Department of Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University, Bratislava

Visualization of tagspace

(Bachelor thesis)

Juraj Ďuďák

Thesis advisor: RNDr. Martin Homola

Bratislava, 2009

By this I declare that I wrote this bachelor thesis by myself, only with the help of the referenced literature, under the careful supervision of my thesis advisor.

. . . . . .

Contents

Abstract

Users taggings(assigning metadata keywords) Internet content are creating so called folksonomy, a bottom-up build taxonomy describing relations between resources. This taxonomy can be used as a form of navigation, even though it does not leads to answers, but rather leads to more questions and interesting content.

By aggregating tag according to their name, we get navigation elements called tagclouds. This thesis introduces basic methods of inline tagcloud generation, also, defines tag relations, to improve tagclouds and the model of navigation in this tagspace. Tagcloud generation methods are compared, and experimentally tested, by implementing these algorithms and using them on real data. As a main result we will introduce relational tagbrowser created for portal blog.matfyz.sk, that uses graph layout to visualize tags and relations between them. Algorithms and technologies that lead to creating of this tagbrowser are described and implementation details are listed.

Keywords: tags, relational tagcloud, graph layout

Chapter 1

Introduction

Tags are metadata keywords assigned to data, such as images, atricles or music, to help search and browse this data. Users add this keywords, and create so called folksonomy, a taxonomy, that is different from standard scientifical taxonomies, since it does not have hierarchical structure, but rather describes relations. Taxonomy is build bottom-up, not by dividing objects into categories, but rather by joining related objects.

We can look at this taxonomy as a 3-dimensional space(dimensions are user,tag,object), and we want to allow users to move in this space and browse data, according to tags or users. Users should be able to select so called pivots, and see the tagspace in concrete pivot point, so that they can further move in this space, in direction they want. Tags are visualized in form of tagclouds, mostly lists of tags(inline tagclouds), with font size(or color) representing tag importance. Drawback of these clouds in that they do not show relations, and should be replaced with new generation, relational tagbrowsers. Inline tags are only able to visualize 2 dimensions, so these browsers should use position to visualize third dimension, relations of tags.

Among possible solutions, best idea was to visualize small part of tagspace, as a graph, with tags as vertices and relations shown as edges. Problem

was transformed into problem of finding a graph layout. After considering all methods, we have chosen force-directed layout, and easy to implement heuristics, that work really great for task like this one. Force-directed layout(or spring algorithm) uses physical model of particles and springs, to simulate dynamics of system over time, and finds local kinetic energy minimum. Whole application was done for blog.matfyz.sk blog portal, that is known among students. Portal heavily uses XML technologies, such as native XML database and XSLT transformation.

We will introduce basic tagcloud generation techniques, present options on tagspace navigation, also write about all parts of tagbrowser developed, its implementation and technologies on blog.matfyz.sk.

Chapter 2

Tagging

In our computer era, everyone produces large amounts of data and wants to share with the world. In 2006, the blogosphere was doubling in size every 200 days[Sif06], or about once every 6 and a half months. On July 31, 2006, Technorati tracked its 50 millionth blog, and users produce millions of articles, photographies and videos every day. This data has to be categorized, and somehow, we need to create abstracts, or meaning of data. As far as we know, tags are the best solution.

2.1 Tag

A tag is a non-hierarchical keyword or term assigned to a piece of information. This metadata define context or meaning of such information. A list of tags(also called keywords) is assigned to data to help other users browse and search data. Tag does not have a semantic value itself, but since tagging is collaborative, larger set of tags help to define data semantic. Tag in general, is any textual information, specific or abstract, the power of tags comes from open nature of tagging systems, so users does not need to be experts to tag data. One wrong tag is suppressed by 100 correct. This also brings some

problems including plural and synonymy. But this problems can be removed by using libraries or dictionaries containing such words.

2.2 Tagging

Tagging(creating, browsing tags) is protocol independent, so it can and have to be implemented by web-designers. When implementing such system, programmer has to take into account all the possible tag sources:

- **author** - tags are provided by data author, she manages them, but usually only on data creation time, this option is good for systems with less users, but authors does not tend to consider all aspects of their creations.
- **user** - allows all the users to tag content. This is a good alternative on large systems, without a large user community, someone can exploit this possibility.
- **machine** - uses algorithm to extract tags from given text.
- **combination of above** - probably best solution is combination of above approaches, but be careful to explain to users where does tags come from.

Tagging creates so called folksonomy, which comes from words folk and taxonomy, a user-generated classification system, that is quite different from standard scientific taxonomies. Folksonomy does not create hierarchical system, but rather describes properties and relations between objects, since data used on the Internet is not easy to categorize. Tags are then used for web search, and as a navigation in various forms, mostly as tag clouds. Tagging is intended for users and made by users, and what is most important it does not need much afford to maintain.

Tagging, as we know it today, dates back to 2003, when social bookmarking service Delicious[1] introduced tags, that allowed users to easily find their stored bookmarks.
Another great example is flickr[2], a photo sharing webservice, on which tagging is absolutely essential, because search engines are unable to search for images without being given an image description.
Very innovative approach in tagging was used by Google, in google image labeler, a game, rather than application, where 2 users are paired and both shown a picture, that they are supposed to label with words. Players receive point if their labels match, and the more complex label is, the more points they get. When some image was already labeled, its label becomes off-limits, and this label cannot be used anymore. This game has many players, is quite interesting and provides valuable data. Before image labeler, Google image search was only relying on image context or description, and google was unable to remove undesirable content from web search. But with image labeler, they were given a set of tags associated with pictures, that 2 users, not knowing each other, has both given to picture. This means that label is good enough because 2 people thought of it, moreover, usage of off-limit label enforces users to focus on details of picture, so labels does not only describe key elements of the picture, but all its properties. All this data was retrieved for free and also in very motivating way.
Music is also being tagged, a web-application last.fm created large folksonomy of music, and allowed users to freely add, browse, and filter music by tags. This way you can create your own playlist with music according to specific criteria. Unmoderated and free nature of tagging many times result in irrelevant tags, or even expressing opinions and arguing by adding tags.

[1] www.delicious.com
[2] www.flickr.com

2.3 Future of tagging

In past 6 years tagging became inseparable part of the Internet, and is considered the future of web search, and one of the key features of Web 2.0. Web 2.0, a new generation of web services, relies on social-networking and user-generated content, that needs to be organized, and tags, as far as we know, are the best solution. They need almost no help from service administrators, because users themselves are responsible for quality and usability of tags. Tagging is also important for semantic web[XFMS06], since tags create taxonomy that is not only suitable for humans, but also is machine readable. Tags define most important aspects of data, so that search engines can improve their results(like google did with image labeler) by taking in count the importance of tags. What is more, list of tags is usually part of a web page, so even today, they improve search results.

Chapter 3

Tagclouds

Tagcloud are visual representation of tagspace. They are usually list of tags(inline tags) that visualize tag relevance, or count by size or color. Tag clouds evolved from maps visualization, that used font size to visualize city sizes. Cloud was used for first time by Flickr co-founder and interaction designer Stewart Butterfield. Tagclouds are currently very popular form of navigation, since they are very easy to implement and provide good alternative to clasical navigation. But today, we know they have many drawbacks, and we will also show that we also need some mechanism for navigation between tagclouds.

3.1 Inline tagclouds

The most common form of tag cloud is inline tagcloud, that uses list of tags(usually one word) that are rendered as hypertext links on a website. These lists are usually sorted by alphabet, or by importance of tags. Tagclouds also visualize importance by using different font-size, or changing a color.

When using font-size for displaying tag importance, it is essential to select

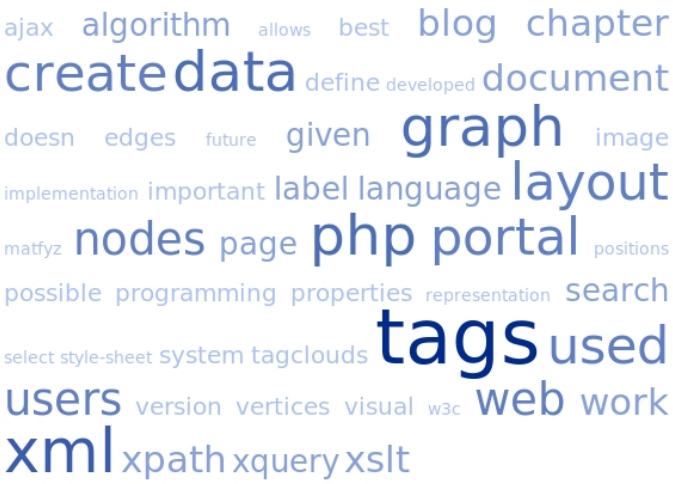

Figure 3.1: Typical tag cloud ordered by alphabet. Font sizes represent frequency of certain tag.

proper scaling formula, because you have to choose compromise between legibility, usability and relevance of font-size. Importance of tags can have almost any value, so to make proper tag cloud, we have to assign maximum and minimum font-sizes for tags. Otherwise, after some time, some tags may become too large or small. This gives us basic parameters for our problem:

- list of n tags with defined importances, with minimum value $\min(importance)$ and maximum value $\max(importance)$
- minimum font-size $minFS$
- maximum fot-size $maxFS$

Our task is to scale all importance values into interval between $minFS$ and $maxFS$, so we need to find function FS, with tag importance as an input and with final font-size as an output. Thus, this function must hold these

properties:

$$\forall i < n \quad FS\left(tag_i\right) \leq maxFS \wedge FS\left(tag_i\right) \geq maxFS$$

$$FS\left(\min\left(importance\right)\right) = minFS$$

$$FS\left(\max\left(importance\right)\right) = maxFS$$

In general, you can choose from 3 best approaches, considering design of your interface and semantics of data you visualize.

- proportional scaling - Font-sizes of tags are distibuted into this interval, according to simple scaling rule

$$FS\left(importance\left(tag_i\right)\right) = minFS+$$

$$\frac{\left(maxFS - minFS\right)\left(importance\left(i\right) - \min\left(importance\right)\right)}{\max\left(importance\right) - \min\left(importance\right)}$$

  Advantage of this scaling is that tags differ a lot, it can be easily seen which tag is the most important. This is because scaling is really done in proportional way. On the other hand, if one tag is a lot more important than the others, all other tags will have font-size a little above minimum font-size, and thus the tag cloud does not have the properties we want.

- linear scaling - uses the same formula as proportional scaling, but instead of using $importance(x)$, we use $log(importance(x))$. This makes the differences between tag importances smaller and produces tagcloud with interesting properties.
  Font-sizes are evenly distributed along interval, and tagcloud aesthetically better. But sometimes, tag-sizes are too close to each other and it is difficult to see which tag is largest. This method mainly improves performance for large importance range tag clouds.

- group scaling - We need to order tags by size, then divide them into n groups. Now, i_{th} group will be assigned font-size

$$minFS + i\frac{maxFS - minFS}{m}$$

where $minFS(maxFS)$ is minimum(maximum) font-size. This technique is quite nasty, it does not really show font-size as a function of importance, but still can be used on systems with very special tag distribution.

Figure 3.2: Comparing 2 different tags size techniques. On the left proportional scaling brings in front most important tag and size really show importance, while linear scaling, on the right, make differences smaller using logarithm, and all tags can be compared according to their size

First two techniques are used very often[Smi08], each of them has its advantages and disadvantages, but to select a technique, one must consider legibility, usability and of course aesthetics.

3.2 Navigating through tagspace

In means of navigation, we can look at the folksonomy created by tagging as a space. This space has three dimensions, author, tag and an article(resource). It is important to use all of them, and allow users to look at tagspace from each of these dimensions.
Common are inline tag clouds, that summarizes all articles, from all users, and visualizes tag count as importance of tags. These tagclouds provide outlook on tagspace, but to provide full feature navigation, user should be allowed to move in any direction in this tag space by selecting so called pivots.
Pivot can be either tag, user or article. By selecting a pivot in one dimension, other dimensions are scaled, so that they contain only the relevant information to this pivot. But also, we should be able to provide tag cloud for each pivot. Standard tag clouds can be created with an user or article pivot point. Also standard tagclouds are only list of tags(one dimensional) and use font-size to visualize tag importance, so we only have 2 dimensions for 3-dimensional space. This mean, that we can provide tagcloud that shows tags that are related to user (tag that user has used), also tags related to article (articles tags). But we can not visualize third dimension, tag. We have to visualize tagcloud, that has a tag as an pivot point, and show all tags related to this chosen tag. But also this tag cloud should somehow visualize relations between its tags so that user can further explore this space.
But first, we have to define relation of tags. There are many ways to do this, but one of the easiest, but still very often used is to relate tags through

articles.

Definition. *Tags t_1, t_2 are related, if there exists a article tagged with both tags.*

This provides us the last piece of information, to finalize tagspace model. We can put into relations all of the 3 dimensions. With a complete model of tagspace, we are able to navigate through it using pivot points, and what is more, we can visualize current space point, and thus give user further options for pivot selection. According to pivot, we can provide:

- **User pivot**
  - list of related users - users using related tags
  - list of articles created by selected user
  - tagcloud consisting of user added tags, tag sizes are only measured from user tags
- **Article pivot**
  - list of related articles according to related tags
  - list of other articles written by author of pivot article
  - tagcloud consisting of tags assigned to pivot article
- **Tag pivot**
  - tagcloud of tags related to pivot tag
  - list of articles tagged with pivot tag
  - list of users using pivot tag

Also, there are other possible dimensions such as time, or popularity(visit count). But these are not "true" dimensions, they are not as strong as those

3 above, but are great for scaling and sorting lists of users and articles. Almost every website uses list of new articles, top rated articles, most visited articles, or sometimes top users(either as most read or best rated). Main goal is to allow users move in tagspace, and explore new dimensions. There is a fundamental difference in the activities of browsing to find interesting content, as opposed to direct searching to find relevant documents in a query. It is similar to the difference between exploring a problem space to formulate questions, as opposed to actually looking for answers to specifically formulated questions.[Mat04]
Now, we have designed a full-feature navigation through tag-space and it is time to show options on how to create better tag clouds, 3-dimensional clouds showing also relations between tags.

3.3 Tagclouds with arbitrary positions

Typical inline tag clouds are 2-dimensional, and to add third dimension, position can be used to create metric. This is typically done by positioning related tags next to each other. This is quite difficult problem, because we have to deal with 2 problems, calculating positions and rendering tag cloud on web page.

- **Calculating positions** - this process is computationally difficult. Many optimizing algorithms working on graphs are NP-complete, and for this purpose, they are replaced with heuristic solutions, that produce quite nice result in reasonable time. Algorithms include greedy principles, dynamic programming and simulation.

- **Rendering** - according to chosen model, one can use more alternatives to display such tag cloud. Possibilities include HTML tables[LK07], divs and floating and also, absolute positioning. Even javascript or

flash animation can be used to produce aesthetically valuable tag cloud.

Other possibility is to connect related tags with lines and to create graph. Then we can display relations by graph edges. We have considered this solution as the best for the portal, and implemented such visualization, since system on portal is not very large.

Figure 3.3: Tag cloud with arbitrary positions use position to visualize tag relations. Related tags are rendered close to each other. This layout can be achieved using HTML tables.

3.4 Geotagging

Another variant of tag clouds are maps, that use pins to visualize geographical source of data. Geotagging is assigning geographical metadata to data. They use special tags, called triple tags or machine tags. They can be used in RDF[1] format and thus in form of 3 values. Namespace, predicate and value itself. For example tag **geo:lat=47** is saying that geographical latitude is 47. These tags are then used for many purposes, most noticable are already mentioned maps, such as google earth. Geotagging is really good choice for photographs, and since these triple tags are easily machine readable, user can browse and search photos by their geographical source.

[1]The Resource Description Framework (RDF), a family of World Wide Web Consortium (W3C) specifications[Gro04a]

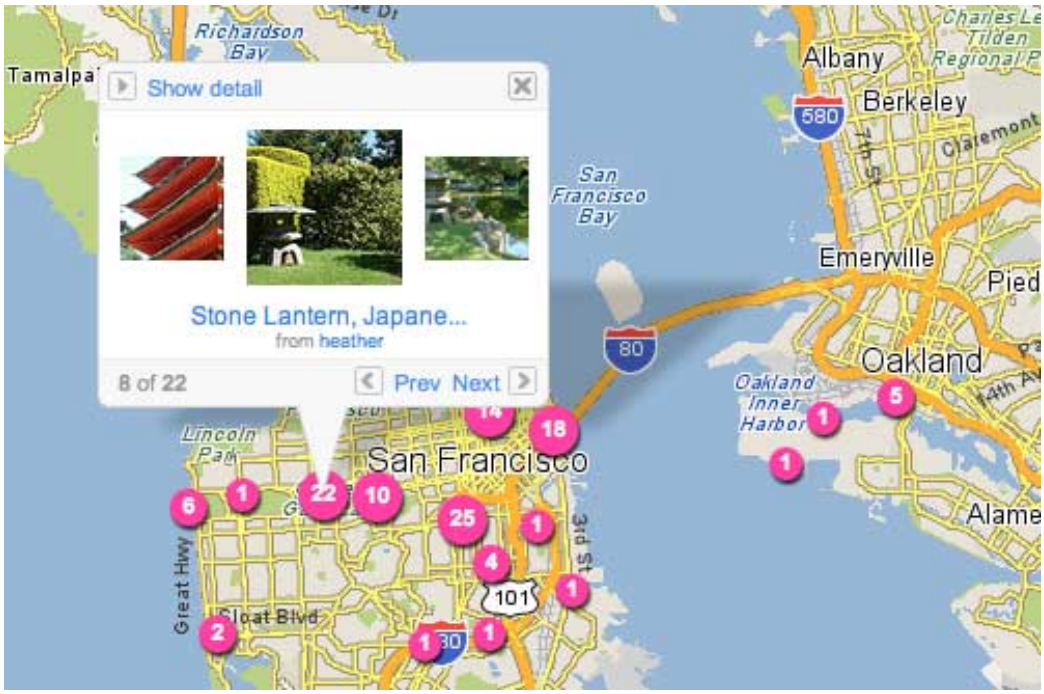

Figure 3.4: Geotagging can be used to create maps with pictures assigned to various places

Chapter 4

Used technologies

4.1 XPath

Current version of XPath is XPath 2.0[W3C07a]. This language was defined by World Wide Web Consortium(W3C). It became a recommendation on 23 January 2007. XPath is used for addressing, selecting parts of XML document.

XPath deals values as sequence of nodes and atomic values. Nodes are XML constructs such as elements, attributes and text nodes. XPath deals XML document as a tree, and addresses its part through selecting nodes on path from root. Root element is / and for example to select elements A, a child of B, under root, one will use $/A/B$ expression. Also, predicates can be used to filter selected nodes i.e. $/A[@B = "C"]$ will only select those children of root element, that has attribute B set to value "C". XPath in used as a part of XQuery and XSLT, both used for work with XML.

4.2 XQuery

XQuery[W3C07b] is a functional programming language, that was developed by XML query working group of W3C. Current version is 1.0 and it is in W3C recommendation since 2007. XQuery can be used both for retrieving and storing data in XML format.
XQuery works with XML tree like structure, and uses XPath expressions to select xml nodes. Language is strongly binded with XML, it uses nodes, attributes, list and XML atomic values, as defined in XML Schema. XQuery uses FLOWR expressions, to make SQL like SELECT from database. FLOWR stands for FOR, LET, ORDER BY, WHERE, RETURN. Queries are similar to SQL queries, although SQL is declarative language. Advantages of XQuery are allowing to work directly on XML structure, no chance to make mistake in XML structure, since both input and output of XML are valid XML documents, and also, this language is quite easy to learn, of course, for user that is already familiar with XPath. XQuery is really suitable for operations over XML, but it is not suitable for creating large applications, since it is primary developed as query language.

4.3 Php

Php[Gro04b] is a scripting language used mainly for creating dynamic web pages. Php was created by Rasmus Lerdorf in 1995 and currently being developed by PHP Group. With this working team change, name was changed from "Personal Home Page" to "Php: Hypertext Preprocessor". It is distributed with Php License, which is open-source license. Software can run on almost all platforms an operating systems. Php is server-side and as many other scripting languages is kept in human-readable form, so compilation is typically done at run-time by the Php compilers. Several methods can be

used for script execution acceleration, such as caching a compiled version of a script in memory[Lin02].
Php variables use $ prefix, and anything written with this prefix is considered as variable, because php variable type does not have to be given in advance. Script can be directly inserted into html, php only parses code between <?*php* and ? > delimiters, anything else is given directly as output. But for good practice, it is reasonable to keep controlling and visual representation of data apart. Best option is object-oriented programming that was introduced in php version 3 and completely rewritten in version 5, and is really useful.

4.4 XSLT

XSLT[W3C07c] stands for Extensible Stylesheet Language Transformations and it is developed by the W3C. The most recent version is XSLT 2.0, which reached W3C recommendation status on 23 January 2007. XSLT is used to transform one XML document to another. This technology uses style-sheets, that that are given along with input XML to a XSLT processor which outputs XML document transformed according to style-sheet. XSLT style-sheet itself, is a XML document. XLST is like functional programming language and also relies on pattern-matching and declarative approach. Typical style-sheet consists of several templates, that match certain document nodes. These matching patterns are written in XPath. Template processor takes document as a tree and starting from its root, applies templates that match currently processed node. Since XSLT works with XML, it can be used like CSS style-sheet, to give data visual representation, typically, it takes raw XML data as an input and outputs XHTML document.

4.5 Ajax

Ajax[Gar05], asynchronous JavaScript and XML, is a group of related web development techniques. Ajax is used on web pages to dynamically load data from server and display it, without reloading the page. This is done using XMLHttpRequest, a javascript object, that allows xml document to be loaded. Then, XML has to be parsed and added to document, using DOM. All parts of this process are client-side, but the source of XML can be anything, including server-side script. Advantages of Ajax count faster response, since only parts of page are reloaded, not a whole page, scripts and style-sheets are also loaded only once. On the other hand, usage of Ajax should be considered, because it relies on javascript, and without it, an internet application simply will not work. Also, programmer has to deal with dynamic character of pages, and should provide links for bookmarking and history search.

Chapter 5

Implementation and results

Implementation of tagbrowser was done for blog.matfyz.sk portal, a blog portal, that is being developed by faculty students, and is used for different purposes.

- **social**: students create their own blogs and write articles, express their opinions on school, or write just for fun. Also, portal is great information source about cultural life on faculty, and in community of mathematicians and programmers.
- **educational**: Portal is developed by students, and is used for testing and learning new technologies, also many master and bachelor thesis are done on the portal. Students also use portal for making surveys among portal users. There is even a course that teaches students webdesign, and main goal is to develop a blog, using XSLT transformation
- **scientific**: Apart from student development, portal uses native XML database Sedna, that is developed by Russian colleagues from Institute for System Programming at Russian Academy of Science. Portal provides workload that is necessary for debugging this database. Bug reports help developers to locate and repair problems is this system.

Portal is heavily based on modern approaches, and usage of XML, for storing data and transforming data to its visual representation. Main portal controller is written in php, and portal is divided into classes, each created to serve several tasks. For this work class TagBrowser was created. This class provides methods for retrieving both inline and arbitrary positioned tagclouds in xml format. This xml is then parsed using XSL transformation and presented to the user.

5.1 Retrieving data

All data is stored in XML native database Sedna and it was necessary to obtain them using XQuery. Portal itself is using php, and queries on Sedna are done by passing a query string to the Sedna controller class. To make work on tagging easier, we have implemented set of XQuery functions for basic tag operations, that formed a string, that was then concated to all the queries done by TagBrowser class. Tags are retrieved from database in 3 steps:

1. Each article in database has a list of tags assigned to it. Tags of all articles(or articles of certain author) are selected, along with the articles ID

2. Tagging system on portal holds tag database with canonical representations of tags, to eliminate plural and synonym problems. Selected tags are looked up in this database, so their canonical representations along with post ID are ready for further computation.

3. Tags are aggregated according to their canonical representation, and final XML is a list of tags, each provided with count attribute, and list of articles tagged with this tag.

This XML holds all information needed to create inline tags(and to create a list of articles tagged with certain tag), but has no information on relations. Also, since Sedna was very slow for this task[1], caching was the only option. TagBrowser has a script, that is writting XML for each of portal languages into external file. This script can be scheduled with cron, to run in periodical time intervals. When TagBrowser is tasked to return relational tagcloud for selected tag, this external file is opened, and its data is given again, as an input for XQuery. XML we have created allows us to easily retrieve related tags, since related tags share a common article, and every tag has a list of articles assigned to it.

5.2 Force-directed layout

In order to visualize tag strength(importance) and also relations, it was necessary to use tag cloud with arbitrary positions. For our purpose, best idea was to look at the tag cloud as a graph and display the layout of a graph. Layout(drawing) of a graph is a visual representation of graph, which is planar, and focuses on certain graph properties. Our graph should visualize tags as graph vertices and their relations will be shown by graph edges. Biggest challenge is that we only know that graph is connected. All other properties depend on relations of tags. Since there is not any best graph layout, best approach for run-time graph visualization is heuristic. Also so we have to define properties that we want layout to have:

- uniform edge length
- scalable layout, that can be fitted into web page
- nodes as far from each other as possible, to achieve good legibility

[1] experimental time was approximately 5 seconds

- aesthetic standards

From possible solutions, such as spectral layout, orthogonal layout, symmetric layout etc. We have chosen force-directed layout, because of good result properties. Force-directed layout[JM02](also called spring-algorithm) is a heuristic algorithm for graph drawing, that is based on physical model. Given a graph $G = (V, E)$, this approach deals vertices as electrically charged particles, and edges as springs. Algorithm simulates dynamics of this system over time. Since vertices are electrically charged, they repel each other according to simplified Coulumb law

$$F = \frac{q_1 q_2}{r^2}$$

Where q_1, q_2 are vertice sizes(weight) and r is vertice distance. And with edges as springs, they are contracting and pushing nodes closer to each other according to Hooke's law

$$F = k\mathbf{x}$$

Where $\mathbf{x}$ is distance from equilibrium position, k is spring constant, in our case same constant for all edges. This equations provide systems energy, and constants have to be properly set up, so that there exists a equilibrium between these forces. Algorithm can be schematically written as follows:

```
1 initialize node velocities to (0,0)
2 initialize node positions randomly //also, we can use some more
     sophisticated positions, for example circle
3 loop
4   total_kinetic_energy := 0 // running sum of total kinetic
       energy over all particles
5     for each node
6       tmp-force := (0, 0) // running sum of total force on this
           particular node
7
8       for each other node
```

```
9           tmp-force := tmp-force + Coulomb_repulsion( this_node,
                other_node )
10
11        for each edge incident to this node
12          tmp-force := tmp-force + Hooke_attraction( this_node,
                edge )
13
14        this_node.velocity := (this_node.velocity + timestep * tmp
              -force) * damping
15        this_node.position := this_node.position + timestep *
              this_node.velocity
16        total_kinetic_energy := total_kinetic_energy + this_node.
              mass * (this_node.velocity)^2
17  until total_kinetic_energy is less than some small number
```

System is moving according to specified rules, until total kinetic energy is small enough. This means that the system has reached one of its local energy minimal configurations.

Damping is introduced, because without it, for some special configurations, system could move forever.

Clearly in every iteration, we have to calculate all the forces that apply to vertices:

$$F(v) = \sum_{v' \in V; v' \neq v} CoulombRepulsion\,(v, v') + \sum_{e \in E; v \in e} HookeAtraction\,(v, e)$$

For a graph with N vertices and M edges, this results in computational complexity of $O\,(N^2 M)$ for each iteration.

Best technology for this algorithm implementation among those used by portal, was php. From version 5 php supports object-oriented programming, so it was possible to create universal classes for graph drawing. This small library consists 3 objects, Node(vertice), Edge, and Graph itself. Classes Node and Edge were basically serving as data holders, Graph was calculating everything. Algorithm was directly coded into php, with one difference,

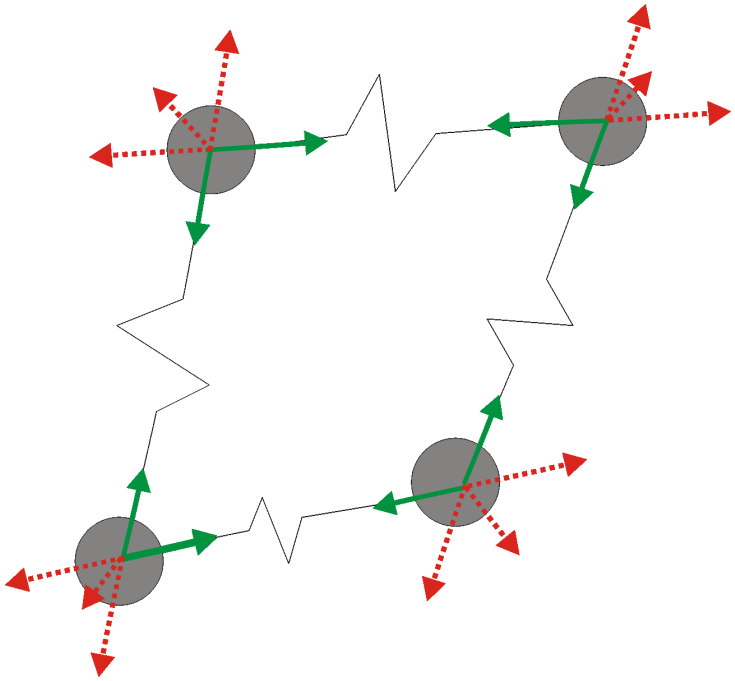

Figure 5.1: Graph layout dynamics. Red dotted lines represent repulsive forces, green lines represent spring forces.

maximum number of iterations was set to 500, to stop long running calculations[2].

To work with the library, most important is class Graph. Once created an instance of this class, use this methods to create graph layout:

- **addNode** - specify unique node name, and possibly node weight
- **addEdge** - specify incident nodes, possibly length
- **randomize** - randomizes node positions, this is used because algorithm needs nodes to have different positions

[2] As far as we know from test results, number of iterations does not exceed 100

- **normalize** - resizes and moves layout so that is completely using square $(0, 1)^2$
- **step** - one iteration step, probably useful, when trying to create animation, also for debugging
- **make** - starts spring-algorithm, iterates until kinetic energy is small enough

This library(set of classes) is suitable for other uses, since it works only with abstract graph data. As future work, we want to further develop this library and implement other features, since php graph drawing is not very common. To create a graph layout, we had to make five steps:

1. add nodes using **addNode**. Nodes related to selected node were retrieved from cached xml, and then only most important were added.[3]
2. add edges using **addEdge**. All nodes(tags) were checked for relations, and edges were added. Since tag relation as we defined it is symetric, graph class have to have duplicate edge removal mechanism.
3. **randomize** to prepare for calculating.
4. **make** to calculate graph layout.
5. **normalize** to make data ready for displaying

Force-directed layout as we defined it, only works for connected graph. Otherwise, repelling forces moves graph partitions away from each other to infinity. Once graph layout is calculated, results are given in form of xml to the page generator and are further transformed using XSLT.

[3]for good legibility maximum of 15 nodes is reasonable

5.3 Arbitrary tag positions and html

Our tag cloud is represented by graph layout, and tags can have almost any positions. We have chosen absolute positioning to move elements(divs) containing tags. One div, that can have any size, contains this layout, that is absolutely positioned in percent of containing div. This allows page to be resized and customized properly. Another problem is to render edges, using html elements. Html and css have mechanisms to draw vertical or horizontal lines[4], but other lines can not be rendered, and have to be drawn using css and absolute positioning. I have chosen a technique of drawing lines using 1 pixel divs. This method produces lot of divs, that slow down page rendering, but unlike other methods, for example image scaling, lines are acurate and thus aesthetically pleasing. All this was achieved by only using html and css, so whole tagbrowser can run without javascript.

5.4 Results

To demonstrate final layout of tag cloud we will provide some images showing algorithm result, on real data taken from blog.matfyz.sk .

[4]for example using borders

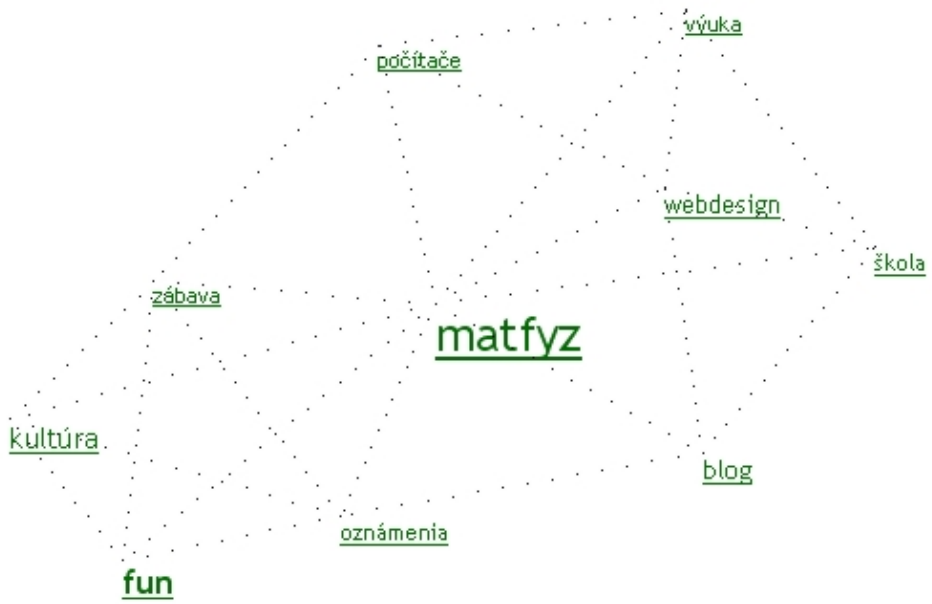

Figure 5.2: Typical cloud of one of the most important tags on portal, matfyz.

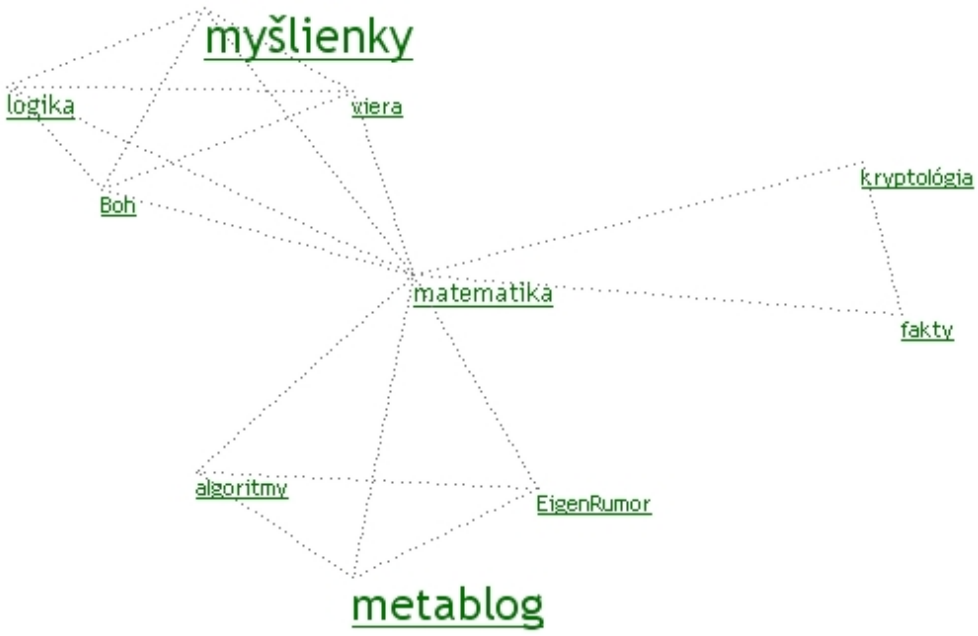

Figure 5.3: This cloud is a great demonstration of how folksonomy creates categories and taxonomy system. We can see, that tags related to "matematika" fall into 3 different cathegories(according to their relations) and this tag connects them.

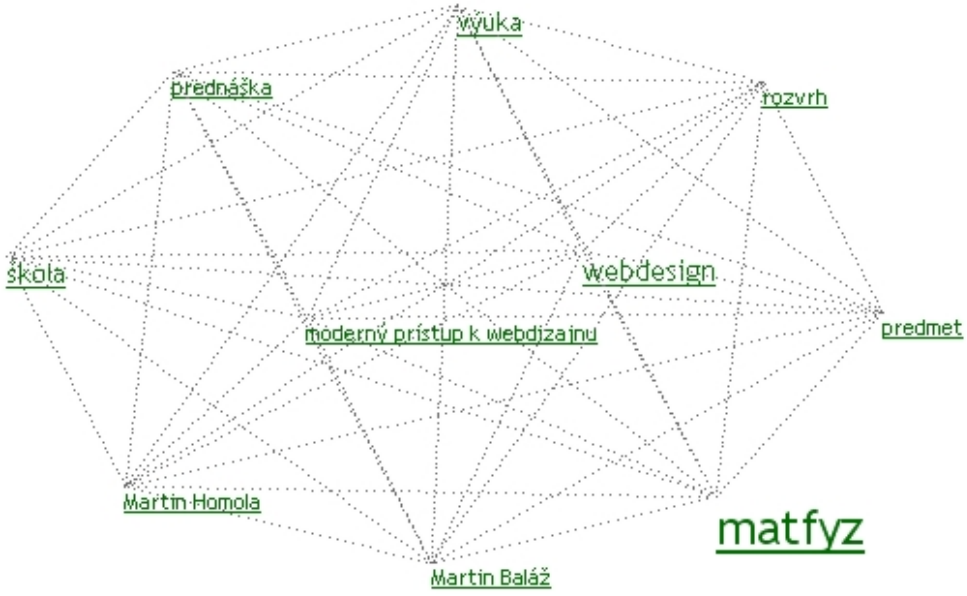

Figure 5.4: Accidentally we have also created a complete graph. This can be eliminated by visualizing only strong relations, not all.

Chapter 6

Conclusion

We have studied and written some technique and algorithms for visualization of tags and tagspace. Further more, we have created model of tagspace navigation system, and implemented important parts of it for blog.matfyz.sk. As a side result, we have created functions for generating an inline tagcloud, using different scaling techniques, that is being used for title tagcloud and also, tagcloud of users. Also experiments show that same technique can be used for large(title) and small(user) tagclouds.

Main result of this thesis is relational tagbrowser, that visualizes part of the tagspace around one selected tag, using graph layout visualization. We have proven that this approach is suitable for this task, and that resulting tagclouds have high aesthetic, usability and also html validity standards. This application is very modern and still not very common among tagging systems used on the internet.

Tagbrowser will serve users of blog portal, and also can be used for research on tags. Inline tagclouds are already used by the portal, and working well.

Force-directed layout was implemented in many languages, and there exists many libraries dealing with this problem. But php implementations are not very common, so this library can be useful for many other programmers and

students. This library was created for this special task, but after a little afford, it can be completely universal. It will be completed and published on the internet, free for others to use.

As a future work, relational tag browsers are still not very much used, and there is a lot of work to be done, either researching new methods and visualizations, or implementing tagclouds in different languages used on the internet. Furthermore, tagging on blog.matfyz.sk has many features, but navigation model introduced in this thesis will be considered and tagging navigation will be remade, to complete navigation possibilities on portal.

Bibliography

[Gar05] Jesse James Garrett. *Ajax: A New Approach to Web Applications*. 2005.
http://www.adaptivepath.com/ideas/essays/archives/000385.php.

[Gro04a] RDF Core Working Group. *Resource Description Framework (RDF)*. 2004.
http://www.w3.org/RDF.

[Gro04b] The PHP Group. *PHP Documentation*. 2004.
http://www.php.net.

[JM02] Fabien Jourdan and Guy Melançon. *A scalable force-directed method for the visualization of large graphs*. 2002.
ftp://ftpdim.uqac.ca/pub/ychirico/wvdr2002/jourdan.pdf.

[Lin02] Nick Lindridge. *The PHP Accelerator 1.2*. 2002.
http://www.php-accelerator.co.uk/PHPA_Article.pdf.

[LK07] Daniel Lemire and Owen Kaser. *Tag-Cloud Drawing: Algorithms for Cloud Visualization*. 2007.
http://www2007.org/workshops/paper_12.pdf.

[Mat04] Adam Mathes. *Folksonomies - Cooperative Classification and Communication Through Shared Metadata*. 2004.

http://www.adammathes.com/academic/
computer-mediated-communication/folksonomies.html.

[Sif06] D. Sifry. *State of the Blogosphere*. 2006.
http://www.sifry.com/alerts/archives/
000436.html.

[Smi08] Gene Smith. *Tagging: People-Powered Metadata for the Social Web*. New Riders, 2008.

[W3C07a] W3C. *XML Path Language (XPath) 2.0*. 2007.
http://www.w3.org/TR/2007/REC-xpath20-20070123/.

[W3C07b] W3C. *XQuery 1.0: An XML Query Language*. 2007.
http://www.w3.org/TR/2007/REC-xquery-20070123/.

[W3C07c] W3C. *XSL Transformations (XSLT) Version 2.0*. 2007.
http://www.w3.org/TR/2007/REC-xslt20-20070123/.

[XFMS06] Zhichen Xu, Yun Fu, Jianchang Mao, and Difu Su. *Towards the Semantic Web: Collaborative Tag Suggestions*. 2006.
http://www.semanticmetadata.net/hosted
/taggingws-www2006-files/13.pdf.

Abstrakt

Tagovaním (pridávaním metadátových kľúčových slov) internetového obsahu vzniká takzvaná folksonómia, zdola budovaná taxonómia, popisujúca vzťahy dátových objektov. Táto taxonómia môže byť použitá ako forma navigácie, napriek tomu že neslúži na vyhľadávanie odpovedí na otázky, ale na hľadanie nových otázok a zaujímavého internetového obsahu.
Agregáciou tagov podľa ich mena vzniká navigačný element nazývaný tagcloud. Táto práca uvádza základné metódy generovania riadkových tagcloudov, takisto definuje príbuznosť tagov, pre zlepšenie navigačného modelu v tomto priestore tagov. Metódy generovania tagcloudov sú porovnané a experimentálne otestované implementovaním týchto algoritmov a ich následným použitím na reálnych dátach. Ako hlavný prínos prezentujeme relačný tagbrowser vytvorený pre portál blog.matfyz.sk, ktorý využíva kreslenie grafov na vizualizáciu tagov a vzťahov medzi nimi. Algoritmy a technológie ktoré viedli k vytvoreniu tohto tagbrowsera sú popísané spolu s implementačnými detailami.

Kľúčové slová: tagy, relačný tagcloud, kreslenie grafov