bylo snadné vkládat a zpracovávat multimediální a grafický obsah na internet. A to bez potřeby využívat dodatečných modulů plug–in a rozhraní API. Jednou z nových funkcí je integrace vektorové grafiky *Scalable Vector Graphics* (SVG). HTML5 nejsou poskytovány animace v rámci internetových stránek. Pro animaci HTML či SVG objektů je nezbytné použití JavaScriptu nebo CSS3.

5.3 SVG

Škálovatelná vektorová grafika (Scalable Vector Graphics – SVG) je značkovací jazyk popisující dvojrozměrnou vektorovou grafiku pomocí XML. A to jak statickou, tak i dynamickou (například interaktivní nebo animovanou). Specifikace SVG je otevřený standard vyvíjený organizací W3C od roku 1999.

SVG obrázky a jejich chování jsou definovány v souborech XML. To znamená, že je možné je vyhledávat, indexovat, skriptovat a v případě potřeby komprimovat. Stejně jako XML soubory mohou být SVG obrázky vytvářeny a upravovány jakýmkoliv textovým editorem.

Formát SVG je podporován všemi moderními internetovými prohlížeči, například Mozilla Firefox, Internet Explorer 9, Google Chrome, Opera a Safari. Dřívějšími verzemi prohlížeče Microsoft Internet Explorer není SVG podporováno. Podpora grafiky SVG představuje výkonný prostředek umožňující přidávat na internet jednoduché škálovatelné vizuální prvky vyznačující se vysokou kvalitou, od malých a jednoduchých až po velké a komplexní, bez nutnosti použití modulu plug–in nebo samostatného prohlížeče.

SVG jsou povolovány tři typy grafických objektů:

- vektorová grafika;
- rastrová grafika;
- text.

Grafické objekty mohou být například seskupovány, formátovány (pomocí atributů nebo stylů CSS) nebo polohovány pomocí transformací. SVG je také podporováno ořezávání objektů, alfa maskování, filtrování obrazu a animace. V SVG je obsažena bohatá množina událostí, které mohou být volány různými skripty (obvykle JavaScriptem), kterými lze manipulovat s grafikou v DOM od SVG.

Základní tvary grafiky SVG:

- rect – obdélník;
- circle – kruh;
- ellipse – elipsa;
- line – čára;
- polyline – řetězec linií;
- polygone.

Ukázka zápisu jednotlivých objektů je na výpisu 9, kde je navíc u elementu text ukázka transformace. Stylování jednotlivých objektů a jejich atributů (například fill – barva výplně, stroke – barva ohraničení nebo čáry, stroke–width – šířka ohraničení nebo čáry) můžeme specifikovat přímo v definici objektu nebo v kaskádovém stylu CSS, který lze propojit s objektem jeho atributem „style“.

```
<rect width="150" height="50" fill ="red" />
<circle cx="50" cy="50" r="50" fill ="red" />
<ellipse cx="100" cy="50" rx="100" ry="50" fill ="red" />
<line x1="0" y1="0" x2="200" y2="50" style="stroke:red;stroke-width:2"/>
<polyline points="0,0 0,20 20,20 20,40 40,40 40,60" fill="red" />
<polygon points="20,10 170,20, 80,50" fill ="red" />
<text x="0" y="15" fill ="red" transform="rotate(30 20,40)">Mám rád SVG</text>
```

Výpis 9: Ukázka definování objektů SVG

Grafikou SVG lze jednoduše provádět vizualizace nejrůznějších objektů a tvarů včetně jejich animací.

5.3.1 Události a JavaScript

Abychom mohli objekty rozpohybovat a udělat s nimi požadované transformace či animace, je vhodné pro tento účel použít JavaScript. Většina uživatelů internetu má podporu JavaScriptu zapnutou. SVG ovšem není nijak limitováno na použití konkrétního skriptovacího jazyka.

JavaScriptové funkce mohou být přiřazeny událostem SVG objektů. Funkce je zaregistrována jako posluchač dané události. U každé události je určeno, kdy nastane (například: při kliknutí na objekt – událost *onclick*, při najetí myší na objekt – událost *onmouseover*, apod.). Když událost nastane, spustí se přiřazené skripty. Pro účel transformací a animací jsou definovány různé události u všech animačních prvků, kontejnerových objektů a u většiny grafických objektů. Pomocí skriptů lze měnit hierarchická struktura objektů v dokumentu a atributy, vlastnosti a stylování objektů. Změny mohou být definovány jako přechod, mezi starými a novými hodnotami, s konkrétní délkou trvání, čímž je vytvořen efekt animace.

5.4 Vizualizace SVG – D3.JS

D3.JS[10] je malá JavaScriptová knihovna pro manipulaci s dokumenty založených na datech a je poskytována zdarma. Pomocí D3 je umožněno navázat libovolná data na DOM a pak použít datově řízené transformace na dokumentu. Například můžeme D3 použít pro generování základní HTML tabulky z pole čísel nebo ze stejných dat vytvořit interaktivní SVG sloupcový graf.

D3 není tradiční vizualizační Framework. Spíše než poskytování monolitického systému se všemi funkcemi, které by někdo mohl potřebovat, je D3 řešeno pouze jádro problému: efektivní manipulace s dokumenty na základě dat. To dává D3 mimořádnou flexibilitu za plné podpory souvisejících technologií, jako jsou CSS3, HTML5 a SVG. D3 je velmi rychlé s minimální režií, s podporou velkých datových souborů a dynamického chování pro interakci a animace.

5.4.1 Práce s D3.JS

Dále je popsáno několik základních funkcí a vlastností, které jsou v knihovně D3 implementovány. Prací s D3 je myšleno využívání těchto funkcí či vlastností při vytváření vlastních JavaScriptových funkcí. V podstatě se jedná o:

- Vytváření, mazání a výběr prvků v dokumentu.
- Nastavení nebo změnu vlastností vybraných prvků.
- Transformace a animace.

Výše uvedené operace lze provádět jak staticky, tak dynamicky díky možnosti propojení dat s dokumentem.

[10] `http://www.d3js.org/`

Selekce Úprava dokumentů pomocí nativního W3C DOM API je zdlouhavá. Složitost funkcí vyžaduje ruční iterace a uchovávání dočasného stavu. Chcete-li například změnit barvu textu v elementu *paragraph* (výpis 10).

Místo manipulace s jednotlivými prvky je v D3 použito alternativního přístupu fungujícím na libovolných sadách prvků nazývaných selekce (výběr). Selekce se samozřejmě může skládat s pouze jednoho prvku. Příklad použití je ve výpisu 11. Prvky mohou být vybírány různými predikáty, například na základě obsahu, hodnot atributů, přiřazených tříd nebo ID.

```
var paragraphs = document.getElementsByTagName("p");
for (var i = 0; i < paragraphs.length; i++) {
  var paragraph = paragraphs.item(i);
  paragraph.style.setProperty("color", "white", null);
}
```

Výpis 10: Použití W3C DOM API

```
d3.selectAll("p")
  .style("color", "white");

d3.select("body")
  .style("background-color", "black");
```

Výpis 11: Použití D3.js – selekce

Knihovnou D3 je tedy poskytováno standardní zázemí pro úpravu prvků: nastavování atributů a stylů, registrace posluchačů a událostí, přidávání, odebírání a třídění prvků a změna HTML nebo textového obsahu. S tímto si lze vystačit pro většinu potřeb. Pokud však použití základního DOM API je nezbytně nutné, můžeme také pomocí D3 k prvkům přistupovat přímo, protože každá selekce vytvořená knihovnou D3 je prostě pole prvků.

Dynamické vlastnosti Styly, atributy a jiné vlastnosti mohou být v D3 zadány pomocí funkcí na základě dat, ne jenom jako jednoduché konstanty. Příklad použití: nastavení náhodných barev prvků, využití indexu prvků (*i*) jako druhého parametru funkce. Lze i vytvořit vazbu mezi daty a prvky v selekci a využít data při výčtu vlastností. Data jsou uvedeny jako pole libovolných hodnot a každá hodnota se předá jako první argument (*d*) do proměnné funkce. První prvek v datovém poli je předán do prvního prvku selekce, druhý prvek do druhého prvku, a tak dále. Jakmile jsou data svázána s dokumentem, lze vynechat datový operátor (*data*). Jsou-li data již jednou přiřazena k selekci, tak jsou D3 automaticky načítána. Ukázka využití dynamicky definovaných vlastností je ve výpisu 12.

```
d3.selectAll("p")
  .style("color", function() {
    return "hsl(" + Math.random() * 360 + ",100%,50%)";
  });

d3.selectAll("p")
  .style("color", function(d, i) {
```

```
      return i % 2 ? "#fff" : "#eee";
    });

d3.selectAll("p")
  .data([4, 8, 15, 16, 23, 42])
  .style("font-size", function(d) { return d + "px"; });
```

Výpis 12: Použití D3.js – dynamické vlastnosti

Vstup a výstup Pomocí D3 může být snadno manipulováno s již existujícími prvky. V případě, že prvky ještě neexistují, chceme je vytvořit. Jestliže jich existuje více, než prvků v datovém poli, chceme odstranit přebytečné prvky. Řešením je použití selekcí pro vstup a výstup (enter a exit), díky kterým můžeme přidávat nové prvky, které by odpovídaly datům, a odstranit prvky, které již nejsou potřeba.

Když jsou data svázána se selekcí prvků, tak je každý prvek v datovém poli spárován s odpovídajícím prvkem v selekci. Je-li méně prvků než dat, lze vytvořit další prvky ve vstupní selekci, kde lze konkrétně definovat, jaký prvek má být vytvořen. Vstupním operátorem je vzato jméno prvku a ten je připojen k dokumentu. Ukázka je ve výpisu 13.

```
d3.select("body").selectAll("p")
  .data([4, 8, 15, 16, 23, 42])
  .enter().append("p")
  .text(function(d) { return "Jsem číslo " + d + "!"; });
```

Výpis 13: Použití D3.js – přidávání prvků

V základě lze práci rozdělit do tří kategorií: aktualizace, přidávání prvků a odebírání prvků. Ukázka je ve výpisu 14.

```
// Aktualizace...
var p = d3.select("body").selectAll("p")
  .data([4, 8, 15, 16, 23, 42])
  .text(String);
// Přidání...
p.enter().append("p")
  .text(String);
// Odebrání...
p.exit().remove();
```

Výpis 14: Použití D3.js – aktualizace, přidávání a odebírání prvků

Využitím těchto tří případů samostatně lze provádět nezbytné úpravy na každé sadě prvků. To je zvláště užitečné při určování transformací. Například: sloupcový graf je svázán se starými daty, po změně údajů aktualizujeme na data nová, poté provedeme odebrání přebývajících prvků a dále přidání chybějících. Aktualizace je výchozím nastavením. Pokud není uvedeno, zda se jedná o vstup nebo výstup, budou se automaticky vybírat pouze prvky, pro něž existují odpovídající údaje.

Funkce D3 umožňují transformovat existující dokumenty (hierarchii prvků) na základě dat. Toto zobecnění zahrnuje vytváření nových dokumentů, kde je výchozí selekce

prázdný prvek. Provádění změn a animací existujícího dokumentu je umožněny na základě interakcí uživatele.

Transformace Knihovnou D3 není poskytována nová grafická reprezentace. Není zde žádný nový slovník značek, které by bylo potřeba se naučit. Místo toho je stavěno přímo na standardech jako CSS3, HTML5 a SVG. Tento přístup nabízí řadu výhod. Jednou z nich je plný přístup k základním funkcím prohlížeče. Například lze pomocí D3 vytvářet prvky a poté je stylovat externím CSS.

Podíváme-li se na kruh. Obvykle je definován pomocí operátoru elipsy, který má atributy x a y (souřadnice středu elipsy) a šířku s výškou (délky os), případně pomocí operátoru kružnice s atributy x, y (souřadnice středu kružnice) a rádiusu. Oproti tomu v D3 kolo tzv. nevynalézáme, ale používáme standardní kruhový prvek SVG jako ve výpisu 15. Jak lze vidět, D3 není specifikována konkrétní reprezentace kruhu. Můžeme definovat alternativní formy, které mohou nabízet lepší výkon a kompatibilitu jako čisté HTML.

```
svg.append("circle")
  .attr("cx", 50)
  .attr("cy", 40)
  .attr("r", 10);

body.append("div")
  .style("position", "absolute")
  .style("left", "40px")
  .style("top", "30px")
  .style("width", "20px")
  .style("height", "20px")
  .style("border-radius", "10px")
  .style("background-color", "#000");
```

Výpis 15: Použití D3.js – SVG vs. HTML

Přechody Vzhledem k zaměření D3 na manipulaci s dokumenty na základě dat, nejenom na jednorázovém mapování dat do statické reprezentace, obsahuje D3 podporu pro tzv. hladké přechody (transformace). Jedná se o postupné interpolace stylů nebo atributů v čase. V D3 je implementována interpolace čísel a čísel skrytých v řetězcích (velikost písma, data, atd.). Můžeme zadat vlastní interpolaci k rozšíření přechodů pro složitější vlastnosti.

Nejjednodušší přechody jsou vytvářeny přes operátor přechodu (transition). Ve výpisu 16 je ukázán příklad změny pozadí stránky na černou.

```
d3.select("body").transition()
  .style("background-color", "black");
```

Výpis 16: Použití D3.js – transformce

Také lze použít pro přechody kaskádové styly CSS3. Složitější změna velikosti kruhu může být vyjádřena jednoduše jako ve výpisu 17.

```
d3.selectAll("circle").transition()
  .duration(750)
  .delay(function(d, i) { return i * 10; })
  .attr("r", function(d) { return Math.sqrt(d * scale); });
```

Výpis 17: Použití D3.js – transformce s animací

Trvání přechodu (transition duration) a zpoždění (delay) lze přizpůsobit, stejně tak i jiné vlastnosti, které jsou specifikovány jako funkce dat. To je vhodné například pro animace, kde je spouštěno odstupňované zpoždění na základě indexu (*i*) tak, že je pro diváka snazší sledovat přechody jednotlivých elementů.

Ovlivňovány jsou skutečně pouze ty atributy, které jsou měněny v průběhu přechodu. Tím je v D3 eliminována režie, čímž je umožněna větší složitost a vyšší grafické snímkování. Na konci provádění přechodu je odeslána událost, díky které jsou umožněny sekvenční přechody.

Subselekce Většina dokumentů má hierarchickou strukturu. Kombinací funkcí pro selekci jsou vytvořeny subselekce (podvýběry). Například pokud bychom chtěli vybrat všechny seznamy a v nich všechny položky. Tento příklad je ve výpisu 18.

```
d3.selectAll("ul")
  .selectAll("li")
  .text(function(d, i) { return "Jsem číslo " + i + "!"; });

d3.selectAll("ul")
  .data(otazky) // pole otazek
  .selectAll("li")
  .data(function(d) { return d.odpovedi; }) // vnořené pole odpovědí
  .text(function(d) { return d.text; }); // text odpovědi
```

Výpis 18: Použití D3.js – subselekce

Podvýběr je seskupen podle prvního výběru. Index (*i*) pro položky seznamu (li) odpovídá jejich indexu uvnitř seznamu, ne indexu v celém dokumentu. Seskupováním podvýběrů je D3 umožněno zachovat hierarchickou strukturu dokumentu.

Když jsou data svázána s výběrem, lze data předat jako argument funkce. Ve spojení se vstupními a výstupními operátory lze pomocí D3 sestavovat a aktualizovat složitou hierarchickou strukturu dokumentu za použití minimálního množství kódu.

SVG objekty

Jednoduchá ukázka vytvoření SVG objektu (kruhu) včetně nastavení atributů je ve výpisu 15 na straně 33. Stylování jednotlivých objektů a jejich atributů můžeme specifikovat přímo nebo v kaskádovém stylu CSS, který lze s objektem propojit jeho atributem „style“. Ve výpisu 19 je ukázka vytvoření SVG objektů, konkrétně čar, s napojením na data a nastavením atributů. Dále je v ukázce nastaveno stylování přímé i pomocí stylů CSS. Jak lze pomocí D3 vytvářet objekty vektorové grafiky SVG s napojením na datový soubor formátu JSON je uvedeno v odstavci 7.3.2 na straně 42.

```
// CSS styl
line.link {
 stroke-opacity: .6;
}
// CSS styl

svg.selectAll(".link")
   .data(hrany)
   .enter().append("line")
   .attr("class", "link")
   .style("stroke-width", function (d) { return Math.sqrt(d.value); })
   .style("stroke", "#999")
   .attr("x1", function (d) { return d.source.x; })
   .attr("y1", function (d) { return d.source.y; })
   .attr("x2", function (d) { return d.target.x; })
   .attr("y2", function (d) { return d.target.y; });
```

Výpis 19: Použití D3.js – stylování

6 Slovní vyhledávání pomocí Google API

V následujících odstavcích je popsána webová služba Google API a způsob, jakým lze pomocí této služby získat výsledky slovního vyhledávání ve vlastních aplikacích.

6.1 Webové služby

Webovými službami je umožněno předávání parametrů a volání funkcí vzdáleného serveru protokolem HTTP. Vzdáleným serverem jsou poté zaslány požadované výsledky, se kterými je možno dále pracovat. Nezáleží na tom, v jakém jazyce byla webová služba napsána nebo z jakého jazyka ji voláme. Důležitý je formát výměny dat. Existují dva druhy webových služeb: SOAP a RESTful. REST je architektura umožňující přistupovat k datům na určitém místě pomocí standardních metod HTTP. Výsledky jsou vráceny ve formátu XML, JSON nebo ATOM/RSS.

6.2 Gogle API

Google API je RESTful webová služba. Ve svých programech ji můžeme využít pro získání výsledků vyhledávání. Ty můžeme dále zpracovávat. Nejdříve je sestaven dotaz. Volitelnými parametry jsou: hledaný výraz, jazyk a pozice, od které má být vyhledáváno. Sestavený dotaz je zaslán webové službě Google API. Následně jsou vráceny výsledky ve formátu JSON.

6.3 Stahování výsledků vyhledávání

Cílem je vytvoření aplikace, která by na zadaný dotaz získala výsledky pomocí Google API a dále stáhla celý zdrojový kód pro každý nalezený odkaz. O každém odkazu jsou získány následující údaje:

- nadpis – title,
- odkaz – unescaped url,
- odkaz na Google Cache – cached url,
- popis – content a
- pozice.

Pokud se jedná o pdf soubor, nelze stáhnout jeho zdrojový kód. Google ovšem převádí pdf soubory na html stránky, které ukládá do Google Cache. Pomocí Google Cache lze získat zdrojový kód pdf souboru. Načítané kódy jsou aplikací omezeny maximální velikostí 2MB. Google taktéž do Cache neukládá kódy větší než 2MB.

Zdrojové kódy byly načítány ze získaných odkazů, pro pdf soubory z Google Cache. Jeden stahovací cyklus vrací výsledek obsahující 8 odkazů. Pro urychlení stahování bylo použito paralelizace pomocí funkce *Parallel.For*. Touto funkcí je stahování spuštěno ve více paralelních vláknech.

Komunikace s Google API je prováděna v kódování UTF-8. Problémem je, že zdrojové kódy různých stránek mohou mít jiné kódování. Kódování je určeno znakovou sadou, kterou se nepodaří vždy úspěšně načíst. Řešením je načtení celého zdrojového kódu. Nyní lze správně určit kódování. Pokud je jiné než UTF-8, je zdrojový kód stažen znovu se správným kódováním. Pro omezení dvojitého stahování jsou zdrojové kódy dočasně ukládány na straně klienta. Ze získaného zdrojového kódu je nakonec získán čistý obsah stránky.

Omezením využívání Google API a Google Cache je, že se v neplacené verzi po určitém počtu stáhnutí výsledků služby zablokují. Toto omezení by mohlo být odstraněno placenou verzí, ale podpora Google API již byla společností Google ukončena.

Ukázka stahování výsledků pomocí Google API je ve výpise 20.

```
Parallel.For(0, 8, (currentIndex) =>
  {
    string request = "";
    using (WebClient web = new WebClient())
    {
      web.Encoding = Encoding.UTF8;
      string googleURL = String.Format("{0}?v=1.0&q={1}&rsz=large&hl={3}&start={2}", "http
          ://ajax.googleapis.com/ajax/services/search/web", query, currentIndex * 8, gl);
      request = web.DownloadString(googleURL);
    }

    JObject json = JObject.Parse(request);

    if (json["responseData"].HasValues)
    {
      int position = 0;
      foreach (var res in json["responseData"]["results"])
      {
        string cacheUrl = (string)res["cacheUrl"];
        string unescapedUrl = (string)res["unescapedUrl"];
        string cachedContent = "";
        string pattern = @".*\.pdf";
        Regex rgx = new Regex(pattern, RegexOptions.IgnoreCase);
        Match match = rgx.Match(unescapedUrl);

        if (match.Success)
        {
          if (cacheUrl != "")
          {
            Download_Content dc = new Download_Content();
            cachedContent = dc.Download_Stream1(cacheUrl);
          }
        }
        else
        {
          Download_Content dc = new Download_Content();
          cachedContent = dc.Download_ReadRealCharset(unescapedUrl);
        }
        results.Add(new GoogleItem()
        {
```

```
                Position = currentIndex * 8 + position,
                UnescapedUrl = unescapedUrl,
                CacheUrl = cacheUrl,
                Title = (string)res["title"],
                TitleNoFormatting = (string)res["titleNoFormatting"],
                Content = (string)res["content"],
                CachedContent = cachedContent,
            });

            position++;
        }
    }

});

results.Sort();
```

Výpis 20: Stahování výsledků slovního vyhledávání pomocí Google API

7 Hyperbolický layout – vizualizace grafu v hyperbolickém prostoru

Grafem vizualizovaným v hyperbolickém prostoru je získán detailní pohled na vybraný vrchol a jeho okolí při zachováno propojení okolních vrcholů. Pro vizualizaci grafu v hyperbolickém prostoru je nutné vytvořit pro tento graf hyperbolický layout.

Existuje mnoho aplikací využívajících hyperbolický layout. Tyto aplikace ale obsahují určitá omezení. Například nutnost použití stromové struktury grafu[23]. V této kapitole je popsán způsob vytvoření hyperbolického layoutu pro vizualizaci rozsáhlých grafů v hyperbolických prostorech, neobsahující výše zmíněné omezení. I při vizualizaci celého grafu vytvořeným layoutem je zachována detailnost a přehlednost.

7.1 Návrh řešení

Proces vytvoření hyperbolického layoutu by mohl být rozdělen do 4 fází:

1. Vytvoření prvotního layoutu grafu.
2. Vizualizace grafu.
3. Výběr vrcholu našeho zájmu.
4. Modifikace prvotního layoutu hyperbolickým prostorem.

Modifikaci hyperbolickými prostory lze aplikovat až na nějaký vizualizovaný graf. Proto je vytvoření prvotního layoutu nejdůležitější fází. Na kvalitě tohoto layoutu závisí kvalita celkového výsledku. Rozložení vrcholů rozsáhlého grafu v prostoru není triviálním úkolem. Možným řešením je použití již existujících algoritmů pro vizualizaci grafu.

V druhé fázi je potřeba graf vizualizovat, aby poté mohl být vybrán vrchol našeho zájmu. Vhodným řešením je vizualizace grafu v internetovém prohlížeči pomocí vektorové grafiky SVG.

Pro modifikaci layoutu grafu hyperbolickými prostory bude nejprve nutné vytvořit model hyperbolické geometrie. Tento model bude aplikován na prvotní layout grafu, čímž vznikne layout hyperbolický. Tímto budou upraveny prostorové souřadnice jednotlivých vrcholů. Vizualizací takto upraveného grafu bude docíleno požadovaného výsledku, tj. grafu vizualizovaném v hyperbolickém prostoru.

7.2 Propojení jednotlivých komponent

Při realizaci navrhovaného řešení je využito několika komponent. V následujících odstavcích je popsáno jejich vzájemné propojení.

Pro vytvoření prvotního layoutu je využito Gephi Toolkitu popsaném v kapitole 4.1.1 na straně 19. Tento toolkit je napsán v jazyce Java. Dále musí být použito programu IKVM aby mohl být tento toolkit ovládán z prostředí ASP .NET. Příkazem ve výpisu 21 je vytvořena knihovna, kterou je nutno připojit k aplikaci. K ovládání takto vytvořené knihovny je zapotřebí k aplikaci také připojit knihovny jádra programu IKVM. Tímto

je docíleno možnosti ovládat Gephi Toolkit z prostředí .NET příkazy programovacího jazyka Java.

```
ikvmc -target:exe gephi-toolkit.jar
```

Výpis 21: Vytvoření knihovny Gephi Toolkit

Pro potřebu ukládání struktury grafu včetně všech atributů je použito souboru ve formátu JSON, který je popsán v kapitole 5.1 na straně 24. Pro usnadnění práce s tímto souborovým formátem je využito Frameworku JSON.NET, který je popsán v kapitole 5.1.1 na straně 26. Pro možnost využívat třídy pro export a import, tj. *JsonTextWriter* a *JsonTextReader*, je zapotřebí připojit k aplikaci knihovnu Newtonsoft.Json.

Vizualizace grafu v internetovém prohlížeči vektorovou grafikou SVG je řešena pomocí JavaScriptové knihovny D3.js popsané v kapitole 5.4 na straně 30. Pomocí knihovny D3 jsou data v souboru formátu JSON svázána s hierarchickou strukturou webové stránky. Na základě dat v ní lze vytvářet objekty vektorové grafiky SVG a dále s nimi pracovat.

7.3 Vlastní implementace

Následuje popis vlastní implementace jednotlivých fází směřujících k vytvoření hyperbolického layoutu.

7.3.1 Vytvoření prvotního layoutu grafu

Pro vytvoření layoutu grafu byl použit algoritmus Yifan Hu Proportional. Tento algoritmus v sobě implementuje program Gephi. Pro možnost volání funkcí programu Gephi ve svých programech byl vytvořen program Gephi Toolkit (viz kapitola 4.1.1 na straně 19), který lze z prostředí .NET ovládat pomocí programu IKVM compiler (viz kapitola 4.2.1 na straně 21). Načtení grafu je realizováno pomocí importéru implementovaném v Gephi Toolkitu. Ukázka importu je ve výpisu 3 na straně 22.

Po načtení grafu je potřeba nastavit parametry algoritmu Yifan Hu Proportional. Po nastavení je algoritmus spouštěn v cyklech. Na začátku každého cyklu dojde k inicializaci algoritmu. Poté jsou prováděny iterace algoritmu. V závěru každé iterace je snížen maximální dovolený posun vrcholu. Délka cyklu je určena délkou posunu nebo energií systému. Jakmile maximální dovolený posun vrcholu klesne pod určitou mez nebo je energie celého systému minimální, je cyklus ukončen. Lepších výsledků je dosaženo opětovným spouštěním cyklu algoritmu. Jelikož jsou vrcholy grafu po načtení náhodně rozmístěny, mohou být délky cyklů i výsledky nad stejným grafem různé. Po experimentech byl zvolen počet iterací na 400. Ukázka aplikace layout algoritmu Yifan Hu Proportional je ve výpise 22. Ukázka je pokračování výše uvedeného importu grafu. Výstupem je neorientovaný graf.

```
//Získání odkazu na model grafu načteného ve workspace
GraphModel graphModel = ((GraphController)Lookup.getDefault().lookup(new DhnsGraphController
    ().getClass())).getModel();

//Nastavení parametrů algoritmu
```

```
YifanHuProportional yifanHuProportional = new YifanHuProportional();
YifanHuLayout yifanHuLayout = yifanHuProportional.buildLayout();
yifanHuLayout.setGraphModel(graphModel);
yifanHuLayout.resetPropertiesValues();
//parametry Barnes-Hut
// maximální úroveň kvadrantového stromu
yifanHuLayout.setQuadTreeMaxLevel(new java.lang.Integer(10));
// Theta
yifanHuLayout.setBarnesHutTheta(new java.lang.Float(1.2));
//parametry Yifan Hu
// optimální vzdálenost - přirozená délka pružin
yifanHuLayout.setOptimalDistance(new java.lang.Float(100));
// relativní síla mezi elektrickou sílou (odpuzení) a sílou pružiny ( přiblížení )
yifanHuLayout.setRelativeStrength(new java.lang.Float(0.2));
// velikost počátečního kroku ve fázi zavedení
yifanHuLayout.setInitialStep(new java.lang.Float(20));
// poměr kroku, který je použit k aktualizaci velikosti maximálního posunu
yifanHuLayout.setStep(new java.lang.Float(0.95));
// přizpůsobivé chlazení
yifanHuLayout.setAdaptiveCooling(new java.lang.Boolean(true));
// relativní práh konvergence energie
yifanHuLayout.setConvergenceThreshold(new java.lang.Float(0.0001));

// počet iterací
int Runs = 400;
// aktuální počet iterací
int i = 0;
do
{
    yifanHuLayout.initAlgo();
    while (yifanHuLayout.canAlgo() && i < Runs)
    {
        yifanHuLayout.goAlgo();
        i++;
    }
} while (Runs != i);

return graphModel.getUndirectedGraph();
```

Výpis 22: Yifan Hu Proportional layout grafu

Pro export grafu byla vytvořena třída, kterou je uložena struktura grafu do souboru formátu JSON. Programem Gephi jsou souřadnice vrcholů rozmisťovány kolem středu o souřadnicích $(0; 0)$. Pro další vizualizaci potřebujeme souřadnici $(0; 0)$ v levém horním rohu. Z tohoto důvodu je nejprve zjištěn rozptyl vrcholů, tj. minimální a maximální souřadnice x a y. Tímto je zároveň zjištěna potřebná šířka a výška zobrazovací plochy pro vykreslení grafu. Pomocí Frameworku JSON.NET a v něm implementované třídy *JsonTextWriter* jsou do souboru postupně zapisovány objekty. Nejprve všechny vrcholy, dále všechny hrany a nakonec rozměry vykreslovací plochy. U každého vrcholu jsou uloženy následující údaje: název vrcholu, skupina a upravené souřadnice. Skupina zde zatím nemá žádný význam, tento parametr bude použit později pro určení hloubky vrcholu v grafu. U každé hrany jsou uloženy identifikátory vrcholů, které hrana spojuje,

a ohodnocení hrany. Identifikátory vrcholů představují jejich pořadí ve výčtu začínající od nuly.

V této fázi máme vytvořen layout grafu. Graf je uložen v souboru formátu JSON. Nyní je vše připraveno pro vizualizaci grafu v internetovém prohlížeči.

7.3.2 Vizualizace grafu a výběr vrcholu

Pro vizualizaci grafu v internetovém prohlížeči vektorovou grafikou SVG byl napsán skript v jazyce JavaScript. Tento skript využívá JavaScriptové knihovny D3.js (viz kapitola 5.4 na straně 30). Vizualizace je tedy prováděna na straně klienta. Nejprve jsou načtena data vytvořená v předchozí fázi. Poté je vytvořena vykreslovací plocha pro SVG objekty. Tuto plochu v internetové stránce představuje prázdný element *div*, kterému je nastaveno bílé pozadí a id „graf“. Rozměry jsou určeny uloženou šířkou a výškou v souboru. Dále jsou načteny všechny vrcholy a všechny hrany. Jelikož jsou u každé hrany uloženy identifikátory vrcholů, tak je dalším krokem propojení každé hrany s konkrétními vrcholy podle identifikátorů. Následuje vykreslení hran a poté vykreslení vrcholů.

Pro každou hranu je vytvořen SVG objekt *line* (čára) a pro každý vrchol vytvořen SVG objekt *circle* (kruh). Objektům *line* jsou nastaveny následující parametry: průhlednost, šířka dle ohodnocení hrany, barva a souřadnice určující počátek a konec hrany. Souřadnice jsou načteny z vrcholů, které jsou hranou propojeny. Objektům *circle* jsou nastaveny tyto parametry: barva a šířka ohraničení, souřadnice středu, poloměr, barva výplně a popisek. Nad objekty představující vrcholy je definována funkce, kterou je po kliknutí na vrchol předán jeho název pro další zpracování. Pro lepší přehlednost je vybrání vrcholu kliknutím doprovázeno animací vybraného objektu *circle*. Skript pro vizualizaci je ve výpisu 23 v příloze A na straně 55.

Po výběru vrcholu je možno nastavit do jaké hloubky má být graf vizualizován. Pro potřeby hyperbolického layoutu lze nastavit vizualizační hranici prostoru v procentech. Tato hranice je vztažena ke vzdálenosti nejvíce vzdáleného vrcholu od vrcholu vybraného. Vizualizační hranice prostoru je vypočítána jako x % z největší vzdálenosti od vybraného vrcholu, kde x je nastavená hodnota. Ve vytvořeném hyperbolickém modelu si tuto hranici můžeme představit jako kružnici se středem ve středu hyperbolického prostoru a poloměrem odpovídajícímu 75 % poloměru hyperbolického prostoru. Vrcholy se vzdáleností menší než je vizualizační hranice jsou tedy vykresleny ve vzdálenosti prvních 75 % prostoru od vybraného vrcholu a vrcholy se vzdáleností vetší než je vizualizační hranice jsou vykresleny na okraji hyperbolického prostoru. Tímto je docíleno efektu roztažení prostoru, kdy vzdálené vrcholy uvolní prostor pro přehledné vykreslení blízkých vrcholů.

V této fázi máme vybrán vrchol a nastaveny parametry pro vytvoření hyperbolického layoutu grafu. Vše je předáno na server. Nyní je potřeba načíst graf ze souboru. Pro import grafu byla vytvořena třída, kterou je načtena struktura grafu ze souboru formátu JSON. Pomocí Frameworku JSON.NET a v něm implementované třídy *JsonTextReader* jsou postupně načítány objekty. Všechny vrcholy včetně jejich parametrů jsou uloženy ve třídě *Nodes*. Hrany a jejich parametry jsou uloženy ve třídě *Edge*. Vše je završeno třídou *Graph*, do které je uložen seznam všech vrcholů a hran. Dalším krokem je vytvoření stromu z na-

čteného grafu. Kořen stromu bude vybraný vrchol. Hloubka stromu bude dle zadaného parametru v internetovém prohlížeči.

7.3.3 Modifikace prvotního layoutu hyperbolickým prostorem

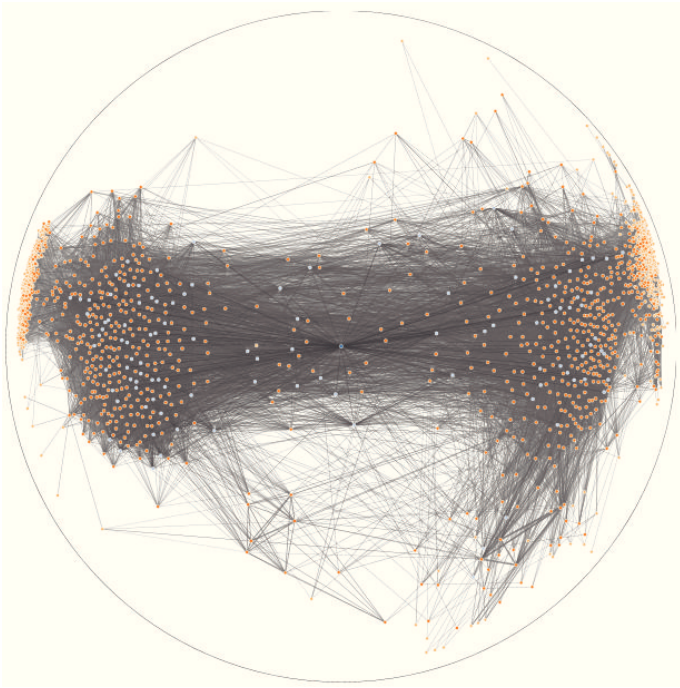
Pro účel vytvoření stromu byla vytvořena třída *Tree*, ve které je obsažen seznam vrcholů a hran. Jako první je do seznamu vrcholů uložen vybraný vrchol. Tomuto vrcholu je nastavena hloubka 0. Dále je postupováno v cyklech. Na začátku každého cyklu jsou ve stromu vybrány vrcholy aktuálně zpracovávané hloubky. Pro každý vybraný vrchol jsou hledány hrany, které jedním svým koncem tento vrchol propojují s jiným vrcholem. Všechny tyto hrany jsou uloženy do seznamu hran. Vrcholy na druhém konci hran jsou přidány do seznamu vrcholů s hloubkou, která je rovna aktuálně zpracovávané hloubce navýšené o jedničku. Na konci každého cyklu je hloubka navýšena. Zpracovávání je ukončeno, pokud je dosaženo předem zvolené hloubky nebo pokud byl graf zpracován celý. Při ukládání hran je potřeba dbát na uložení správných identifikátorů vrcholů, neboť se oproti načtenému grafu mění pořadí vrcholů v seznamu.

Poslední fází je aplikace hyperbolických prostorů na zpracovaný graf. Nejprve jsou vypočítány vzdálenosti všech vrcholů od vybraného vrcholu. Tyto vzdálenosti jsou uloženy do pole. Dále je vypočítána vizualizační hranice prostoru. Všechny vrcholy grafu jsou posunuty tak, aby vybraný vrchol byl na souřadnicích $(0; 0)$. Souřadnice všech vrcholů budou dále upraveny pomocí vytvořeného hyperbolického modelu. Souřadnice vrcholů obsahují pouze složky x a y. Graf je tedy v rovině $z = 0$. První krokem je namapování vrcholů grafu na hyperboloid. Umístíme horní polovinu hyperboloidu nad graf tak, aby byl nejnižší bod hyperboloidu na souřadnici $(0; 0; 1)$, tj. $z = 1$. Vezmeme souřadnice vrcholu, při zohlednění vizualizační hranice, a vedeme z něj kolmici. Souřadnice z vznikne průnikem kolmice s plochou hyperboloidu. Předposledním krokem je využití středového promítání podobně jako v Poincaré disk modelu. Střed promítání je umístěn na souřadnice $(0; 0; -1)$. Vrcholy na hyperboloidu jsou promítány do roviny $z = 0$. Vrcholy jsou nyní rozmístěny v rovině uvnitř hyperbolického prostoru, jehož hranice představující nekonečno je kružnicí o poloměru 1. Posledním krokem je roztažení prostoru do celé vykreslovací plochy a přepočet souřadnic vrcholů.

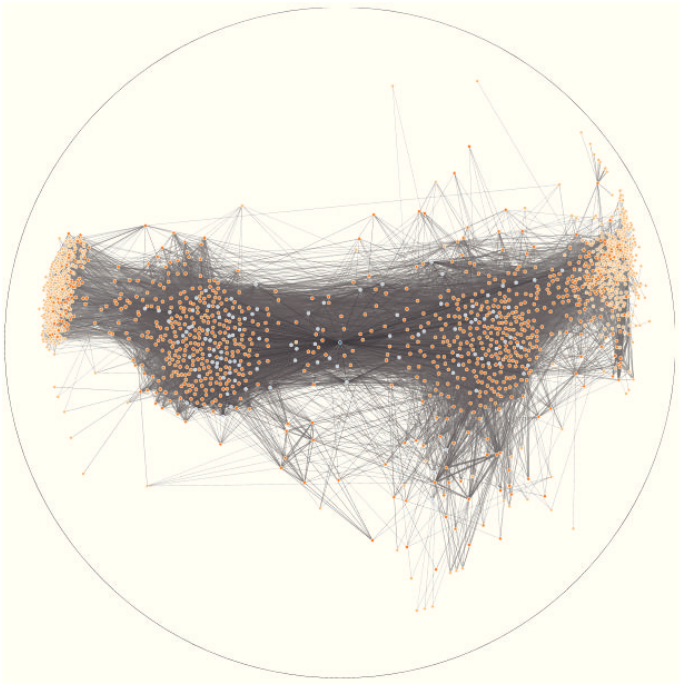
Export grafu byl již popsán výše. Souřadnice jsou opět upraveny tak, aby souřadnice $(0; 0)$ představovala levý horní roh vykreslovací plochy. Jelikož je hyperbolický prostor ohraničen kružnicí, jsou rozměry vykreslovací plochy čtvercové. Pro další využití při vizualizaci vrcholů má nyní parametr skupina význam. Je v něm uložena hodnota hloubky, ve které se vrchol vůči vybranému vrcholu nachází.

Rozdílem oproti výše uvedenému způsobu vizualizace je vykreslení hranice prostoru. Pro další zpřehlednění je aplikováno postupné zprůhledňování hran podle hloubky vrcholů, které jsou hranou propojeny.

Ukázka grafu vizualizovaného v hyperbolickém prostoru pomocí vytvořeného hyperbolického layoutu je na obrázcích 4 a 5. Rozdíl mezi obrázky je v nastavení vizualizační hranice hyperbolického prostoru, kde obrázek 4 je v hyperbolickém prostoru více roztažen.



Obrázek 4: Experiment – Graf č. 1 vizualizovaný v hyperbolickém prostoru – hranice 15%



Obrázek 5: Experiment – Graf č. 1 vizualizovaný v hyperbolickém prostoru – hranice 30%

8 Experimenty s vizualizací

Tato kapitola se zabývá experimenty s vizualizací. Na základě těchto experimentů budou v programu nastaveny vhodné parametry a vyslovena doporučení pro jeho používání. Z důvodu přehlednosti textu a možnosti porovnání obrázkových výstupů budou všechny obrázky uvedeny v příloze B na straně 57. Zhodnocení výsledků jednotlivých experimentů obsahuje sekce 8.1.

Experimenty byly prováděny nad třemi kolekcemi dat. První představuje graf nejmenované rozsáhlé sociální sítě. Z grafů určených pro experimenty je tento největší. Druhá a třetí kolekce představuje graf sítě DBLP získaný z dvou různých pohledů. DBLP server poskytuje bibliografické informace o významných publikacích a článcích týkajících se informatiky. Původně byl systém zaměřen na databázové systémy a logické programování, odtud DBLP (DataBase systems and Logic Programming). Parametry jednotlivých kolekcí jsou uvedeny v tabulce 3.

Časy uváděné v tabulkách jsou získané průměrem jednotlivých měření. Každé měření bylo provedeno minimálně 5 krát. Všechny časy jsou uváděny v milisekundách.

V první fázi je potřeba graf předzpracovat, vytvořit jeho prvotní layout. První experiment se týká počtu iterací vizualizačního algoritmu. Kvalita předzpracování je ovlivněna nastavenými parametry zvoleného algoritmu (ty byly převzaty z programu Gephi), složitostí grafu, náhodným rozložením a počtem iterací algoritmu. Důležitým faktorem je nejen kvalita, ale i čas, ve kterém je graf předzpracován. Délka jedné iterace závisí na složitosti zpracovávaného grafu. Celkový čas předzpracování závisí na počtu iterací a také na složitosti grafu, neboť se každý prvek grafu (vrcholy a hrany) musí exportovat do formátu JSON. Výsledek nad stejným grafem nemusí být vždy stejný, a to z důvodu náhodného rozložení grafu při jeho načtení. Pro tento experiment byl zvolen graf č. 1 pro jeho nejkomplikovanější strukturu. Na obrázcích 6, 7, 8, 9, 10 a 11 je zobrazen grafický výstup programu pro grafy zpracované v počtu iterací: 0 (náhodné rozložení), 100, 200, 400 a 600. Kromě úrovně zpracování se grafy mohou lišit svou orientací v prostoru, a to opět z důvodu náhodného rozložení grafu v prostoru před jeho zpracováním.

Každá hrana v grafu je ohodnocena číslem. Jedním z parametrů při vytváření layoutu grafu je hodnota ohodnocení hran, určující, s kterými hranami bude pracováno. Například v síti spoluautorů chceme zobrazit autory (vrcholy), kteří spolu napsali více než 3 (hrany s ohodnocením větším než 3) publikace či články. Tímto jsou z grafu při předzpracování vyřazeny hrany s ohodnocením 3 a menším. Pokud nastane situace, že po odstranění hran existuje vrchol, ke kterému není přiřazena žádná hrana, tak je také vyřazen. Jedná se o nevýznamného autora (vrchol). Tímto se zjednoduší struktura grafu a zkrátí se čas potřebný pro jeho předzpracování. Další výhodou jednodušší struktury je, že po odstranění nevýznamných hran a vrcholů dokáže vizualizační algoritmus graf ještě přehledněji rozložit v prostoru. Druhý experiment se zabývá měřením času potřebného pro předzpracování grafu. V tabulce 4 jsou uvedeny parametry jednotlivých grafů po uplatnění parametru pro minimální ohodnocení hran. Sloupec „ohodnocení“ udává hodnotu parametru „ohodnocení hran“, kde byly z grafu odstraněny hrany s ohodnocením menším než je uvedená hodnota. Sloupec „pokles prvků o“ udává, o kolik procent poklesl celkový počet prvků (vrcholů a hran) v grafu při změně ohodnocení o hodnotu $+1$. V tabulce 5 jsou uve-

Číslo	Typ	Vrcholů	Hran
1.	Graf rozsáhlé sociální sítě	2 128	51 279
2.	Graf sítě spoluautorů DBLP 1	3 687	9 154
3.	Graf sítě spoluautorů DBLP 2	472	1 103

Tabulka 3: Grafy pro experimenty

Graf	Ohodnocení	Vrcholů	Hran	Celkem prvků	Pokles prvků o
1.	0	2 128	51 279	53 407	—
1.	1	2 092	22 026	24 118	54,84%
1.	2	1 705	6 185	7 890	67,29%
1.	3	1 414	3 083	4 497	43,00%
1.	4	1 118	1 785	2 903	35,45%
1.	5	904	1 188	2 092	27,94%
2.	0	3 687	9 154	12 841	—
2.	1	1 698	3 056	4 754	62,98%
2.	2	1 075	1 586	2 661	44,03%
2.	3	740	980	1 720	35,36%
2.	4	577	700	1 277	25,76%
2.	5	463	524	987	22,71%
3.	0	472	1 103	1 575	—
3.	1	242	485	727	53,84%
3.	2	166	291	457	37,14%
3.	3	127	213	340	25,60%
3.	4	105	170	275	19,12%
3.	5	90	142	232	15,64%

Tabulka 4: Grafy pro experimenty 2

deny časy potřebné pro jednotlivé fáze předzpracování grafu pro 400 iterací a v tabulce 6 pro 600 iterací. Ve třetím sloupci je uveden celkový čas potřebný pro předzpracování. Ve čtvrtém a pátém sloupci jsou uvedeny časy potřebné pro vytvoření layoutu a pro export.

Po předzpracování je graf vizualizován. Vizualizace je řešena pomocí knihovny D3.JS a vytvořeného vizualizačního skriptu v jazyce JavaScript. Samotná vizualizace je tedy prováděna na straně klienta a čas potřebný pro vykreslení grafu v internetovém prohlížeči závisí na výkonnosti konkrétního počítače, na kterém je vizualizace prováděna, a jeho grafické karty.

Poté co je graf vizualizován, je proveden výběr požadovaného vrcholu. Dalším krokem je nastavení parametrů pro další zpracování grafu. Těmito parametry jsou: hloubka grafu, ohodnocení hran a vizualizační hranice hyperbolického prostoru. Po nastavení těchto parametrů můžeme provést zobrazení zpracovaného grafu buď v původním la-

Graf	Ohodnocení	Celkem [ms]	Layout [ms]	Export [ms]
1.	0	76 181	69 680	6 455
1.	1	58 251	55 527	2 704
1.	2	39 587	38 926	654
1.	3	32 371	32 083	284
2.	0	63 820	62 115	1 697
2.	1	26 851	26 544	304
2.	2	16 110	15 985	122
2.	3	11 365	11 305	56
3.	0	6 321	6 278	41
3.	1	3 344	3 326	16
3.	2	2 157	2 143	11
3.	3	1 652	1 639	8

Tabulka 5: Časy předzpracování grafu – 400 iterací

Graf	Ohodnocení	Celkem [ms]	Layout [ms]	Export [ms]
1.	0	107 787	101 126	6 386
1.	1	83 818	81 149	2 648
1.	2	55 718	55 076	637
1.	3	44 010	43 724	282
2.	0	96 355	94 632	1 714
2.	1	41 259	40 942	312
2.	2	23 751	23 637	112
2.	3	16 569	16 508	60
3.	0	10 477	10 437	38
3.	1	5 096	5 078	15
3.	2	3 238	3 224	12
3.	3	2 524	2 513	8

Tabulka 6: Časy předzpracování grafu – 600 iterací

youtu, nebo v hyperbolickém layoutu. Stejně jako ve fázi předzpracování je parametr „ohodnocení hran“ možno nastavit i ve fázi vizualizace. Hodnota parametru nemá v této fázi žádný vliv na strukturu grafu ani na jeho layout. Slouží pouze vizualizačnímu skriptu pro určení, které hrany se nemají vykreslit. Dále je pracováno se všemi hranami i vrcholy. Tohoto lze využít, pokud chceme vizualizovat celý graf, avšak netušíme, jaká hodnota ohodnocení je pro naši potřebu dostačující. Po jejím zjištění můžeme graf opět předzpracovat s již konkrétní hodnotou. Další experimenty se zaměřují na rozdíly vykreslení grafu s omezením pro ohodnocení hran a časy potřebné pro zpracování grafu v závislosti na jeho struktuře.

Ve třetím experimentu si nejprve ukážeme rozdíly na prvním grafu. Celý jej můžeme vidět na obrázku 10. Počet všech hran je 51279. Z tohoto důvodu nemůže vizualizační algoritmus vrcholy od sebe dostatečně roztáhnout a vznikají tzv. chumly. Pro další zpracování je zvolen parametr „ohodnocení hran“ tak, aby se pracovalo pouze s hranami, majícími ohodnocení větší než 3. Na obrázku 12 je vidět celý graf s vykreslenými hranami pro ohodnocení větší než 3. Na obrázku 13 je vidět jeden z chumlů vrcholů v předzpracovaném grafu.

Čtvrtým experimentem bylo nastavení vizualizační hranice hyperbolického prostoru. Na obrázcích 4 a 5 je vidět výsledek vizualizace prvního grafu hloubky 4 s hranicí 15% a 30%.

Dále pokračuje třetí experiment nad druhým grafem. Na obrázku 14 je vidět celý graf, na obrázku 15 je vidět graf s omezením ohodnocení a na obrázku 16 je zobrazena část grafu, který byl předzpracován s omezeným ohodnocením. Jak se tyto tři verze tohoto grafu vizualizují do hyperbolického prostoru je vidět na obrázcích 17, 18 a 19. Třetí nejmenší graf je na obrázku 20, jeho vizualizace v hyperbolickém prostoru je na obrázku 21 a posledním obrázkem 22 je vizualizace předzpracovaného grafu s omezením ohodnocení v hyperbolickém prostoru.

Během provádění třetího experimentu byl měřen čas zpracování grafů a vytváření hyperbolického layoutu. Každé měření bylo prováděno jak nad celým grafem, tak nad grafem zpracovávaným do hloubky 4. Část zpracování grafu obsahuje: import grafu, zpracování struktury a export. Část vytvoření hyperbolického layoutu obsahuje: převzetí zpracované struktury, vytvoření hyperbolického layoutu a export grafu. Výsledky měření jsou uvedeny v tabulce 7.

8.1 Zhodnocení výsledků experimentů

Dle vizuálních výsledků prvního experimentu (obrázky 6, 7, 8, 9, 10 a 11) můžeme říci, že dostačující layout grafu je vytvořen po 400 iteracích a dále dochází pouze k malým posunům vrcholů. Vzhledem k časům získaným v druhém experimentu (tabulky 5 a 6) jsou rozdíly mezi 400 a 600 iteracemi na malých grafech 4 vteřiny a na větších přes 30 vteřin. Při předzpracování grafu je doporučeno volit 400 iterací. Dále dle potřeby větší kvality na úkor času lze volit počet iterací větší.

Pokud víme, že nás budou v grafu zajímat pouze vrcholy propojené hranami o daném ohodnocení, je vhodné toto ohodnocení zadat jako parametr při předzpracování. Jak lze vidět v tabulce 4 celkový počet prvků v grafu velice rychle klesá s každým navýšením

Číslo grafu	Ohodnocení	Zpracování grafu do hloubky 4 [ms]	Hyperbolický layout do hloubky 4 [ms]	Zpracování grafu pro celý graf [ms]	Hyperbolický layout pro celý graf [ms]
1.	0	22 328	209	23 206	303
1.	1	5 623	144	5 879	183
1.	2	328	18	1 108	66
1.	3	143	8	401	33
2.	0	897	28	3 488	98
2.	1	166	10	240	10
2.	2	79	8	104	8
2.	3	51	7	53	7
3.	0	85	18	91	20
3.	1	35	12	36	14
3.	2	24	8	24	9
3.	3	18	8	18	9

Tabulka 7: Časy zpracování pro vizualizaci a pro vytvoření hyperbolického layoutu

požadovaného ohodnocení. V závislosti na počtu prvků je zjednodušena struktura grafu a zkracuje se i čas potřebný pro předzpracování grafu (tabulky 5 a 6). Po odstranění nevýznamných hran a vrcholů dokáže vizualizační algoritmus graf ještě přehledněji rozložit v prostoru.

Zhodnocení výsledků třetího experimentu nad prvním grafem je následující. Pomocí omezení vizualizace hran parametrem ohodnocení můžeme vidět vztah vizualizovaných hran vůči celému grafu (obrázek 12). Při omezení ohodnocení již při předzpracování získáváme detailnější pohled díky lepšímu zpracování vizualizačním algoritmem (obrázek 13).

Dle časů získaných při třetím experimentu (tabulka 7) a také ve druhém (tabulky 5 a 6) je odvozeno následující. Pokud se opravdu zajímáme o významné vrcholy v grafu, je doporučeno omezit ohodnocení hran již při předzpracování grafu. Tímto je zmenšena struktura a složitost grafu. Na základě menší složitosti grafu je předzpracování dosaženo v kratším čase. Ve fázi vizualizací dostáváme detailnější pohled díky kvalitnějšímu zpracování vizualizačním algoritmem a samotné zpracování grafu i jeho zobrazení do hyperbolických prostorů je provedeno v kratším čase.

Doporučeným nastavením vizualizační hranice hyperbolického prostoru zjištěným ve čtvrtém experimentu je hodnota 15%. V případě potřeby je možno prostor dále roztahovat.

Při pokračování třetího experimentu byl použit i druhý graf (obrázek 14). Stejně jako u grafu č. 1 lze vidět vztah vizualizovaných hran vůči celému grafu (obrázek 15) a při omezení ohodnocení již při předzpracování (obrázek 16) je získán detailnější pohled na části grafu. Omezením hran při vizualizaci v hyperbolickém prostoru (obrázky 18 a 19) je dosaženo větší přehlednosti oproti vizualizaci celého grafu (obrázek 17). Nejoptimálnější výsledek je dosažen vizualizací předzpracovaného grafu s omezením ohodnocení v hyperbolickém prostoru (viz rozdíly mezi obrázky 18 a 19). Stejně tak u třetího grafu (obrázek 20) (viz rozdíl mezi obrázky 21 a 22).

9 Závěr

Tato diplomová práce se zabývala problematikou vizualizace grafu pomocí hyperbolických prostorů. Byl navržen a poté realizován způsob vytvoření hyperbolického layoutu, pomocí kterého lze detailně vizualizovat rozsáhlé grafy v hyperbolických prostorech se zachováním propojení okolních vrcholů. Hyperbolický layout je založen na layoutu grafu, který byl získán pomocí již existujících vizualizačních algoritmů. Modifikací získaného layoutu grafu navrženým modelem hyperbolické geometrie vznikl hyperbolický layout. Z výsledků provedených experimentů vyplývá, že vytvořený hyperbolický layout pro vizualizaci rozsáhlých grafů v hyperbolickém prostoru je snadný, rychlý a přehledný.

Pro předzpracování grafu a využití již existujících vizualizačních algoritmů byl vytvořen program „Gephi Layout“. V něm bylo použito Toolkitu programu Gephi, který byl ovládán z prostředí ASP .NET pomocí knihovny vytvořené programem IKVM. Pro vizualizaci a další zpracování byl graf po vytvoření prvotního layoutu exportován do souborového formátu JSON.

Pro vizualizaci grafů a jejich další zpracování byla vytvořena webová aplikace „Web for Laying Out Large Graphs in Hyperbolic Space“. Vizualizace byla prováděna v internetovém prohlížeči, na straně klienta, pomocí vektorové grafiky SVG s napojením na datový soubor formátu JSON. Objekty grafiky SVG byly generovány JavaScriptovou funkcí využívající knihovnu D3.JS. Webovou aplikací byl načten graf v souboru formátu JSON. Ten byl dále zpracován a poté vizualizován do hyperbolického prostoru pomocí vytvořeného hyperbolického layoutu. Výstup aplikace byl jak datový (graf zpracovaný hyperbolickým layoutem uložený v souboru formátu JSON), tak vizuální (grafický výstup v internetovém prohlížeči vytvořený grafikou SVG).

V práci se mimo vizualizace řešila i problematika získávání dat. Pro získání dat bylo do webové aplikace implementováno stahování výsledků slovního vyhledávání pomocí Google API a jejich zdrojových kódů. Pro soubory formátu PDF bylo využito Google Cache. Stahování bylo urychleno pomocí paralelizace. Implementované stahování slovního vyhledávání může být dále použito pro vizualizace výsledků vyhledávání.

Vytvořené aplikace jsou vhodné pro další zpracování a rozšíření. Několik návrhů na rozšíření bylo popsáno v kapitole 9.1. Vizualizace pomocí hyperbolických prostorů již má své uplatnění a studování této problematiky má pro detailní a přehledné vizualizace rozsáhlých a komplexních sítí svou budoucnost.

9.1 Návrhy na rozšíření vizualizačního algoritmu

V této kapitole jsou popsány návrhy, jak by mohl být vytvořený vizualizační algoritmus dále rozšířen.

Prvním návrhem je obohacení vizualizace o prostorovou složku. Při vizualizaci rozsáhlých grafů obsahujících mnoho hran, dochází k častému křížení hran a znepřehlednění vizualizace. Cílem rozšíření o prostorovou složku je získání 3D efektu při vizualizaci, kdy by se vrcholy směrem od hranice hyperbolického prostoru k jeho středu opticky přibližovaly (zvětšovali se). Došlo by také k oddálení hran v závislosti na oddálení vrcholů, kdy by se vzdálenější hrany postupně ztrácely (zprůhledňovaly se).

Druhým návrhem je upravení algoritmu pro paralelizaci. Cílem tohoto rozšíření je spouštění algoritmu v paralelních vláknech, čímž by se docílilo rychlejšího zpracování.

Třetí návrh na rozšíření se týká fáze vytvoření prvotního layoutu (předzpracování grafu). Tato fáze hraje důležitou roli pro další zpracování grafu. Pro možnost dosáhnutí kvalitnějších výsledků je vhodné pomocí programu Gephi rozšířit možnosti vizualizace o další již existující vizualizační algoritmy či jejich kombinace.

Zpracování grafu vizualizačním algoritmem je ovlivněno mnoha volitelnými parametry. V kapitole 8 zabývající se experimenty, byly uvedeny výsledky měření, jak mohou různé hodnoty parametrů ovlivnit kvalitu a čas zpracování grafu. Cílem čtvrtého návrhu na rozšíření je vytvoření vlastního vizualizačního algoritmu. Spíše než hledání optimálního řešení pomocí již existujících algoritmů či jejich kombinací (používání programu Gephi) je vhodné vytvořit algoritmus vlastní. A to také z důvodu velkého množství parametrů ovlivňujících výsledek předzpracování grafu. Tyto parametry by měly přímý vliv na vlastní vizualizační algoritmus. Tím by algoritmus byl dynamičtější.

Pátým návrhem na rozšíření je vytvoření vlastní vizualizace v internetovém prohlížeči. Nyní je vizualizace prováděna pomocí JavaScriptové knihovny D3.JS a vlastní JavaScriptové funkce. Cílem tohoto rozšíření je vytvoření vizualizačního nástroje, kterým by byly generovány prvky vektorové grafiky SVG.

Šestým návrhem na rozšíření je vytvoření webové služby poskytující výsledky zpracování grafu hyperbolickým layoutem ve formátu GDF, JSON nebo SVG.

10 Reference

[1] Bostock, Mike: *D3.js – Data–Driven Documents*, 2012, http://www.d3js.org/

[2] Connelly, Bob: *Chapter 15 – Hyperbolic geometry*, Classical Geometries, Cornell University 2010, http://www.math.cornell.edu/~web4520/CG15-0.pdf

[3] Crockford, Douglas: *The application/json Media Type for JavaScript Object Notation (JSON)*, červenec 2006, http://tools.ietf.org/html/rfc4627

[4] Eades, Petr: *A heuristic for graph drawing*, Congressus Numeratium 1984.

[5] Frijters, Jeroen: *IKVM.NET*, květen 2011, http://www.ikvm.net/

[6] Fruchterman, Thomas M. J.; Reingold, Edward M.: *Graph Drawing by Force–directed Placement*, Software – Practise and Experience, 1991.

[7] Gephi: *The Open Graph Viz Platform*, 2012, http://gephi.org/

[8] Houston, Ben: *Exocortex.DSP – An open source C# Complex Number and FFT library for Microsoft .NET*, říjen 2003, http://www.exocortex.org/dsp/

[9] Hu, Yifan: *Algorithms for Visualizing Large Networks*, AT&T Labs – Research, 16. září 2011.

[10] Hu, Yifan: *Efficient and High Quality Force–Directed Graph Drawing*, Wolfram Research Inc.

[11] Hu, Yifan; Koren, Yehuda: *Extending the Spring-Electrical Model to Overcome Warping Effects*, AT&T Labs – Research; Yahoo! Research.

[12] Introducing JSON: http://www.json.org/

[13] Ito, Tadao: *Hyperbolic Non-euclidean World*,http://web1.kcn.jp/hp28ah77/

[14] Ježek, F.: *Geometrie a počítačové modelování*, Pomocný učební text, Plzeň, verze 8.1 - únor 2006.

[15] Joyce, David E.: *Euclid's Elements*, Dept. Math. & Comp. Sci., Clark University 2002, http://aleph0.clarku.edu/~djoyce/java/elements/elements.html

[16] Kamada, Tomihisa; Kawai, Satoru: *An algorithm for drawing general undirected graphs*, Information Processing Letters, 31:7-15, University of Tokyo, 12. květen 1989.

[17] Křížová, Kristýna: *Neeuklidovská geometrie*, Bakalářská práce, Brno 2008.

[18] Křížová, Kristýna: *Neeuklidovská geometrie*, Diplomová práce, Brno 2010.

[19] Matula, Radek: *Grafická reprezentace grafů*, Diplomová práce, Brno 2009.

[20] Munzner, Tamara: *H3: Laying Out Large Directed Graphs in 3D Hyperbolic Space,* Stanford University 1997, http://graphics.stanford.edu/papers/h3/html.nosplit/

[21] Munzner, Tamara; Burchard, Paul: *Visualizing the Structure of the Worls Wide Web in 3D Hyperbolic Space,* University of Minnesota; University od Utah 1995, http://graphics.stanford.edu/papers/webviz/webviz.72dpi.pdf

[22] Newton, James: *JSON.NET,* duben 2012, http://james.newtonking.com/

[23] Toman, Adrian: *Datová analýza a vizualizace studijních aktivit studentů,* Diplomová práce, Ostrava 2009.

[24] Tunkelang, D.: *A Numerical Optimization Approach to General Graph Drawing,* Carnegie Mellon University, Pittsburgh 1999.

[25] Zhang, Jin: *Visualization for Information Retrieval,* The Information Retrieval Series, Vol. 23, 2008.

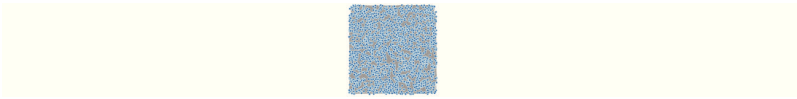
A Výpisy zdrojového kódu

```
var fill = d3.scale.category20();

d3.json("data/input.json", function (json) {

    var svg = d3.select("#graf")
        .append("svg")
        .attr("width", json.w)
        .attr("height", json.h);

    var uzly = json.nodes;
    var hrany = json.links;
    var o, i, m = hrany.length;

    for (i = 0; i < m; i++) {
        o = hrany[i];
        if (typeof o.source == "number") o.source = uzly[o.source];
        if (typeof o.target == "number") o.target = uzly[o.target];
    }

    var lines = svg.selectAll(".link")
        .data(hrany)
        .enter().append("line")
        .attr("class", "link")
        .style("stroke-width", function (d) { return Math.sqrt(d.value); })
        .style("stroke", "#999")
        .attr("x1", function (d) { return d.source.x; })
        .attr("y1", function (d) { return d.source.y; })
        .attr("x2", function (d) { return d.target.x; })
        .attr("y2", function (d) { return d.target.y; });

    var nodes = svg.selectAll(".node")
        .data(uzly)
        .enter().append("circle")
        .attr("class", "node")
        .attr("cx", function (d) { return d.x; })
        .attr("cy", function (d) { return d.y; })
        .attr("r", 5)
        .style("fill", function (d) { return fill(d.group); })
        .on("mouseover", function () {
            d3.select(this).style("fill", "orange");
        })
        .on("mouseout", function () {
            d3.select(this).style("fill", function (d) { return fill(d.group); });
        })
        .on("click", function (d) {
            svg.selectAll(".selected").remove();

            var souradnice = {
                x: d.x,
                y: d.y,
                label: d.name
```

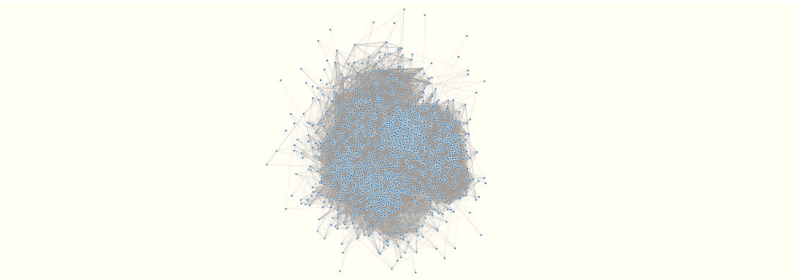
```
            };

            svg.selectAll(".selected")
                .data([souradnice])
                .enter().append("circle")
                .attr("class", "selected")
                .attr("cx", function(d) { return d.x; })
                .attr("cy", function(d) { return d.y; })
                .attr("r", 60)
                .style("fill", "none")
                .style("stroke-opacity", 1e-6)
                .transition()
                .duration(750)
                .attr("r", 5)
                .style("stroke-opacity", 1);

            var hidden = document.getElementById("SelectedNodeId");
            hidden.setAttribute("value", souradnice.label);
        });

    nodes.append("title")
        .text(function(d) { return d.name });
});
```

Výpis 23: JavaScript pro vizualizaci grafu

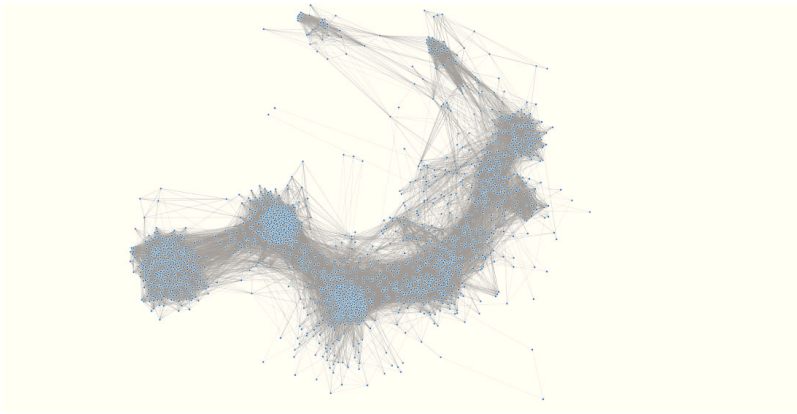
B Obrázky



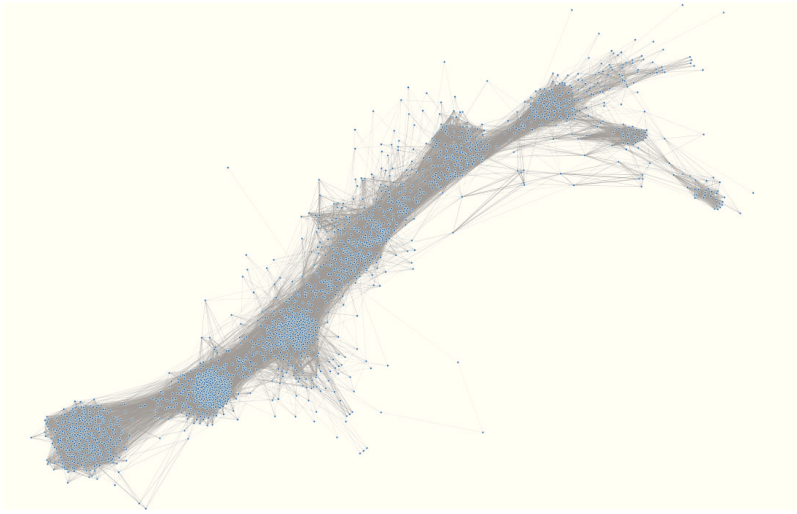
Obrázek 6: Experiment – Graf č. 1 zpracovaný počtem iterací 0



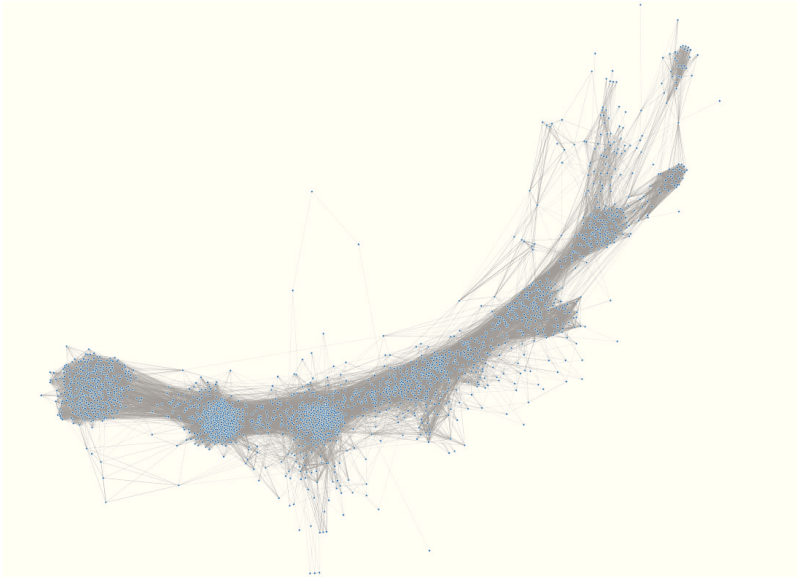
Obrázek 7: Experiment – Graf č. 1 zpracovaný počtem iterací 100



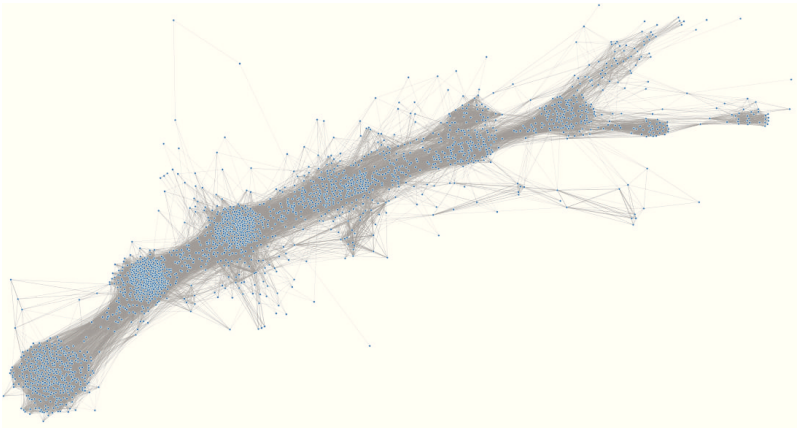
Obrázek 8: Experiment – Graf č. 1 zpracovaný počtem iterací 200



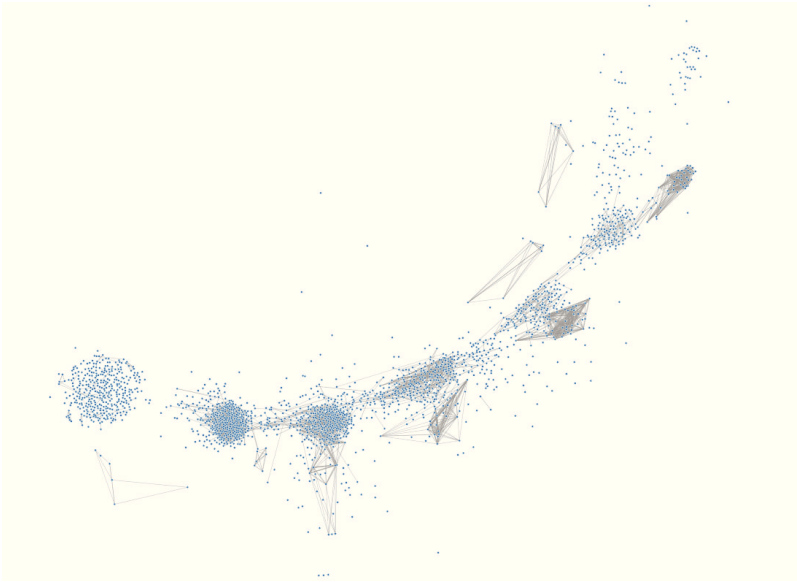
Obrázek 9: Experiment – Graf č. 1 zpracovaný počtem iterací 400



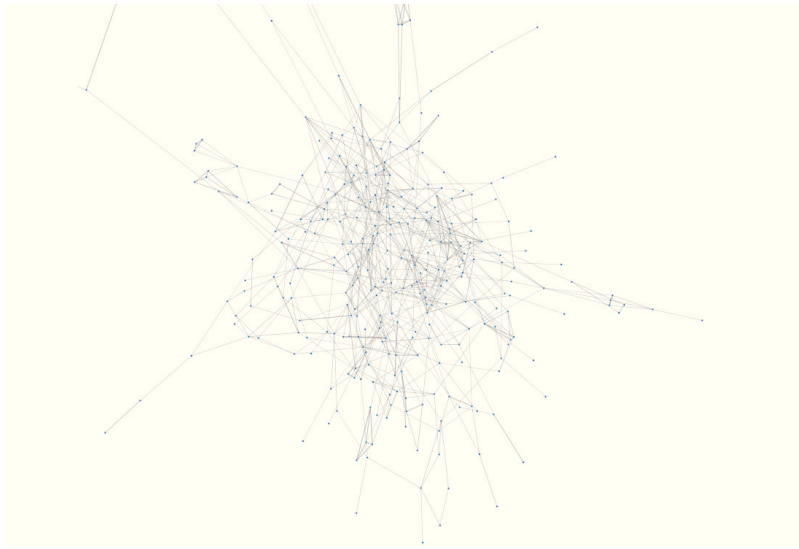
Obrázek 10: Experiment – Graf č. 1 zpracovaný počtem iterací 600



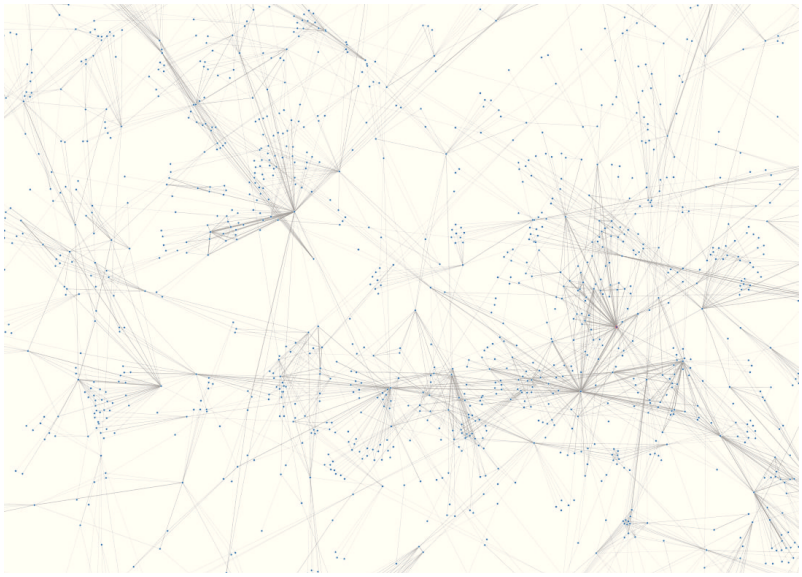
Obrázek 11: Experiment – Graf č. 1 zpracovaný počtem iterací 1000



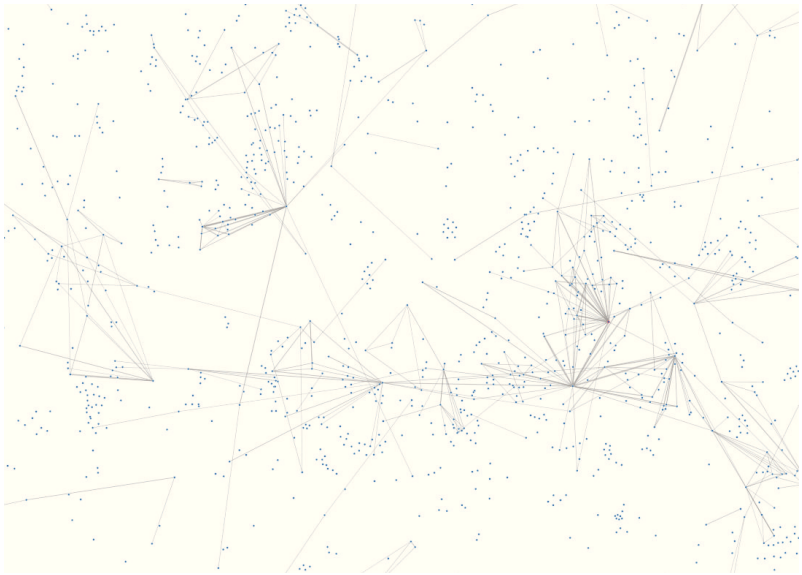
Obrázek 12: Experiment – Graf č. 1 s omezením ohodnocení



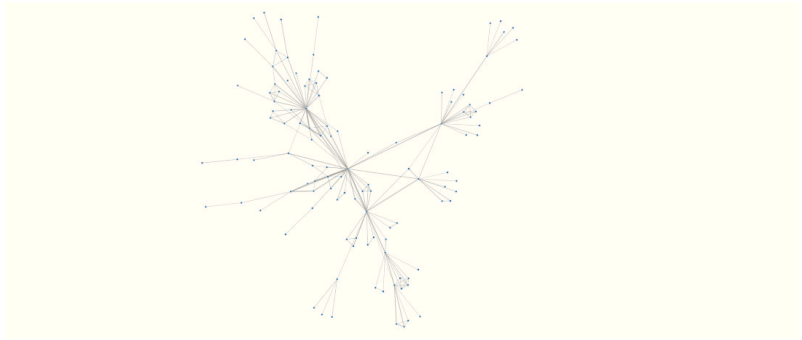
Obrázek 13: Experiment – Část grafu č. 1 z předzpracovaného grafu s omezením ohodnocení



Obrázek 14: Experiment – Graf č. 2 zpracovaný počtem iterací 600

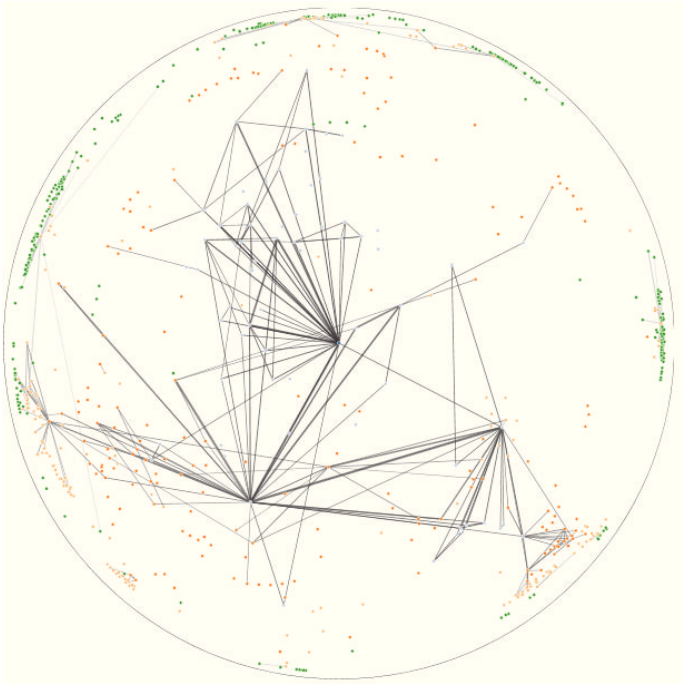


Obrázek 15: Experiment – Graf č. 2 s omezením ohodnocení

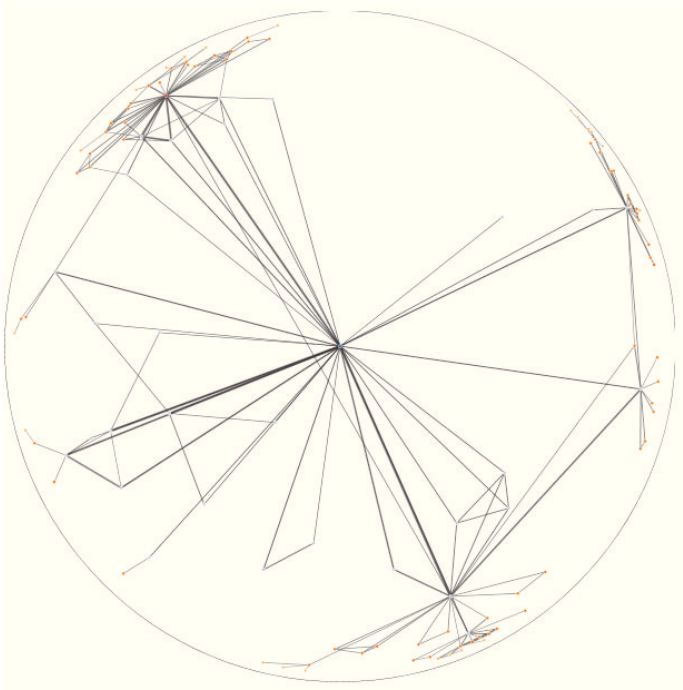


Obrázek 16: Experiment – Část grafu č. 2 z předzpracovaného grafu s omezením ohodnocení

Obrázek 17: Experiment – Graf č. 2 vizualizovaný v hyperbolickém prostoru

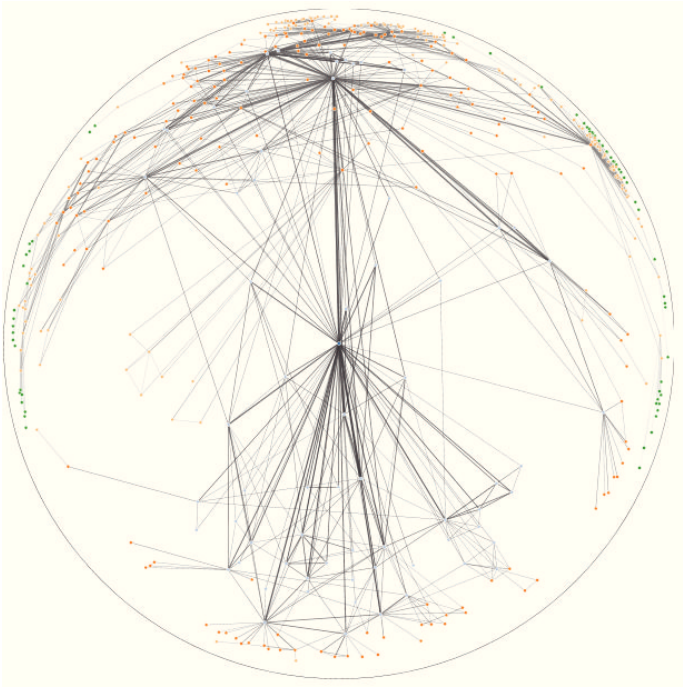


Obrázek 18: Experiment – Graf č. 2 s omezením ohodnocení vizualizovaný v hyperbolickém prostoru

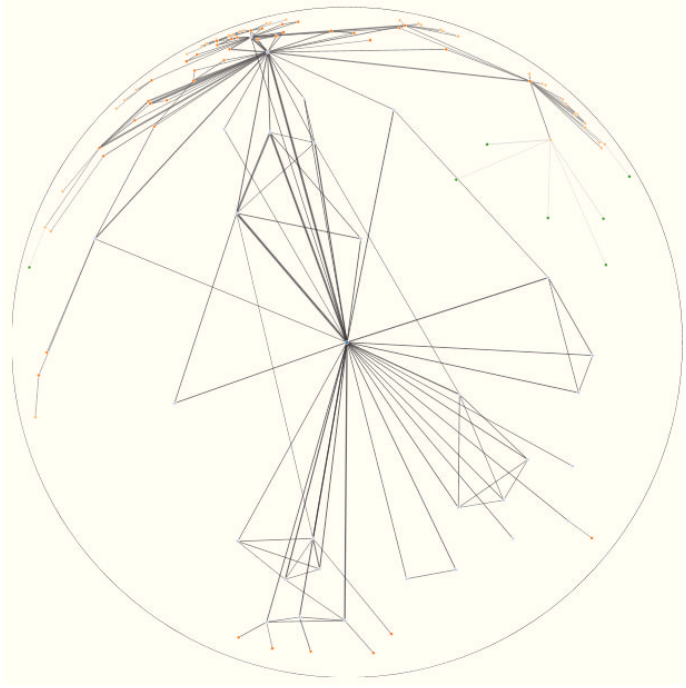


Obrázek 19: Experiment – Část grafu č. 2 z předzpracovaného grafu s omezením ohodnocení vizualizovaná v hyperbolickém prostoru

Obrázek 20: Experiment – Graf č. 3 zpracovaný počtem iterací 600



Obrázek 21: Experiment – Graf č. 3 vizualizovaný v hyperbolickém prostoru



Obrázek 22: Experiment – Část grafu č. 3 z předzpracovaného grafu s omezením ohodnocení vizualizovaná v hyperbolickém prostoru

C CD-ROM

Obsah CD-ROM:

- Text diplomové práce v elektronické podobě
- Program „Gephi Layout“ pro předzpracování grafu a jeho zdrojový kód
- Zdrojový kód webové aplikace „Web for Laying Out Large Graphs in Hyperbolic Space“
- Ukázková vstupní data ve formátu GDF
- Předzpracovaná data pro další zpracování a vizualizaci ve formátu JSON