



Seminar

Algorithmic Methods in the Humanities

Summer 2016

Algorithmics I
Institute of Theoretical Informatics
Department of Informatics
Karlsruhe Institute of Technology

Preface

Digital humanities is an area of research at the intersection of computing and the disciplines of the humanities, e.g., history, philosophy, linguistics, literature, art, archaeology, music, as well as cultural and social sciences. Multiple computer science fields have developed computational methods that are successfully used in the humanities sciences. The goal of the seminar *Algorithmic Methods in the Humanities* was to get a glimpse on the algorithms that lie at the core of these computational methods. In particular the seminar considered the topics of Visualization in the Humanities, Analysis of Textual Variation, Topic Labeling and Topic Recognition. The seminar is the part of the Master's program on Informatics at KIT. It is an interdisciplinary seminar supervised by Institute of Theoretical Informatics and Institute of Philosophy and was intended to the students from both Institutes. The aim of the seminar was to involve the students in the scientific research. They learned how to conduct literature research, identify, classify existing approaches and critically evaluate the literature. They practiced in devising a presentation in the context of a scientific topic in a way suitable for the audience. Finally they produced an extended written summary of the methods that they had identified; and tried to do it in a structured way, complying with the standards of scientific survey papers. This booklet is the collection of these summaries. The seminar took place in the summer semester 2016 and had 10 participants. The booklet contains the summaries of 7 participants from the Institute of Theoretical Informatics.

1 Contents

Visualizations for Distant Reading – Map Representations of Relational Data Sets	4
Estimating Similarity of Notions using Google	23
Text Variant Graph and its construction	36
Block Edit Models for Approximate String Matching	49
Latent Dirichlet Allocation	64
Syntax Trees and Syntax Tree Drawing	78
Graph-based Topic Labelling Using DBpedia	91

2 Visualizations for Distant Reading – Map Representations of Relational Data Sets

Sophie v. Schmettow

Abstract

In an age of big data and an abundance of electronic textual information one question is of central importance: How to make huge quantities of information visually accessible while preserving as much content and its underlying structural relationships as possible? How to do so in an intuitive and even aesthetic way? This paper presents two applications designed to help digital humanities scholars navigating through various types of large textual databases. First, the Trading Consequences project which is concerned with worldwide commodity trading in the 19th century. Second, the Herodotus Space-Text-Imaging Archive, which supports an advanced way of studying one of the world’s first historical narratives. Furthermore, two approaches to data visualization are examined; on the one hand, proportional symbol maps used to display statistical data linked to geographic locations and on the other hand, pseudo-geographic map representations of graphs which use fictional countries to communicate proximity information in an explicit manner.

2.1 Introduction

In recent decades, numbers and figures have pervaded more and more scientific disciplines. As of late, the literary sciences seem to have been spared but this has changed with the digitalization of whole libraries and full-text archives. Traditionally, literary scholars have studied texts by **Close Reading**, defined as “A thorough interpretation of a text passage by the determination of central themes and the analysis of their development” [6].

Franco Moretti is a professor of literature at Stanford University. In his book *Graphs, Maps, Trees: Abstract Models for Literary History* [9] he points out an obvious problem in this approach: “Knowing 200 novels is already difficult. *Twenty thousand?* How can we do it, what does ‘knowledge’ mean, in this new scenario?” In the scope of this endeavor he coined the term **Distant Reading**, referring to the process of reviewing large quantities of text, dissolving the actual phrase structure and extracting important meta-aspects of texts. According to [6] it is used to “Generating an abstract view by shifting from observing textual content to visualizing global features of a single or of multiple text(s).” A recent article in the Sunday paper *Welt am Sonntag* [11] discusses the works of Moretti in the Stanford Literary Lab, Martin Grandjean at the University of Lausanne and Andrzej Kulig of the Institute of Nuclear Physics of the Polish Academy of Sciences. The article calls Distant Reading “The latest drug of literary science.” It further declares that, given that in the 19th century in Britain alone about 50,000 novels have been published, literary science practically only covers a very restricted part of its subject. Thus, a more wholesome literary science indeed calls for supplementation by automated quantitative analyses. In order to work with the mined data, a clearly structured and preferably aesthetically pleasing visual representation is essential. As the article also states, numbers quickly grow very big – Shakespeare’s work alone features 1,200 characters and more than 880,000 words of dialogue. Accordingly, creating representations that are as readable as possible without unnecessarily reducing the data’s complexity or creating a misleading picture is a major challenge. Cutting-edge methods include graphs, timelines, taxonomies and maps. The latter are an especially intuitive way

to represent relational data, since most people are familiar with maps and enjoy meticulously studying them.

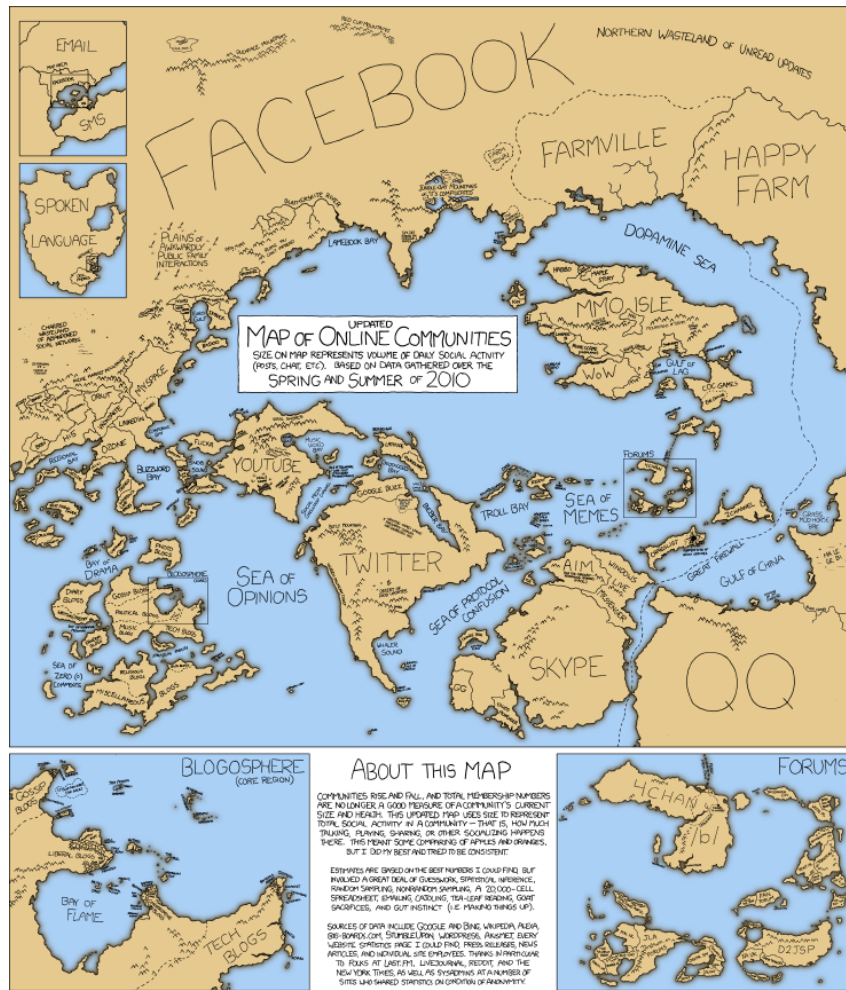
Traditional cartography deals with representing real-world geographic locations as accurately as possible. However, maps are a powerful means to display additional spacial information as well. For instance, choropleth maps make use of color schemes to visualize statistical data such as population densities etc., area cartograms depict area as proportional to some external size rather than the actual area, and Dorling cartograms are composed of discs with size proportional to the dimension of interest. Maps are frequently used to represent imagined places as well. Most major fantasy works feature elaborate maps of the fictional worlds they create.

Figure 1 shows online cartoonist xkcd’s approach to using cartograms as a means to visualize the extent of social activities throughout various popular internet communities: size represents the degree of daily socializing and proximity indicates a somewhat arbitrary metric of similarity [10]. As indicated by two maps superimposed on the top left, online communities make up a relatively small part of overall human communication, as the online-community “country” is surrounded by the much larger “continents” *E-Mail*, *SMS*, and *Cell phones* which, again, are part of the huge landmass of *Spoken Language*.

The *Facebook Region*, being the most spacious “country” includes numerous interesting geographical features such as the *Plains of Awkwardly Public Family Interactions* or the *Privacy Controls* surrounded by a lava pool – a subtle critique referring to how difficult it is to change your account’s privacy settings. Another big “country” is *YouTube* with the picturesque *Viral Shores* and the *HTML5 swamps*, subtly alluding to YouTube’s ragged HTML5 support (at the time the comic was created). Other large “land masses” include *Twitter Land* and *Skype Land* as well as the *QQ Region*. Tencent QQ is a frequently used Chinese instant messaging program. The large “country” has a famous “landmark”, the *Great Firewall*, which addresses internet censorship in China. In a nutshell, this map of an imaginary world is used to convey information on multiple levels in an accessible and humorous manner. This paper reviews state-of-the-art distant reading techniques based on maps. First, two real-world applications ([5], [1]) that enable distant reading of certain text corpora are presented, both of which make use of geographic maps. Among others, they employ a popular type of representation called *Proportional Symbol Maps*. This technique will be detailed with regard to algorithmic problems that arise, as well as approaches to solving them (as devised in [2]). The third section contrasts these actual geographic map representations with pseudo-geographic ones ([3]) and examines how geography can be used as a metaphor to depict relational data sets, as in the comic described above. The paper concludes with a discussion.

2.2 Maps & Timelines: Querying along Temporal, Spatial and Conceptual Dimensions

In this section, two map-based tools for digital humanities scholars are presented, which are especially useful since they allow the user to shift interactively between distant and close reading. Once distant reading techniques have helped the user to locate the exact passages of interest he can “zoom in” and read the desired parts.



■ **Figure 1** xkcd's Map of Online Communities.

2.2.1 Trading Consequences

The Trading Consequences project [5] is an interdisciplinary cooperation between historians, computational linguists and data visualization specialists. It aims to shed light on worldwide commodity¹ trading in the 19th century. Environmental historians are enabled to explore the overall prevalence and importance of goods as well as geographic and temporal contexts in which they were to be encountered. Furthermore, they may navigate through large corpora of historical documents to locate the exact spots where their commodities of interest are mentioned. As the authors put it, they want to “bring[...] the archives alive in ways that the authors of original documents would have never imagined.” The aims are to “support[...] visual querying along spatial, temporal, and conceptual dimensions” as well as to “highlight[...] trends within a range of document data, for instance, relations between different commodity types.”[5]

¹ raw materials, natural resources, and processed goods

The web-based interface² provides three interlinked visualizations. Users can either search for a location or a specific commodity. Figure 2 shows the results of an inquiry about “Gold”. A **proportional symbol map** helps to explore geospatial references to commodities related³ to “Gold”. A vertical **tag cloud** shows the 50 goods that co-occurred most frequently with “Gold” (with font size encoding the number of concomitant mentions). To the right, there is a **ranked list** of documents in which “Gold” was mentioned. A **bar chart** illustrates the temporal distribution of these documents within the corpus. The user is granted direct access to the documents when clicking on a certain document, with the commodity of interest being highlighted. Thereby, distant reading is combined with close reading. Users may also explore a **heatmap** (Figure 3a) that illustrates the geospatial context of “Gold” trade (or any other commodity in the database) or examine a **location cloud** which maps location mentions across time (for a selected commodity, Figure 3b).

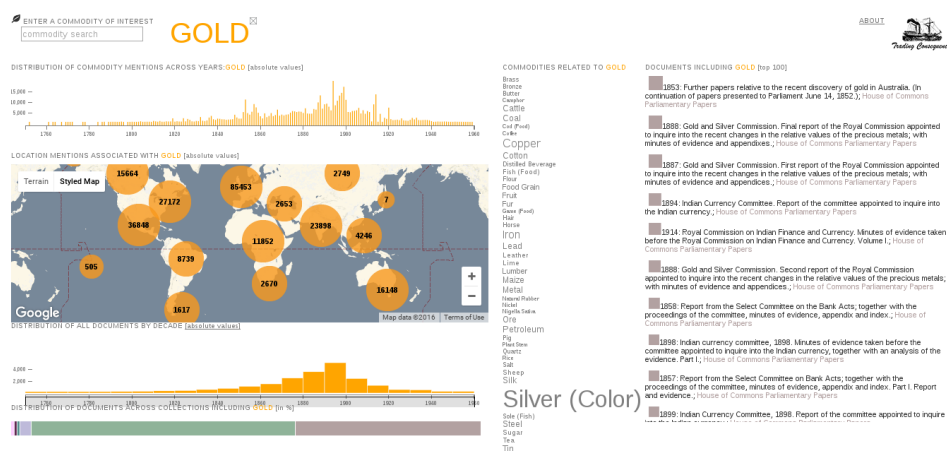


Figure 2 Inquiry about the commodity “Gold” in the Trading Consequences search interface.

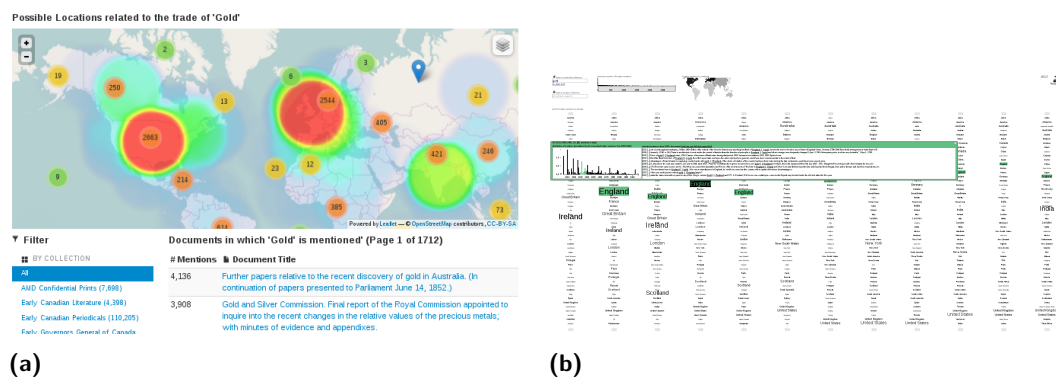


Figure 3 Different visualization techniques. The geographic distribution of “Gold” trade is indicated by a heatmap (a) and its temporal distribution is shown in a location cloud (b).

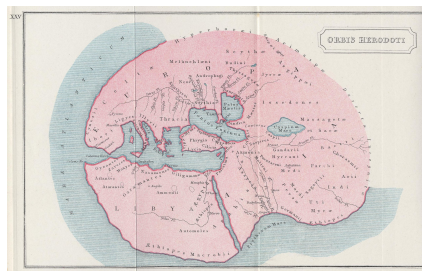
² <http://tcqdev.edina.ac.uk/vis/tradConVis/>

³ with “related” meaning that commodities were mentioned on the same page of a document

2.2.2 The Herodotus Encoded Space-Text-Imaging Archive (HESTIA)

The Herodotus Encoded Space-Text-Imaging Archive (HESTIA⁴) is a 2010 project [1] funded by the *Arts and Humanities Research Council* of the United Kingdom. Primary objective was to revive the magnum opus of Herodotus of Helicarnassus, one of the world's first historians. In his *Histories*, written in 440 BC, he pursues a complex question: Why did the Greeks (Herodotus' people) come into conflict with the Persians? In answering this question he travels all around the known world of the time. In the end, the work comprised nine books and covered a period of 220 years, describing the unraveling of history from 700 to 479 BC leading up to the Persian Wars.

Herodotus' world is often depicted in a similar fashion to that shown in Figure 4. While there



■ **Figure 4** Atlas of Ancient and Classical Geography.

is nothing wrong with such a static depiction, the authors of [1] attempted to turn the world of fifth century BC readily accessible for modern-day users, enabling them to investigate the cultural geography of the ancient world through Herodotus's eyes. The Perseus online library⁵ provides a digital text of the *Histories* with the place names mentioned in the narrative already annotated. This was fed into **modern web mapping devices** such as Google Earth. Deploying the latest satellite imaging openly available from NASA, humanities scholars, students, and researchers can surf the world and zoom in on any place to query what Herodotus has to say about it. For example, one might go to Babylon and explore his thoughts on Babylonian temple prostitution (Figure 5), creating a feeling of temporal proximity.

HESTIA provides a variety of tools, some of which integrate close and distant reading. Figure 6 shows a platform called HestiaVis⁶ where text and maps can be read alongside each other. The data can be subdivided into three categories: settlements, territories, and natural features of the environment, which can be depicted individually.

One more central objective of HESTIA was the construction of **network maps** to capture relations between places in order to challenge a polar worldview – East vs. West, Greece vs. Persia. These networks illustrate an interlinked world and reveal the complex connections that Herodotus draws. Figure 7 contrasts the **timeline view** with a **network view** for the place *Delphi*. Furthermore, the data set can be used to generate maps in GIS (Geographic Information System). Figure 8 shows an illustration that visualizes the number of references to settlements mentioned in the *Histories*. It was composed in QuantumGIS⁷.

⁴ Incidentally, Hestia is also a Goddess in ancient Greek religion, daughter of Cronus and Rhea and sister of Zeus.

⁵ <http://www.perseus.tufts.edu>

⁶ <http://www2.open.ac.uk/openlearn/hestia/index.html#index>

⁷ <http://www.qgis.org/>

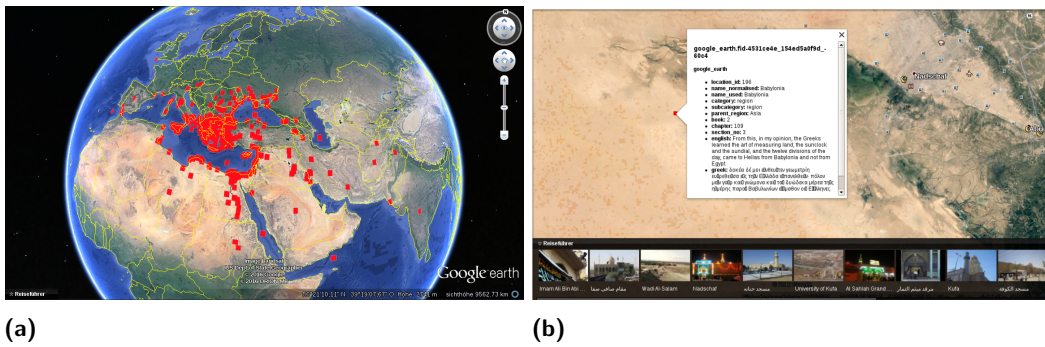


Figure 5 HESTIA in Google Earth. A bird's eye view is given in (a), then the user can shift from distant to close reading (b).

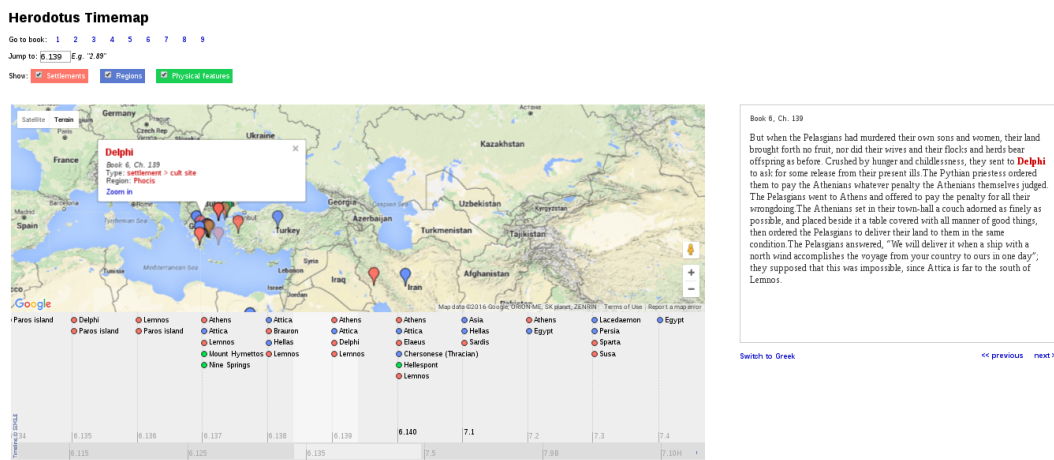


Figure 6 Combining Close & Distant Reading, Temporal and Spatial Dimensions.

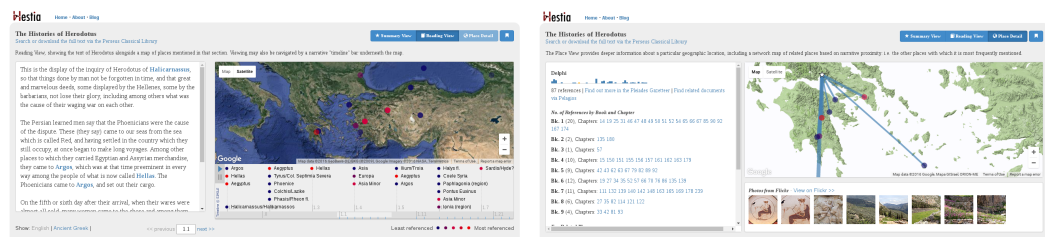
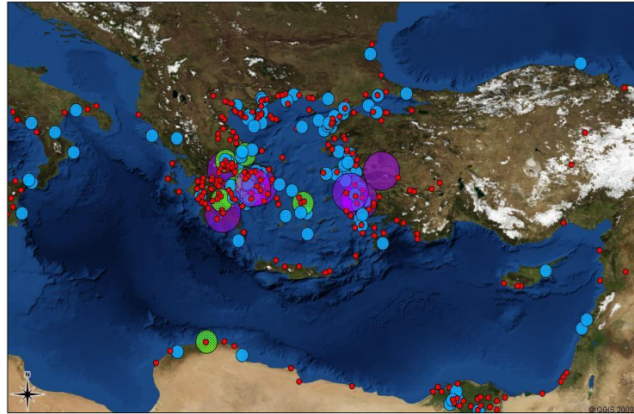


Figure 7 Timeline view vs. network view in HestiaVis.

2.2.3 Algorithmic Problems with Proportional Symbol Maps

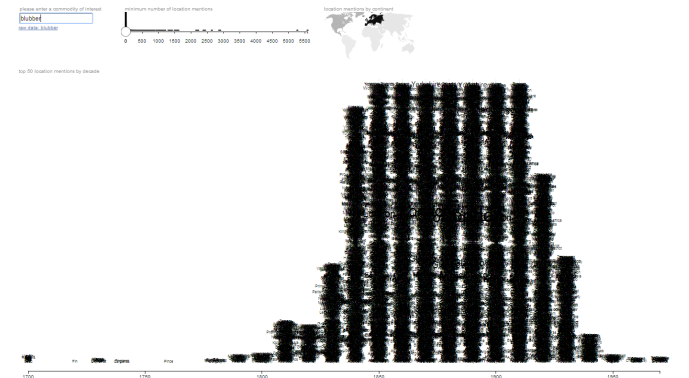
The two projects that were introduced in the previous passages provide real-world tools that are actually employed by humanities scholars. In these applications, several algorithmic problems arise: graph drawing problems (Figure 7), geovisualization problems (Figure 5), and placing labels to minimize overlapping (among others).

Barker et al. [1] focus heavily on the computer-based technology that was necessary to extract spatial data from the *Histories*, in which ways the resulted database can be queried and how humanities scholars can benefit from it. Klein et al. [7] mention the problem of



■ **Figure 8** A proportional symbol map visualizing the number of references of settlements mentioned in the *Histories*.

overlapping location labels: Figure 9 shows all location mentions associated with the label *blubber*⁸. Heavy cluttering makes it impossible to decipher any general trends, rendering the representation useless. This is addressed solely by limiting the amount of information displayed (i.e. only top 50 hits on map, or only decades with most mentions on timeline). However, one may algorithmically optimize visualization techniques in order to get the most



■ **Figure 9** Cluttering labels.

out of the data. This section shall focus on how to do so in the case of **proportional symbol maps**. Proportional symbol maps are a type of thematic maps that is well-established when it comes to the visualization of quantitative data associated with specific locations. They use simple symbols (in our case discs) placed on various locations of interest on a map and scaled proportional to the data value found at the respective location. This is to say, the larger the symbol, the more of the metric in question exists at that place. Thus, its spatial distribution can be examined. The map generated with the HESTIA data set in QuantumGIS (Figure 8) is an example of such a map. The message is put across by means of the symbol areas or rather the ratio between symbol areas. Three types of symbol scaling are considered in [2]. In mathematical scaling areas are directly related to the data. Perceptual scaling

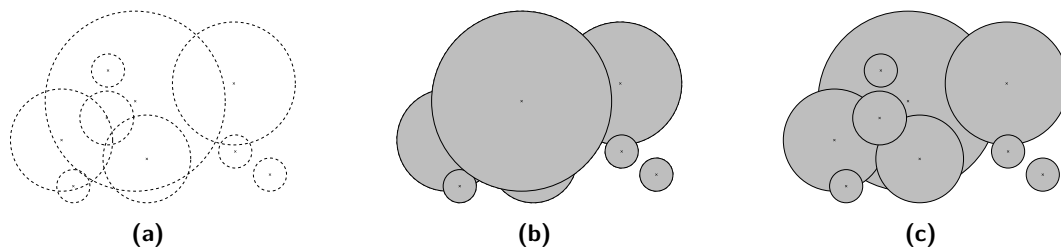
⁸ fish fat, apparently a very important good

takes into account that humans are not proficient at estimating relative sizes (large ones are often underestimated). Thus, area differences are deliberately exaggerated to smother these perceptual pitfalls. In range grading data are subdivided into classes, where symbols of one class have the same area.

The problem of finding the ideal size for the symbols on a map has already been studied extensively: a map should not appear too crowded; however, if the symbols are scattered too scarcely spatial patterns are not easily detected.⁹ Applying range grading or even logarithmic scaling prevents disks connected to high prevalence of the metric in question from overlapping smaller ones, but also makes the map much less easily accessible. Hence, mathematical scaling and perceptual scaling are preferable but one has to determine the right degree of overlapping. User studies have shown that the degree of overlap in a map using opaque symbols is – unsurprisingly – strongly correlated with the error of estimation of respective sizes.¹⁰ Transparent symbols might seem like a simple work-around but users tend to prefer opaque symbols, since the contrast to the underlying geographical map is more sharp.¹¹

PSM are frequently used in both projects. However, optimization with respect to the results stated above is called for: In *Trading Consequences*, the symbols are not scaled directly proportional to the external size in question (Figure 2). Thus, the distribution is not visible at first glance. Instead, the reader has to study the numbers by which the disks are labeled which is pretty cumbersome. The map shown in Figure 8 is rather crowded.

In order to talk about PSM a few definitions and notations have to be introduced: We denote by \mathcal{A} the arrangement formed by the disk boundaries of a set S of n disks D_1, \dots, D_n in the plane. A vertex v is formed by the intersection of the boundaries of two disks. We denote by \mathcal{D} a drawing of S , which is a subset of arcs and vertices of \mathcal{A} drawn on top of the filled interiors of the disks in S . Figure 10 shows an arrangement of a set of disks along with one unsuitable and a preferable drawing of this same set. But what makes a *good* drawing? It should enable the viewer to see at least some part of all symbols and to judge their sizes as correctly as possible! As already mentioned, empirical studies have shown that the accuracy with which the size of a disk can be judged is proportional to the portion of its boundary that is visible. This is illustrated in Figure 11a. Accordingly, two quality criteria can be defined.



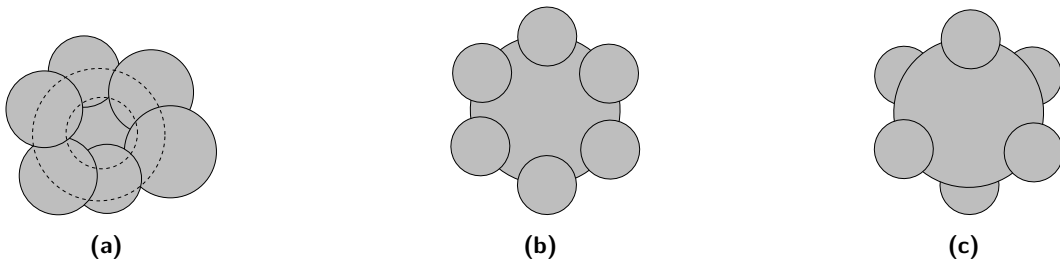
■ **Figure 10** An arrangement \mathcal{A} of disks (a); a *bad* drawing (b) and a *good* drawing (c) \mathcal{D} of \mathcal{A} .

1. **MaxMin:** Find a drawing that maximizes the minimum visible boundary length of each symbol.
2. **MaxTotal:** Find a drawing that maximizes the total visible boundary length over all

⁹ Terry Slocum, Robert McMaster, Fritz Kessler, and Hugh Howard. Thematic Cartography and Geographic Visualization. Prentice Hall, 2nd edition, 2003 (as cited in [2]).

¹⁰ *ibid.*

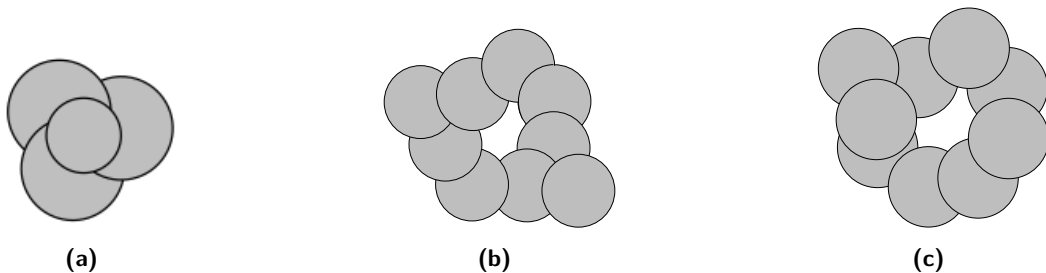
¹¹ Trevor Griffin. The importance of visual contrast for graduated circles. *Cartography*, 19(1):21–30, 1990 (as cited in [2]).



■ **Figure 11** Quality criteria for PSM. Visible perimeter is more central to correct judgment than visible area as illustrated by (a). Disks stacked according to the *Max-Min* criterion are shown in (b) and a stacking of the same disks according to the *Max-Total* criterion is shown in (c).

symbols.

Figure 11b and Figure 11c show two drawings of the same arrangement implementing the different criteria. Two kinds of models are considered in order to implement these criteria: First, **physically realizable drawings**, which are – to put it simple – drawings that could be reproduced with coins, and second, **stacking drawings**, which are a subset of the set of physically realizable drawings restricted by demanding that there exists a total order on the disks (the so-called stacking order). Figure 12 contrasts various models. Figure 12a shows a non-physically realizable drawing characterized by some – as the authors of [2] call it – “Escher-like” qualities. Such a layout is obviously not desirable in a proportional symbol map since it could easily generate confusion as to where the disks’ boundaries are in a more complex arrangement. Figure 12b and Figure 12c show a physically realizable and a stacking drawing. In order to grasp the difference, one may imagine trying to reproduce these images with coins. For the physically realizable drawing in Figure 12b one could – for instance – start with the topmost coin and proceed clockwise. The last coin would then have to be slipped under the starting coin. By contrast, when creating the stacking drawing in Figure 12c one can find a stacking order where no such slipping operation is necessary. Or – to put it differently – if the drawing was made up of coins, one could find a way to remove them one by one without having to move any other coin still on the stack.



■ **Figure 12** A drawing that is not physically realizable (a), a physically realizable drawing (b), and a stacking drawing (c).

With the help of definitions and concepts introduced above, the algorithmic problem posed by a PSM can now be formally defined:

► **Definition 1** (Proportional Symbol Map). Given a set of points $P = \{p_1, \dots, p_n\} \subset \mathbb{R}^2$ and a set of corresponding values $W = \{w_1, \dots, w_n\} \subset \mathbb{R}^+$ (as well as meaning attached to these values), find the order of disks D_i defined by (p_i, w_i) – center p_i and area w_i – s. t. a physically realizable drawing or a stacking drawing that either satisfies MaxMin or MaxTotal is produced.

It can be shown, that this problem is NP-hard for physically realizable drawings, meaning that it is computationally infeasible. Thus, one concentrates on a MaxMin Stacking Model. It can be proven that such a model can be computed in polynomial time for any set of general pseudo-disks (*convex* symbols, i.e. a set of symbols s.t. the boundaries of any two symbols intersect in at most two points) [2]:

► **Theorem 2.** *Given n disks in the plane, a stacking order maximizing the boundary length of the disk that is least visible can be computed in $O(n^2 \log n)$ time.*

The basic idea of a greedy algorithm to achieve this (given in Algorithm 1) is as follows: For each disk in the set you determine how much boundary would be visible if it were the bottommost disk ($vis(D_j)$). Then you place the disk with the maximum value at the bottom and recurse over the $n - 1$ remaining disks.

```

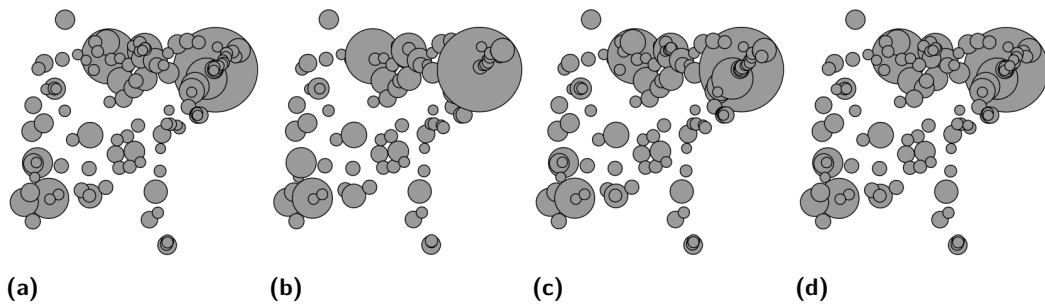
Input : A set of general pseudo-disks  $D = \{D_1, \dots, D_n\} \subset \mathbb{R}^2$ 
Output : A permutation  $\pi$  of  $\{1, \dots, n\}$  such that the proportional symbol map with
           stacking order  $D_{\pi(1)} < \dots < D_{\pi(n)}$  maximizes the minimum visible borders
           of  $\mathcal{D}$ 
initialize  $S \leftarrow D$ 
for  $i = 1, \dots, n$  do
    forall  $D_j \in S$  do
         $vis(D_j) \leftarrow$  vis. boundary of  $D_j$  if  $\pi(i) = j$ ;
    end
     $D_k \leftarrow \arg \max_{D_j \in S} vis(D_j)$ ;
     $\pi(i) \leftarrow k$ ;
     $S \leftarrow S \setminus \{D_k\}$ ;
end
return  $\pi$ 

```

Algorithm 1 : Stacking general pseudo-disks in near-quadratic time

The algorithm computes an optimal stacking: Let \mathcal{A} be the stacking produced by the algorithm and $\mathcal{A}[i]$ be the i^{th} disk of that stacking. \mathcal{A} can be compared to any optimal solution \mathcal{O}^0 . Let i be the smallest value s.t. $\mathcal{A}[i] \neq \mathcal{O}^0[i]$. \mathcal{O}^1 is obtained by moving $\mathcal{A}[i]$ in \mathcal{O} to index i . Analogously, stackings $\mathcal{O}^2, \dots, \mathcal{O}^m$ with $\mathcal{O}^m = \mathcal{A}$ and $min - vis(\mathcal{O}^k) \geq min - vis(\mathcal{O}^{k-1})$ can be iteratively constructed. Running time can be proven by maintaining a special data structure called a segment tree for each disk, where the disk boundary is considered to be a linear interval from its topmost point in clockwise direction. Intersections with other disks induce elementary intervals, which are stored in the leaves of the balanced binary tree. Internal nodes correspond to the union of their child intervals. Additionally, visibility can be stored for each node, where the visibility of the root node corresponds to the visible boundary of the respective disk if it were the bottommost disk.

Three much simpler heuristics to determine a stacking order, which arguably do not perform much worse, are mentioned in [2]. In the drawing shown in Figure 13b the disk with the leftmost center is put first and the remaining disks are drawn recursively on top. In Figure 13c this is done with respect to the left extreme instead of the center. In Figure 13d disks are drawn from bottom to top in order of decreasing radius which produces a unique stacking order given that the radii are distinct. The drawing in Figure 13a was produced using the algorithm outlined above. Since the heuristics require only sorting they can be calculated in linear time. Especially *large-to-small* still produces acceptable results.



■ **Figure 13** The max-min algorithm (a) as opposed to three different heuristics, left-to-right by center (b), left-to-right by leftmost (c), and large-to-small (d).

2.3 Geography as a Metaphor: A Visit to BookLand and other Curious Places

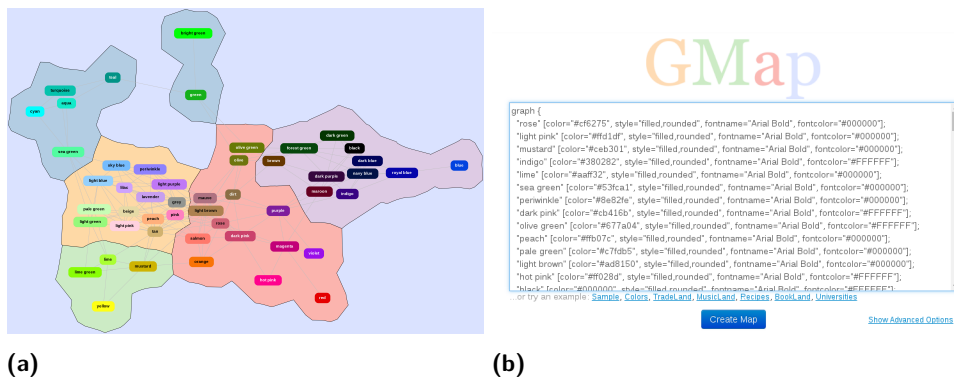
Not only statistical data that are linked to actual places in the world but any set of relational data can be visualized as a map. Relational data sets, in turn, can be described by a graph. A graph is a mathematical object that represents relationships between a set of items. More specifically, it is an ordered pair $G = (V, E)$, with V being a set of vertices and E a set of corresponding edges. Most commonly, vertices are drawn as points and edges are drawn as lines in two- or three-dimensional depictions. Since graphs, represented in this way, as well as charts and tables usually demand substantial effort to grasp, the authors of [3] propose a more intuitive and enjoyable approach to display big relational data sets: the use of 2D geography-like maps. This does not yield *accurate* but very intuitive and readable visualizations which comes naturally, since most people are familiar with maps. It also makes sense because the visualization of data-points usually requires dimensionality reduction to 2D anyway, which is why high-dimensional data are often visualized as points in 2D space. Embedding algorithms such as the so-called force-directed algorithms¹² are used to create 2D drawings of graphs. They display underlying proximity information by tendentiously putting similar items closer together. (The vertices can be imagined as arranging like magnets with varying degrees of magnetic attraction or repulsion.) To perform a structural analysis one may also employ clustering algorithms such as *k-means*. Thus, it stands to reason to define clusters explicitly in the visualization as „national borders” and coloring the regions. The authors want to „borrow map-related cognitive concepts: [...] items within a country are similar to each other; areas separated by a mountain range are difficult to connect; islands might have atypical qualities, etc.”

Figure 14a shows a simple example of a GMap visualizing relationships between colors. The input to the algorithm is the DOT file shown in Figure 14b.¹³

The Mapping algorithm (given in Algorithm 2) is rather straight forward:

¹² See for example Thomas Fruchterman and Edward Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991 (as cited in [3]).

¹³ This example along with some others can be found at <http://gmap.cs.arizona.edu/>. Users can also paste their own DOT files and generate map representations.



■ **Figure 14** “ColorLand” (a) is produced by the input shown in (b).

Input : graph $G = (V, E)$ with $|V| = n$, $|E| = m$ extracted from a relational data set

Output : map representation of G such that relations in G are displayed as good as possible

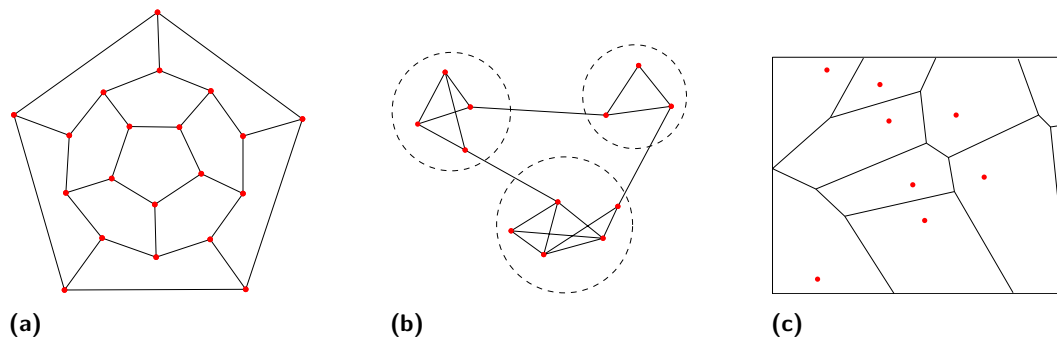
step 1: embed graph in plane (e.g. force directed algorithms...)

step 2: clustering analysis (e.g. k-means...)

step 3: compute Voronoi diagram of the vertices

Algorithm 2 : GMap Algorithm

The set of vertices V corresponds to objects in the data, e.g. books. The set of edges E corresponds to relationships between pairs of objects, e.g. the similarity between two novels. Figure 15 summarizes the algorithm. In the first step, the graph has to be drawn or *embedded*.



■ **Figure 15** The mapping algorithm consists of three basic steps: (a) Step 1: Embedding the graph. (b) Step 2: Clustering analysis. (c) Step 3: Voronoi decomposition of the vertices.

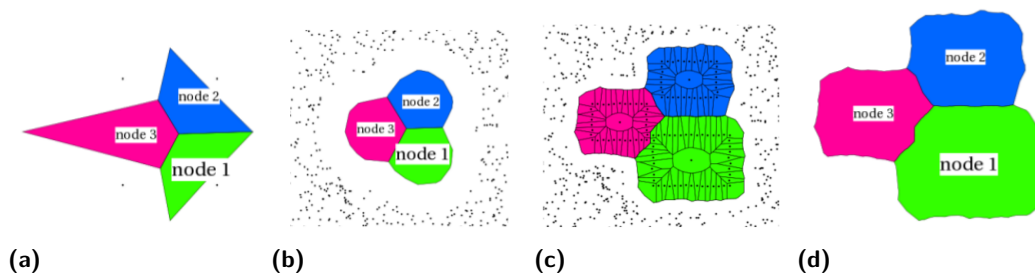
Note that the graph as a data structure is not equal to the depictions commonly associated with it (as shown in Figure 15a). Embedding algorithms usually aim at minimizing the number of crossing edges and making their lengths more or less equal, which is desirable in terms of aesthetics and readability. Possible algorithms are for example the previously mentioned force-directed methods, principal component analysis¹⁴ or multidimensional scaling¹⁵. In the

¹⁴ See for example Ian Jolliffe. Principal component analysis. *Wiley Online Library*, 2002 (as cited in [3]).

¹⁵ See for example Joseph Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964 (as cited in [3]).

next step, a clustering analysis is performed in order to determine groups of vertices that will eventually make up the “countries”. Again, there are some well-established algorithms to do so, such as *k-means*¹⁶. However, an embedding algorithm may not be combined with every other clustering algorithm, since some graph-drawing algorithms do not necessarily put vertices of the same cluster geometrically close together. The last step, in which the Voronoi decomposition of the vertices is computed, yields the actual maps. The Voronoi decomposition of a set of points subdivides the plain into regions (so-called “cells”) such that for each point, the corresponding cell comprises all points that lie closer to that point than to any other point of the set.

Figure 16 shows different approaches to doing so. As can be seen in Figure 16a the node



■ **Figure 16** Approaches to compute Voronoi decomposition. (a) Naive approach. (b) Some improvement. (c) Final method. (d) Result after combining.

labels vary in font size. This is because the underlying graph is vertex-weighted and edge-weighted. Vertex weights reflect some measure of importance and edge weights correspond to some notion of distance between pairs of vertices or rather the real-world objects they represent. Labels are sized according to the vertex weights. Figure 16a shows the result of a naive approach: computing the Voronoi decomposition together with the four points of the bounding box of the labels. (In each of the images, the cells of the auxiliary points are not shown.) As can be seen, this results in very pointy outer boundaries with sharp corners. A more natural appearance can be generated by distributing random points sufficiently distant to the actual vertices and then performing the Voronoi decomposition. As depicted in Figure 16b this leads to more rounded boundaries. Still some issues have to be solved: First, the inner boundaries are completely straight. Second, the uniform size of the cells is inadequate as larger label should have a larger area. This can be solved as illustrated in Figure 16c: Along the bounding box of each label auxiliary points are generated and then randomly perturbed. In contrast to the cells of the outer auxiliary points the cells of these points are drawn as well. Then, cells that belong to the same vertex are merged, leading to the result in Figure 16d. In a final step that is not shown here this is repeated for cells belonging to vertices of the same cluster, generating complex countries. Afterwards, more “geographical features”, such as rivers, lakes, mountain ranges etc., can be added to the map in places where there is considerable space between neighboring labels to further emphasize the distance and solidify the geography metaphor. These are also formed by placing random points in such areas. The algorithm’s time complexity is $O(|V| \log |V|)$. Obviously, the aim

¹⁶ See for example Stuart Lloyd. Least squares quantization in PCM. *IEEE transactions on information theory*, 28(2):129–137, 1982 (as cited in [3]).

here is to create a practical display rather than visualizing a relational data set in a strict graph theoretical way. As a compromise, the actual edges can be superimposed on the map. This may look like the map of “BookLand” shown in Figure 17. The underlying relational data set was obtained with a breadth-first traversal of Amazon’s “Customers Who Bought This Item Also Bought” links, using George Orwell’s “1984” as source node. Each book is connected to the source node by a path of length 9 at max. Merging nodes which represent the same book (with a different publisher or a different binding etc.) reduces the number of vertices to up to 4%. The graph in this case consists of 913 vertices and 3410 edges, with an approximate average degree of 8. “1984” is located in the country “Americana”, which is mainly inhabited by American authors. To the North there is a country called “Russiana”, populated by Russian literature. Here, classic novels by Dostoyevski and Tolstoy. To the West there is “Skakespearea”, which is dominated by Shakespearean tragedies but also contains other literary pieces such as “One Thousand and One Arabian Nights”. Other interesting countries include “Victoriana”, featuring British literature such as the novels by Jane Austen and the Brontë sisters.¹⁷ As highlighted in Figure 17 “1984” appears a second time titled “Nineteen Eighty-Four”. Ideally, vertices that represent the same book (i.e. different editions, publishers, bindings etc.) should be merged. At this point, the algorithm calls for refinement.

2.3.1 Assigning Colors to the Map

A common problem in traditional cartography is how to assign colors to the countries on the map. Obviously, the same color should not be assigned to neighboring countries. A well-known theorem in the field is the Four Color Theorem¹⁸.

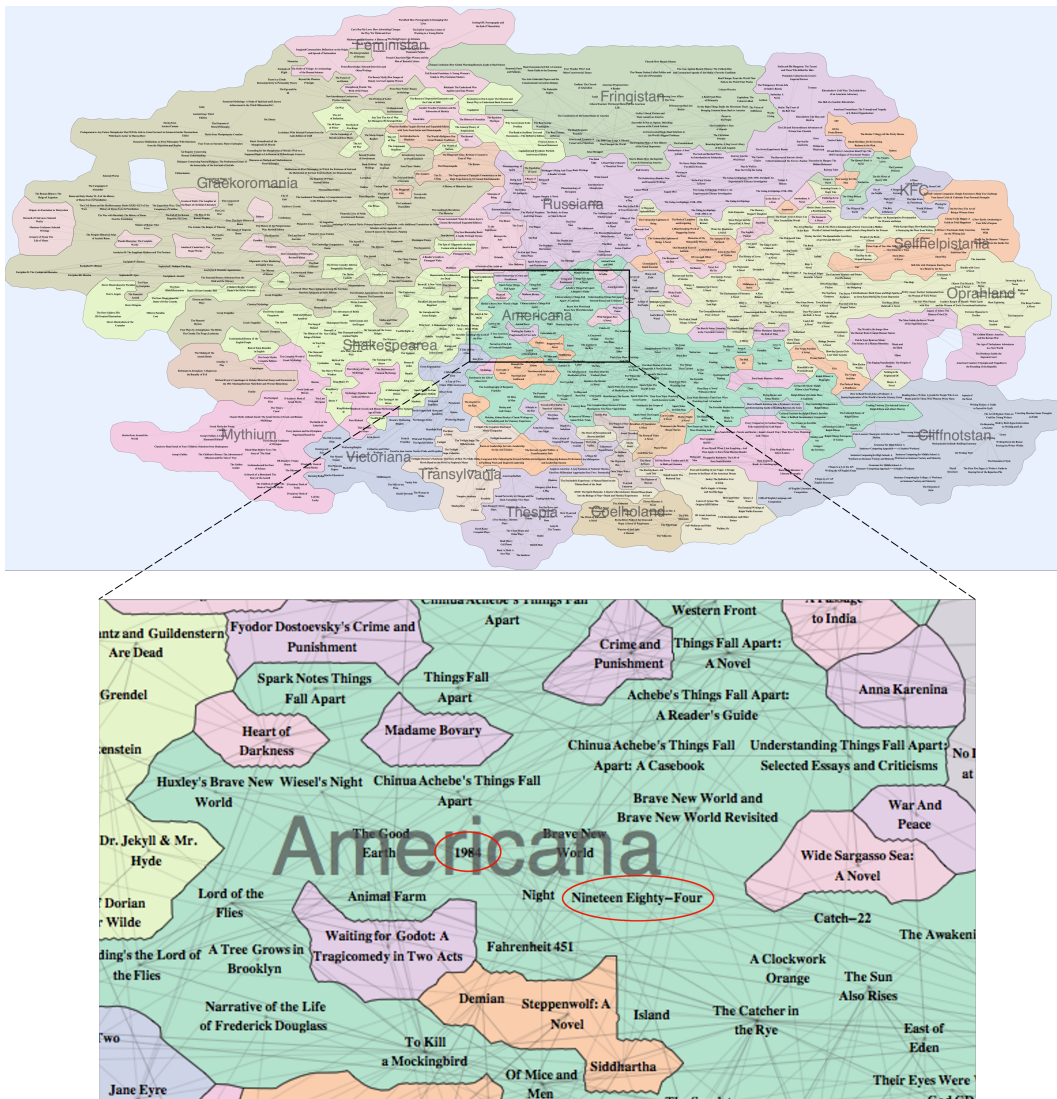
► **Theorem 3** (Four Color Theorem). *Only four colors are needed to color any map so that no neighboring countries share the same color, given that each country forms a contiguous region.*

In the case of GMaps, the constraint stated here is not fulfilled since exclaves exist. These exclaves should be given the same color as their respective parent country. Thus, many more colors may be required. In order to assign colors to the GMaps, the inventors used five distinct base colors which are easy to differentiate and blending between them. Thus a linear and discrete color space C is obtained. Consecutive colors in this array are similar due to the blending. Thus, applying such pairs on neighboring countries on the map should be avoided to maximize readability. In order to solve this, one can define the *country graph* $G_c = \{V_c, E_c\}$. It does not equal the underlying graph that is being visualized by the map. Instead, its vertices represent countries that are connected by an edge if they share a non-trivial boundary. Furthermore, a function $c : V_c \rightarrow C$ that assigns a color to a vertex, as well as a color distance function $d : C \times C \rightarrow \mathbb{R}$ are required. Let $w_{ij} \geq 0$ be the weights¹⁹ associated with edges $\{i, j\} \in E_c$.

¹⁷ An interactive version of a similar map can be explored at <http://yifanhu.net/BookMap/index.html>, where the reader can query about their favorite books and easily find similar ones as the map features links to Amazon.

¹⁸ See for example Neil Robertson, Daniel Sanders, Paul Seymour, and Robin Thomas. The Four-Colour Theorem. *Journal of Combinatorial Theory, Series B*, 70(1):2–44, 1997.

¹⁹ Weights correspond to the inverse of the distance between two countries. Thus, the closer two countries are, the more dissimilar the assigned colors will be, which makes the countries easy to distinguish.



■ Figure 17 “BookLand”

Then, the coloring problem can be formulated as an optimization problem: Define the color function c such that the vector of color distances along the edges

$$v(c) = \{w_{ij}d(c(i), c(j)) | \{i, j\} \in E_c\} \tag{1}$$

is maximized with respect to a cost function. One such cost function that can be considered here is 2-norm, i.e. the weighted sum of squared distances between colors of neighboring countries:

$$\max_{c \in C} \left\{ \sum_{\{i,j\} \in E_c} w_{ij}d(c(i), c(j))^2 \right\} \tag{2}$$

The coloring problem can be modeled as a vertex labeling problem if $C = \{1, 2, \dots, |V_c|\}$ and the color function c is a permutation maximizing the labeling differences. Plugging this in

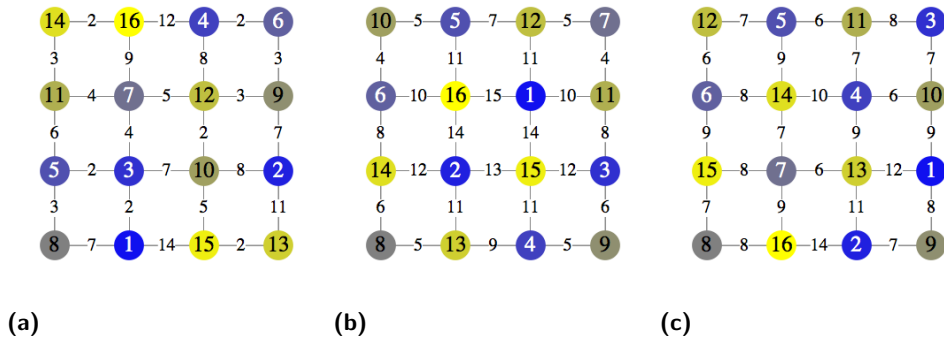
Equation 2 yields

$$\max\left\{\sum_{\{i,j\}\in E_c} w_{ij}(c_i, c_j)^2\right\} \quad (3)$$

with c being a permutation vector of $\{1, 2, \dots, |V_c|\}$ and c_i its i -th element. This can be approximated by continuous constrained optimization problem:

$$\max\left\{\sum_{\{i,j\}\in E_c} w_{ij}(c_i, c_j)^2\right\}, \text{ s.t. } \sum_{k\in V_c} c_k = 1 \quad (4)$$

where $c \in \mathbb{R}^{|V_c|}$, that is the permutation vector is now a vector of decimal numbers with the entries adding up to one. This problem is solved when c is the eigenvector corresponding to the largest eigenvalue of the weighted Laplacian (the difference between the degree matrix and the adjacency matrix) of the country graph. After solving Equation 4 the ordering of that eigenvector is an approximate solution for Equation 3. The authors call this algorithm *SPECTRAL*²⁰. The coloring results can be refined by iteratively swapping pairs of vertices such that the coloring scheme is improved according to the cost function in a greedy manner. However, this has a high computational complexity since all $O(|V_c|^2)$ pairs have to be considered. Figure 18 shows different coloring schemes. The RANDOM coloring produces some undesirable pairings such as in vertices 1 and 3. One possible application for the GMap



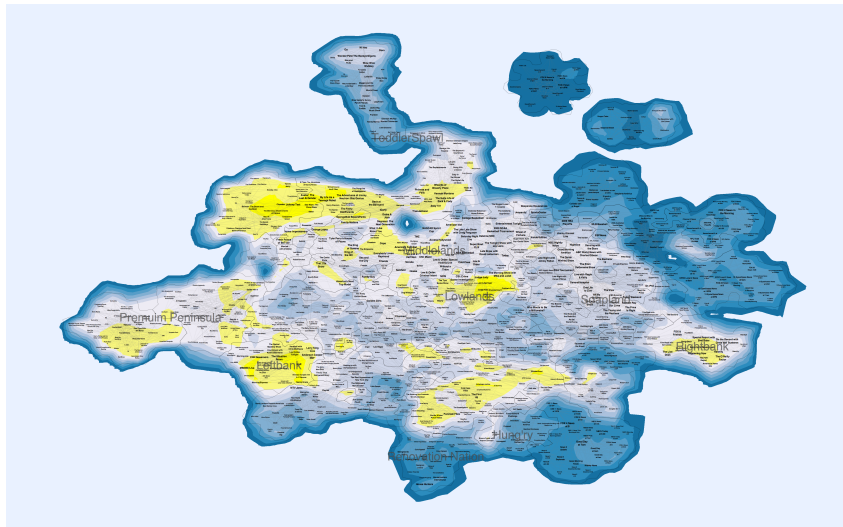
■ **Figure 18** Different coloring schemes. (a) RANDOM. (b) SPECTRAL. (c) SPECTRAL + GREEDY.

representation is to create a recommendation system for movies or books as the authors propose in [4]. In many online streaming services it seems not very obvious to the user why an item is recommended to them on the basis of their previous viewing behavior. By contrast, if you could study a map of “MovieLand”, clusters and neighborhoods in the Movie database would be revealed. Chris Volinsky, one of the authors of [4] participated in a 2007 contest held by the widely popular platform *Netflix* [12]²¹. They offered one million dollars to the research team that could make their recommendation system *CineMatch* 10% more accurate. It may not sound like much but it turns out that this is a very twisted challenge due to manifold idiosyncrasies in individual viewing behavior. For instance, very polarizing films, which are either loved or hated by most people, present a problem. Additionally, it is

²⁰ Spectral graph theory investigates the properties of a graph in terms of eigenvalues and eigenvectors of matrices associated with it, such as the adjacency matrix etc.

²¹ http://www.nytimes.com/2008/11/23/magazine/23Netflix-t.html?pagewanted=all&_r=

difficult to predict a user’s preference for sequels. It is widely agreed upon that the first part of the *Matrix* trilogy was a stroke of creative genius while the two sequels fail to live up to its example. Volinsky actually ended up in the winning team two years later, when his team – in coalition with three other teams – managed to improve the accuracy by 10.05% [8]²². Anyhow, Figure 19 shows how a GMap can be used to beautifully display a landscape of TV shows. Color coding can be personalized: warm colors represent areas of high interest while less relevant shows appear in cooler colors. The alert observer may have noticed that very different areas are highlighted. The reason is that data are based on households rather than individuals. For instance, children living in a household naturally will prefer cartoons and the like while adults may follow the daily news or talk shows. A representation like this makes it very intuitive and enjoyable to identify shows that suit one’s taste.



■ **Figure 19** Personalized recommendation heat map.

2.4 Conclusion & Discussion

“To read or not to read, that is the question.” asks the previously mentioned newspaper article [11] provocatively, referring to Moretti’s work in the Stanford Literary Lab where he performs network analyses of Shakespearean tragedies. Even hidebound traditionalists have to admit that a holistic study of literature is simply impossible without automation. Visualization techniques for distant reading are vital to cope with the horrendous amounts of textual data. However, it is not trivial to communicate large quantities of data in an efficient and readable way.

In this paper, two applications, which use proportional symbol maps and other visualization techniques, were reviewed. Proportional symbol maps are an effective way to depict spacial statistical data. However, some problems remain to be solved, such as developing a good heuristic for physically realizable drawings that solves Max-Min or Max-Total. In general, if the amount of data to be displayed becomes too large one has to employ workarounds such

²² <http://bits.blogs.nytimes.com/2009/06/26/and-the-winner-of-the-1-million-netflix-prize-probably-is/>

as superimposing a small section of special interest on the map.

Furthermore, it was examined how pseudo-geographic maps can be used as a powerful method to disclose structural relationships within datasets that can be stored as graphs. When using GMaps – or any other 2D representation of high-dimensional data – one has to accept some degree of distortion of the underlying information. Nonetheless they display clustering information intuitively and beautifully and allow interactive navigation through large data sets. However, the algorithm may be refined such that multiple vertices representing the same real-world entity are eliminated.

Irrefutably, the presented visualization techniques are a valuable addition to traditional humanities and allow to an overview of large amounts of data much more quickly than formerly possible.

References

- 1 Elton Barker, Stefan Bouzarovski, Christopher Pelling, and Leif Isaksen. Mapping an Ancient Historian in a Digital Age: The Herodotus Encoded Space-Text-Image Archive (HESTIA). 2010.
- 2 Sergio Cabello, Herman Haverkort, Marc van Kreveld, and Bettina Speckmann. Algorithmic Aspects of Proportional Symbol Maps. *Algorithmica*, 58(3):543–565, 2010.
- 3 Emden Gansner, Yifan Hu, and Stephen Kobourov. GMap: Visualizing Graphs and Clusters as Maps. In *2010 IEEE Pacific Visualization Symposium (PacificVis)*, pages 201–208. IEEE, 2010.
- 4 Emden Gansner, Yifan Hu, Stephen Kobourov, and Chris Volinsky. Putting Recommendations on the Map: Visualizing Clusters and Relations. In *Proceedings of the third ACM conference on Recommender systems*, pages 345–348. ACM, 2009.
- 5 Uta Hinrichs, Beatrice Alex, Jim Clifford, Andrew Watson, Aaron Quigley, Ewan Klein, and Colin M. Coates. Trading Consequences: A Case Study of Combining Text Mining and Visualization to Facilitate Document Exploration. *Digital Scholarship in the Humanities*, 30(suppl 1):i50–i75, 2015.
- 6 Stefan Jänicke, Greta Franzini, Muhammad Faisal Cheema, and Gerek Scheuermann. On Close and Distant Reading in Digital Humanities: A Survey and Future Challenges. In *Eurographics Conference on Visualization (EuroVis) - STARs*. The Eurographics Association, 2015.
- 7 Ewan Klein, Beatrice Alex, Claire Grover, Richard Tobin, Colin Coates, Jim Clifford, Aaron Quigley, Uta Hinrichs, James Reid, Nicola Osborne, et al. Digging into Data White Paper: Trading Consequences. 2014.
- 8 Steve Lohr. And the Winner of the \$1 Million Netflix Prize (Probably) Is ... *The New York Times*, 2009-06-26.
- 9 Franco Moretti. *Graphs, Maps, Trees : Abstract Models for a Literary History*. Verso, London; New York, 2005.
- 10 Randall Munroe. Map of Online Communities. <https://xkcd.com/802/>, 2010.
- 11 Mark Reichwein. Wie gelesener Blumenkohl. *Welt am Sonntag*, 2016-07-03.
- 12 Clive Thompson. If You Liked This, You're Sure to Love That. *The New York Times Magazine*, 2008-11-21.

List of Figures

1	xkcd's Map of Online Communities [https://xkcd.com/802/]	6
2	Screenshot of a sample inquiry in the Trading Consequences web interface [http://tcqdev.edina.ac.uk/vis/tradConVis/]	7
3	Screenshot of a sample inquiry in the Trading Consequences web interface [http://tcqdev.edina.ac.uk/vis/tradConVis/]	7
4	The Atlas of Ancient and Classical Geography by Samuel Butler [https://commons.wikimedia.org/w/index.php?curid=11500204]	8
5	Screenshots of HESTIA in Google Earth [http://hestia-geo.open.ac.uk:8080/geoserver/wms/kml?layers=hestia:google_earth]	9
6	Screenshot of the HestiaVis Platform [http://www2.open.ac.uk/openlearn/hestia/index.html#index]	9
7	Screenshots of the HestiaVis Platform [http://www2.open.ac.uk/openlearn/hestia/index.html#index]	9
8	PSM generated with HESTIA data [1]	10
9	Cluttering labels in the Trading Consequences web interface [7]	10
10	Illustration of concepts related to PSM	11
11	Illustration of concepts related to PSM, c.f. [2]	12
12	Illustration of concepts related to PSM, c.f. [2]	12
13	Results of different stacking algorithms [2]	14
14	Example of the GMap visualization tool [http://gmap.cs.arizona.edu/]	15
15	Illustration of the GMap algorithm	15
16	Voronoi decomposition in the GMap algorithm [3]	16
17	"BookLand" [http://www.cs.arizona.edu/~kobourov/PROJECTS/maps.html]	18
18	Map coloring schemes [3]	19
19	Personalized recommendation heat map [4]	20

3 Estimating Similarity of Notions using Google

Longfei Mao

Abstract

Words and phrases are the basic tools in human communication. Each word and phrase has special meaning for the human being. But for machines, especially for the computer, they are just constant permutations of letters without meaning, needless to say the relationship between them. However, meaning of words could be of great use, such as a semantic search. In this paper, two algorithms to calculate the similarity of given words are introduced. The first algorithm, named as Normalized Google Distance (NGD for short), is all based on the information distance and Kolmogorov complexity. The occurrence rate of each word is a very important parameter in this algorithm, and the world-wide-web and Google are used to estimate it. Theoretically the methods should be also functional when other search engines and databases are used. Possible applications are given, such as hierarchical clustering, classification and language translation. As millions of independent users enter textual information every day, the stability of the NGD algorithm is not very good. In the end, the Google Core Distance (GCD for short) is given as an improvement. As the information distance is calculated in a more directly way, the result of GCD algorithm is more stable and faster than the NGD algorithm.

3.1 Introduction

The world of human beings is constructed of objects, abstract objects and specific objects. We are living in this kind of world. It is easy to for us to get used to them and use them. But for the machines, especially the computers, these objects are unfamiliar. The world of computers is constructed of numbers. The computers cannot handle the real objects directly, or even understand them directly. As a result, handling the objects' information directly is a very difficult job for the computer. Except some kinds of specialized sensors, most of these jobs need to be done manually. However, the amount of objects increases explosively, and there is too much information from them. It becomes more and more impossible to handle all of them manually, and methods to handle them automatically are required[23, 25]. In this paper several methods will be given, making it possible for computers to handle the relationships of objects in human's world sketchily.

3.1.1 Basic Assumptions

For artificial intelligence questions, we usually start from human behavior, and this algorithm is not an exception. Humans can handle the objects in their society. For the computers, their "society" is the database. But unfortunately at present we cannot construct a kind of data structure, which is able to represent the whole object, neither abstract nor concrete. The meaning space of an object is so wide, that even for humans it is not possible to describe all its characters. As a result, objects are usually classified, remaining the similarity, losing small differences. A very widely used tool to do this job is the language. To make use of this tool, a basic assumption should be satisfied: All the objects have their own names, and all the characters of the objects can be represented by their names[39, 14]. There are objects which cannot be named literally, such as "home". This kind of situation will not be considered in this work[2, 31, 8].

Millions of web users have created billions of web pages with trillions of characters making the world-wide-web a perfect literal database which simulates the human's literary words. Although on average the quality of contents from world-wide-web is not high, and there are possibly self-made "new" words, the sheer mass of information about almost every conceivable topic makes it likely that extremes cancel each other and the majority or average is meaningful in a low-quality approximate sense. Using this, a general method to tap the amorphous low-grade knowledge could be given. What is more, this database is available for all the users, whose search engine has the aggregate page-count function, like Google, which means that the scale of the databa is probably big enough to represent all the human's literary words [13].

3.2 Similarity Distance Implementation

3.2.1 Similarity of Numbers

In general, the input of the algorithms could be any word from human's dictionary. But for the beginning, let us start from a simple simple situations, in which all the inputs are numbers.

As said before, for most artificial intelligence problems, we usually start from the human behavior, which is of great reference value. If someone is required to compare some numbers (such as 10, 99, 100) to find out which two of them are more similar, 99 and 100 must be the answer[37]. Because the only character of the number is the value of it, and the difference value of 99 and 100 is much smaller than any other combination in this case. Sometimes the difference values could be equal (such as 49 and 50 vs. 99 and 100), and for this situation, 99 and 100 will be chosen, because the difference value is 1% of the bigger element (100), but for the other pair, the difference value is 2% of the bigger element (50).

this way to compare numbers can be formulized and expressed by the following formula:

$$N(x, y) = \frac{D(x, y)}{\max(x, y)}$$

$$D(x, y) = |x - y|$$

$N(x, y)$ stands for the similarity of two numbers (positive, all the numbers considered in this case should be positive), the value of it is between 0 (equal) and 1 (totally different), the smaller the value of $N(x, y)$ is, the less difference there will be, and the more similar these two numbers are. $D(x, y)$ is the difference value of the two numbers, which is one of the most important parameters in this formula. With this formula, as long as two positive numbers are given, a definite parameter could be calculated. This parameter is so effective, that only with this parameter the similarity of these two numbers could be known.

3.2.2 Modeling

In previous section, a basic formula for determining similarity of numbers was explained , which can directly calculate the similarity of two numbers. Along this way, it is easy to come up with an idea, that if we could correspond each object to a number that represents it in some sence, then adapting the formula for numbers, the similarity of numbers could also be computed. According to the basic assumptions, all of the objects are thought to have their own literal names, and this literal names can express all the information about the objects. So in the following research, the literal names of the objects will be talked about, which are

terms in the database (world-wide-web). And our objective can be explained more detailed: find out a way to make determined relationships between terms and some certain numbers, and find out a formula to compute the similarities of the numbers in order to find the most similar pairs. Or it can be describe the process more vividly: All the web pages are points in a high dimensional space, and the searched terms are sets of the points. The objective is to find out a way to classify the web pages into sets from the space, and then compute the distance between certain sets. In this case, a short distance equals to a high similarity of searched objects[16, 21].

At this point, we encounter a problem. At present there is no efficient way to transform terms into numbers without any information loss. For example, if we use ASCII code, the term "red" will become "0111 0010 0110 0101 0110 0100". However, in this case most of the information is lost, only the English spelling of the term is reserved. We cannot know the RGB value of this color is (255, 0, 0). For the term "rot"("red" in German) the value will be "0110 1100 1101 0011 0010 1000 1100", and it is hard to say the similarity of these two binary numbers is high. What is in need is a way to transform terms into numbers with as little information lost as possible.

At present there is no algorithm to make a lossless transform, we need to find out a new way to substitute this process as much as possible. The basic idea introduced by Rudi L. Cilibrasi and Paul M.B. Vitányi is to use Google semantics. The Google semantic of a term is a set of all the web pages returned by a Google query. According to the assumptions, the Google semantic is able to explain all the meanings the given term, as long as the database is big enough[14]. Thanks to the great scale of the web pages in world-wide-web, from which almost all the usages of the terms and the frequencies of the usages can be extracted, all the webpages, where the given term is contained, can be thought as an information pool about the terms in question. However, as the Google semantic is just a set of web pages, and at present there is no further processing on the webpages, it may be not very precise. Technically it is possible that some terms of different meanings have the same Google semantic. And when the given terms are very frequently used (such as "and", "not", "or") or their meanings are absolutely opposite (such as "right", "wrong"), the Google semantic cannot distinguish them. All in all, as the web grows, the Google semantic may become less primitive[36].

Although primitive, the Google semantic is a feasible solution to get relationship between search terms. An algorithm to compute the distance between the sets of Google semantics is required. The Kolmogorov complexity can be used for that purpose. The Kolmogorov complexity $K(x)$ of a string x is the length, in bits, of the shortest computer program of the fixed reference computing system that produces x as output. The choice of computing system changes the value of $K(x)$ by at most an additive fixed constant[15]. Since $K(x)$ goes to infinity with the length of x , this additive fixed constant is an ignorable quantity if we consider large x . One way to think about the Kolmogorov complexity $K(x)$ is to view it as the length, in bits, of the ultimate compressed version from which x can be recovered by a general decompression program[14]. It is the lower bound on the ultimate compression: for all the compressors $C(x)$ (no matter already discovered or not), $K(x)$ is always less than or equal to the length of the compressed string of x . In other words, $K(x)$ is the ultimate value of the length of compressed version of x , and the task in designing better and better compressors is to approach this lower bound as closely as possible.

3.2.3 Normalized Information Distance

So far we have found the way to convert the terms into sets of points in the high dimensional data space by applying Google semantics, and the formula to calculate distance between

numbers. If we use these concepts in the basic formula, the expression of the information distance will be done[32, 22].

$$E(x, y) = K(x, y) - \min(K(x), K(y))$$

In the formula above, the function $K(x, y)$ stands for the Kolmogorov complexity of term x and y . The information distance $E(x, y)$ is actually a metric. For all possible compression functions $F(x, y)$, $E(x, y) \leq F(x, y) + c_f$, where c_f is a constant that depends only on the certain compression function itself F but not on the inputs x, y of it. The information distance E should be a universal metric for all computable distances, because E minimizes every member of the distances up to an additive constant. If the distance of x, y is close according to any kind of computable distance, they must be also close to each other according to the information distance E . Just like our simple situation with numbers, the information distance itself could not express the true similarity, it should be compared to the size of the information itself. That is why the normalized information distance is introduced.

$$NID(x, y) = \frac{K(x, y) - \min(K(x), K(y))}{\max(K(x), K(y))}$$

The Normalized Information Distance (NID) inherits the universality of the information distance and has the values from 0 to 1 (0 for exactly the same and 1 for completely different).

3.2.4 Normalized Compression Distance

Apparently the Kolmogorov complexity is talking about the limitation of compression, it is just a theoretical value, and uncomputable. As a result, the NID is also uncomputable. However, Kolmogorov complexity can be approximated by an arbitrary compressor C , since as already mentioned, it is an upper bound for the Kolmogorov complexity. Thus, for any compressor C , using $C(x)$ to denote the length of the compressed version of a string x , Normalized Compression Distance (NCD) can be computed:

$$NCD(x, y) = \frac{C(x, y) - \min(C(x), C(y))}{\max(C(x), C(y))}$$

It is noteworthy that the algorithm used in NCD is actually a family of compression functions, for all possible compressors C . The value changes when the algorithm changes, and the The NID is the limiting case of it, where an algorithm can give Kolmogorov complexity as output. As mentioned, $C(x)$ is always one of the upper bounds of $K(x)$, with the development of compression technology, it is more and more accurate for C to approximates K . As to how the NCD approximates the NID if C approximates K , please see [30]. The NCD is an important medium, using this method, every known compression could be used to approximate the value of NID. Thus, NID is no longer unreachable, man can touch it step by step (although infinite steps are needed).

3.2.5 Normalized Google Distance

Noticing that what the Kolmogorov complexity focuses on is the length of the compressed strings not the compressing algorithm, it is easy to come up with the concept of entropy[34]. In information theory, the entropy (more specifically, the Shannon entropy) is the expected value of the information contained in each message. Shannon defined the entropy I of a discrete random variable X with probability mass function $P(X)$ as:

$$I(X) = -\log_b P(X) = \log_b \frac{1}{P(X)}$$

Common values of the base b are 2, Euler's number e , and 10, and the unit of entropy is Shannon for $b = 2$, Nat for $b = e$, and Hartley for $b = 10$. When $b = 2$, the units of entropy are also commonly referred to as bits[4]. As Kolmogorov complexity just requires the binary length of the string, man can use the above formula to compute the entropy of a certain event with base $b = 2$, and in this case, the events are always that the given term (or two terms for doubleton search) appears. The real possibility of the terms' appearance is unknown forever. However, statistics is able to give an estimated value when the sample space is big enough. The number of web pages currently indexed by Google is approaching 10^{10} . Every common search term occurs in millions of web pages. This number is so vast, and the number of web authors generating web pages is so enormous (and can be assumed to be a truly representative very large sample from humankind), that the probabilities of Google search terms, conceived as the frequencies of page counts returned by Google divided by the number of pages indexed by Google, approximate the actual relative frequencies of those search terms as actually used in society[19, 35]. Define $g(x)$ the probability of term x to appear in a webpage, and $g(x, y)$ the probability of both terms x, y to appear in the same webpage :

$$g(x) = g(x, x), g(x, y) = \frac{|x \cap y|}{N}$$

where N stands for the number of each set and each doubleton set (by definition unordered). Please note that it is quite possible that N is bigger than the total amount M of all the webpages in the database. Because for every legal pair x, y with $x \neq y$, the web page $z \in x \cap y$ are counted 3 times: once in $x = x \cap x$, once in $y = y \cap y$, and once in $x \cap y$.

Now we have the entropy $G(x)$ of terms x , which is defined by

$$G(x) = G(x, x), G(x, y) = \log \frac{1}{g(x, y)}.$$

In this sense, Rudi L. Cilibrasi and Paul M.B. Vitányi suggested to use the Normalized Google Distance (NGD), which is able to capture the assumed true semantic relations governing the search terms approximately[14]:

$$\begin{aligned} NGD(x, y) &= \frac{G(x, y) - \min(G(x), G(y))}{\max(G(x), G(y))} = \frac{\log N - \log f(x, y) - \min((\log N - \log f(x)), (\log N - \log f(y)))}{\max((\log N - \log f(x)), (\log N - \log f(y)))} \\ &= \frac{\log N - \log f(x, y) - (\log N - \max((\log f(x)), (\log f(y))))}{\log N - \min(\log f(x), \log f(y))} = \frac{\max(\log f(x), \log f(y)) - \log f(x, y)}{\log N - \min(\log f(x), \log f(y))} \end{aligned}$$

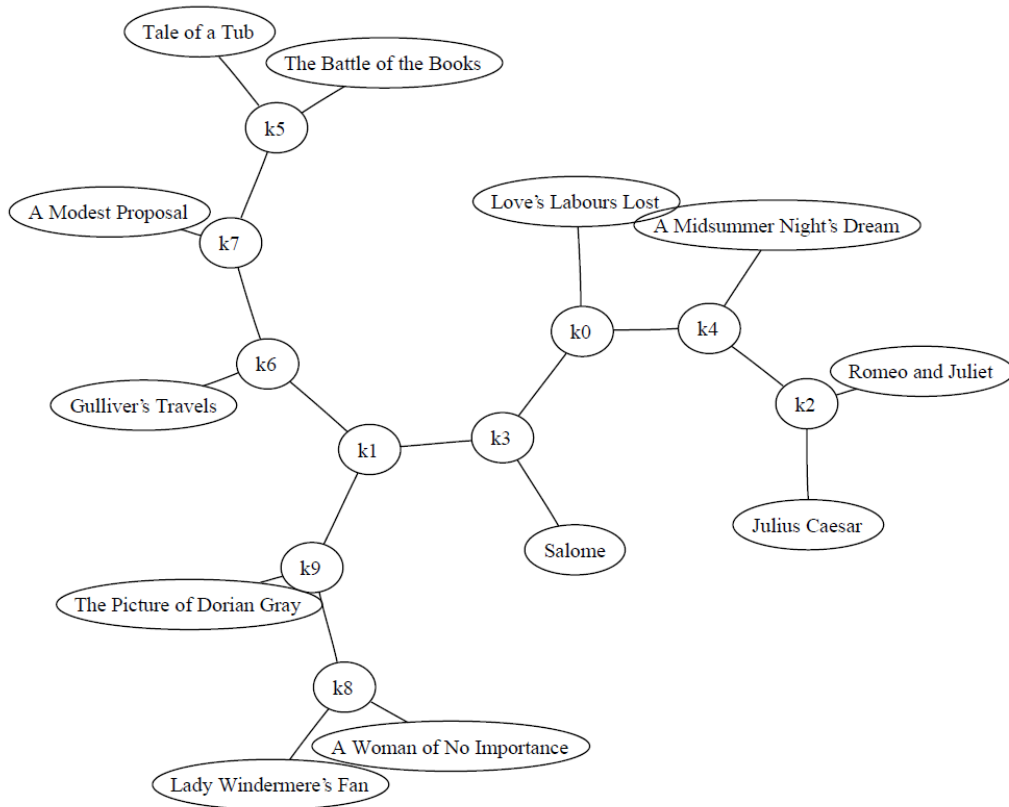
where $f(x)$ is the number of reported pages, which contain string x , and $f(x, y)$ denotes the number of pages containing both x and y , as reported by Google[29].

The NGD is an approximation to the NID by using the prefix code-word lengths (entropy) generated by the Google distribution as defining a compressor approximating the length of the Kolmogorov code, using the background knowledge on the web as viewed by Google as conditional information[9, 11, 5].

3.3 Performance Test

Now that the way to calculate NGD is explained, it is time to discuss how it performs. A possible method is to use NGD to cluster objects by constructing a hierarchical cluster tree, which is also named as the semantic tree[38]. There are two main steps to do this. Firstly compute the NGD-matrix with all the input terms, and then produce the best-matching unrooted ternary tree using a novel quartet-method style heuristic based on randomized hill-climbing using a new fitness objective function for the candidate trees. The first step of hierarchical clustering is to compute the distances between all the input clusters. Then

just keep on finding out the closest two terms, and classify them into a new cluster and computing the distance between the new cluster and all the old clusters until only one cluster is left. There are several kinds of hierarchical clusterings, although slightly different from each other in details, the main steps of them are almost the same, for more explanation please see [1, 28, 30].



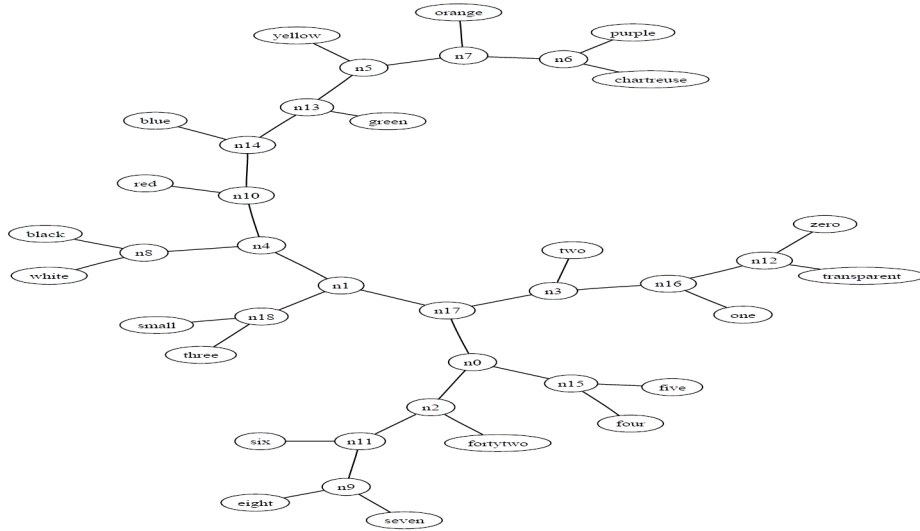
■ **Figure 1** Hierarchical clustering of authors

■ **Table 1** Distance matrix of pairwise NGD 's

A Woman of No Importance	0.000	0.458	0.479	0.444	0.49	0.149	0.362	0.471	0.371	0.300	0.278	0.261
A Midsummer Night's Dream	0.458	-0.011	0.563	0.382	0.301	0.506	0.340	0.244	0.499	0.537	0.535	0.425
A Modest Proposal	0.479	0.573	0.002	0.323	0.506	0.575	0.607	0.502	0.605	0.335	0.360	0.463
Gulliver's Travels	0.445	0.392	0.323	0.000	0.368	0.509	0.485	0.339	0.535	0.285	0.330	0.228
Julius Caesar	0.494	0.299	0.507	0.368	0.000	0.611	0.313	0.211	0.373	0.491	0.535	0.447
Lady Windermere's Fan	0.149	0.506	0.575	0.565	0.612	0.000	0.524	0.604	0.571	0.347	0.347	0.461
Love's Labours Lost	0.363	0.332	0.607	0.486	0.313	0.525	0.000	0.351	0.549	0.514	0.462	0.513
Romeo and Juliet	0.471	0.248	0.502	0.339	0.210	0.604	0.351	0.000	0.389	0.527	0.544	0.380
Salome	0.371	0.499	0.605	0.540	0.373	0.568	0.553	0.389	0.000	0.520	0.538	0.407
Tale of a Tub	0.300	0.537	0.335	0.384	0.492	0.347	0.514	0.527	0.524	0.000	0.160	0.421
The Battle of the Books	0.278	0.535	0.359	0.330	0.533	0.347	0.462	0.544	0.541	0.160	0.000	0.373
The Picture of Dorian Gray	0.361	0.415	0.463	0.229	0.447	0.324	0.513	0.380	0.402	0.420	0.373	0.000

Figure 1 shows the clustering result of some famous books with their NGD matrix in table 1. It is not difficult to find out that the terms will be near to each other in the tree when the NGD between them is small, and vice versa. To get a more obvious result to show the performance, we have figure 2, in which the terms are all numbers, colors or tricky words.

It is clear that the colors are distributed concentratedly to one certain part of the tree, and the numbers to the other part. On the farthest reach of the color side and the number side, there are words with only a few meanings, such as chartreuse and fortytwo. Terms with more ambiguous interpretation are put in the center, such as "black", "white", "zero", "two". As to the terms which are not exactly colors or numbers are also in the center, like the word "small".



■ **Figure 2** Colors and numbers arranged into a tree using NGD

To some extent, this result is satisfying. Almost all the terms are distributed to its “family”, as to some details in the tree, such as the brother of the term “zero” is not other numbers but “transparent”, it is still acceptable. Some of the terms are quite frequently used, for example "black", "white", “zero”, “one”, "two", as a result, for a large part of the returned web pages of them will be the same. However, the NGD is still able to identify which class these terms belong to. This is a good proof that NGD is able to compute the similarity of similar and frequently used terms.

3.4 Applications

As a relatively new algorithm in term comparisons, because of the lack of follow-up applications, there is few situations suitable for using NGD directly. However, there are potential applications. Yet one of the best potential applications of the NGD algorithm is the language translation, or dictionary function, for example, an English-Spanish dictionary[10, 17].

This process is not complicated. The system is considered to try to infer a translation-vocabulary between English and Spanish. At the outset, eight English words with their matched Spanish translation are given as the training data for the system, 2 . Then a NGD matrix will be formed, the columns of which are labeled as the (in this situation eight) translation-known English words, the rows as the (in this situation five) translation-unknown English words, and every entry of the matrix is the NGD value between its column and row. Next, form another NGD matrix in the same way but with the Spanish words. In addition the columns should be in the same order as the English words, as the translation of these

words is already known as the training data, this is very easy to accomplish. After this work, keep on rearranging the rows from the Spanish matrix in order to get the permutation with the highest positive pairwise correlation with the values in the given English word basis matrix. If there is no positive correlation, report a failure to extend the vocabulary. The result of the test is satisfying, the computer inferred the correct permutation for the testing words, see table 3 and table 4.

From all the above, it is clear to see that NGD is an adequate algorithm to calculate the similarity of given words/phrases, which are terms on the world-wide-web. The value of NGD is usually between 0 (means identical) and 1 (means unrelated).

■ **Table 2** Given starting vocabulary

English	Spanish
tooth	diente
joy	alegria
tree	arbol
electricity	electricidad
table	tabla
money	dinero
sound	sonido
music	musica

■ **Table 3** Unknown-permutation vocabulary

English	Spanish
plant	bailar
car	hablar
dance	amigo
speak	coche
friend	planta

■ **Table 4** Predicted (optimal) permutation

Salome	Spanish
plant	planta
car	coche
dance	bailar
speak	hablar
friend	amigo

3.5 Improvements

3.5.1 Drawbacks and Improvements of NGD

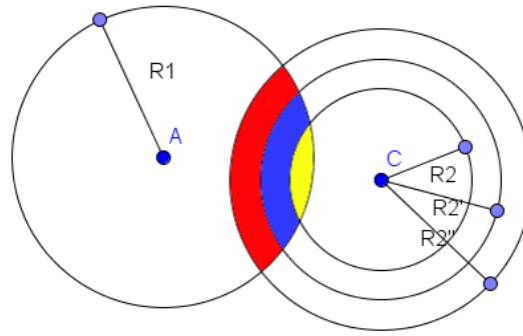
However, this algorithm has some drawbacks. As we use the term's entropy to approximate its Kolmogorov complexity, and use the term's appearance rate in database to approximate its real probability, the quality of the database becomes extremely important for the algorithm.

In other words, if the database is not big enough or representative enough, the algorithm will not give satisfactory results. What is more, if we use the biggest database in the world, the world-wide-web, just like in the paper, as this database is updated in a very high speed, it is very possible that the NGD value of a term will change overtime. Especially when one or more of the terms suddenly becomes popular. Because at that time the common part of two terms will also increase, and there will be two big variables in the formula[24].

This is illustrated in Figure 3. When term y for is getting popular, there will be more and more webpages with term y , and the circle C , which stands for term y , will also expand. As a result, the corresponding set of pages will become bigger and bigger (illustrated as the increase of its radius $R2$ and the increase of the value of $f(y)$), and the common part of the two circles will also increase. This common part is $f(x, y)$ in NGD's expression:

$$NGD(x, y) = \frac{\max(\log f(x), \log f(y)) - \log f(x, y)}{\log N - \min(\log f(x), \log f(y))}$$

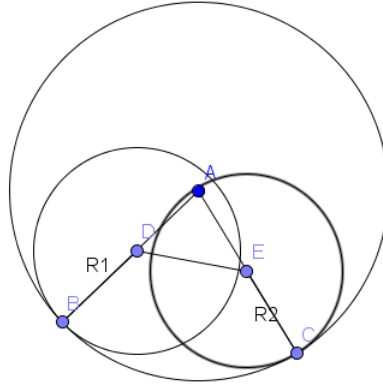
After this growth if $f(x) > f(y)$, then $NGD(x, y) = \frac{\log f(x) - \log f(x, y)}{\log N - \log f(y)}$ and if $f(x) < f(y)$, $NGD(x, y) = \frac{\log f(y) - \log f(x, y)}{\log N - \log f(x)}$. It is not difficult to find that in both situations, the numerator and the denominator change disproportionately. As a result, the value of NGD should not be considered to remain constant[3].



■ **Figure 3** variations of intersection

The original NGD algorithm supposed that both the total number of pages and the total number of search terms in Google would continue to grow for some time, so that the similarity distance between two keywords would not be significantly changed. However, that is just the common situation. There are so many possibilities that one term suddenly becomes focused on[12]. Such as a breakthrough in some area, or a concept mentioned in a popular TV show, or even the advent of new products[26]. Since how similar two objects are should not depend on the popularities of them, but the real meanings, it is preferable that the relationship between two terms stays as stable as possible if no new meanings are added into the terms. As this is not guaranteed, the time-variance effect is a big drawback. Measure called Google Core Distance (GCD for short) was introduced to overcome these drawbacks of NGD[27].

As can be seen in figure 4, the information distance of term D and term E is the length of the segment DE . And an important assumption here is that the angle at center of the circle ($\angle BAC$ in this case) should not be bigger than 90° . Noticing that generally the sizes of circles D and E is very small in compare with the outside circle A . And if $\triangle ADE$ is a obtuse triangle, the public part of circle D and E will be very small, which means that the



■ **Figure 4** Google Core Distance

NGD will probably be 0. The outside circle A should be the tangent circle of all the inside circles (both D and E), because every circle here stands for the total scope of a certain term, and the biggest circle (circle A in this case) should be the total scope of all the terms in the database, which means that the margin of circle A is consisted of the farthest points form all small circles (remember the whole database is considered as a high-dimensional space, not just a 2D circle represented by the figure). So obtuse triangle means no similarity, and as long as the similarity is positive, the triangle is not obtuse. If the right triangles are considered as the extreme case, the Pythagorean Theorem can be used to compute the upper limit of GCD, which is $\sqrt{(R - r_1)^2 + (R - r_2)^2}$. And now the range of GCD is got:

$$GCD(x, y) \in [0, \sqrt{(R - r_1)^2 + (R - r_2)^2}]$$

3.5.2 GCD vs. NGD

To compare these two algorithms, Ping-I Chen and Shi-Jen Lin used the Spearman footrule distance. In this experiment, same terms will be given to the two algorithms, and for each system the system will construct a set of ranking lists on the terms (from internet) which are the most similar to the given terms. The value of this distance means the proportion that the terms keeps their positions on the ranking list. In other words, the bigger this value is, the more stable the algorithm is[6, 18].

■ **Table 5** Spearman's footrule measurement of GCD and NGD

Measurement	Algorithm	
	GCD	NGD
Spearman's footrule	0.74	0.56

Sometime has passed and the experiment was repeated. The Spearman footrule takes all the values from both algorithms into consideration. From the result (table 5) of the experiment, 56% of the terms chosen by NGD are still on the ranking lists, and the value by GCD is 74%. This result has fully demonstrated the better stability of GCD.

And this algorithm has another advantage, it is faster. By NGD we need to query 3 times (for $f(x)$, $f(y)$ and $f(x, y)$), but by GCD twice is enough, r_1 (from $f(x)$) and r_2 (from $f(y)$)[7, 20].

3.6 Summary

In this paper, algorithms to calculate the similarity of the objects are given. Based on several assumptions on the human language's representative capacity, normalized information distance (NID) is thought as the best way to express the similarity. However, as the Kolmogorov complexity is incomputable, the information entropy is used to approximate the value of Kolmogorov complexity, and as a result, the Normalized Google Distance (NGD) is obtained[33]. Although technically NGD is just one of the upper bounds of NID. Using NGD to compute the similarity of the objects is of good accuracy, and performed well in the experiments. It can fully identify the differences between numbers and colors, and for the Hierarchical clustering of authors, the result is also good. There is no doubt that to let the computer do more and more repeated simple work instead of human is one of the tendencies of the technological innovation. And this algorithm gives us a method to compute the similarity of objects in human's world which is totally different from the computer's world of digital numbers.

As the database used in NGD is the world-wide-web, which is updated at a horrendous speed, making the stability of the result a big problem. At last a new GCD algorithm is given as an improvement of NGD. The two algorithms are based on the same assumptions, but GCD computes the information distance more directly. Therefore it performs more stable than NGD in the Spearman footrule distance experiment. The precision of the GCD system was nearly 90%, making it GCD a good adaption of NGD. However, there is a problem that GCD is also not able to solve. Some words will be given new meanings, such as the emoticon. Long ago it is just a grammatically wrong and meaningless permutation of punctuation, but nowadays it is at least able to express some emotions of human. In other words, the meanings of terms will change, and this tendency is hardly predictable. As a result the stability of both these two algorithms can never reach 100%.

The entire Google data base is many terabytes, so each Google search takes a significant amount of time. It takes time to apply these two algorithms (although GCD is faster). And better adaptations with higher accuracy and less running time will be needed in the future.

References

- 1 P.M.B C. Costa Santos, J. Bernardes. Clustering fetal heart rate tracings by compression. *In: Proc. 19th IEEE Symp. Computer-Based Medical Systems*, pages 685 – 690, 2006.
- 2 P. Gács C.H. Bennett. Information Distance. *IEEE Trans. Information Theory*, 44(4):1407–1423, 1998.
- 3 L.F. Chien. PAT-tree-based Keyword Extraction for Chinese Information Retrieval. *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 50 – 59, 1997.
- 4 Wikipedia contributors. Entropy (information theory). [https://en.wikipedia.org/w/index.php?title=Entropy_\(information_theory\)&oldid=729509999](https://en.wikipedia.org/w/index.php?title=Entropy_(information_theory)&oldid=729509999), 2016.
- 5 T.M. Cover and J.A. Thomas. Wiley, New York. *Elements of Information Theory*, 1991.
- 6 A. Berger D. Beeferman. Agglomerative Clustering of a Search Engine Query Log. *Proc. ACM SIGKDD*, pages 1448 – 1463, 2000.
- 7 L. Gravano E. Agichtein, S. Lawrence. Learning to find answers to questions on the Web. *ACM Transactions on Internet Technology*, 4(2):129 – 162, 2004.
- 8 C.A. Rtanamahatana E. Keogh, S. Lonardi. Toward parameterfree data mining. *Proc. 10th ACM SIGKDD Intl Conf. Knowledge Discovery and Data Mining*, pages 206–215, 2004.
- 9 The Economist. Corpus colossal: How well does the world wide web represent human language? <http://www.economist.com/science/displayStory.cfm?storyid=3576374>, 2005.

- 10 G.A. Miller et.al. WordNet, A Lexical Database for the English Language. <http://www.cogsci.princeton.edu/wn>, 2005.
- 11 M Lapata F Keller. Using the web to obtain frequencies for unseen bigrams. *Computational Linguistics*, 29(3):459–484, 2003.
- 12 W. Meng F. Liu, C. Yu. Personalized Web search for improving retrieval effectiveness. *IEEE Transactions on Knowledge and Data Engineering*, 16(1):28 – 40, 2004.
- 13 D. Graham-Rowe. A search for meaning. *New Scientist*, pages 21 – 29, 2005.
- 14 C. Burgess K. Lund. a general model semantic representation, Brain and Cognition. *Hyperspace analogue to language (HAL)*, 30(3):5 – 17, 1996.
- 15 A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems Inform. Transmission*, 1(1):1 – 7, 1965.
- 16 T. Landauer and S. Dumais. A solution to Plato’s problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychol. Rev.*, 104:211–240, 1997.
- 17 M.E. Lesk. Word-word associations in document retrieval systems. *American Documentation*, 20(1):27 – 38, 1969.
- 18 M. Li and P.M.B. Vitányi. Algorithmic Complexity. *International Encyclopedia of the Social and Behavioral Sciences*, pages 376 – 382, 2001.
- 19 P. Ritrovato M. Gaeta, F. Orciuoli. Advanced ontology management system for personalised e-learning. *Knowledge-Based Systems*, 22(4):292 – 301, 2009.
- 20 Y. Matsumoto M. Kitamura. Automatic extraction of word sequence correspondences in parallel corpora. *Proceeding of the 4th Workshop on Very Large Corpora*, pages 78 – 89, 1996.
- 21 P. M. B. Vitanyi M. Li. IAn Introduction to Kolmogorov Complexity and Its Applications. *Springer-Verlag*, 1997.
- 22 X. Li M. Li, X. Chen. The similarity metric. *IaaEEE Trans. Information Theory*, 50(12):3250 – 3264, 2004.
- 23 H. Muir. Software to unzip identity of unknown composers. *New Scientist*, 2013.
- 24 J. Srivastava P.-N. Tan, V. Kumar. Selecting the right interestingness measure for associating patterns. *Proc. ACM-SIGKDD Conf. Knowledge Discovery and Data Mining*, pages 491 – 502, 2002.
- 25 K. Patch. Software sorts tunes. *Technology Research News*, 2003.
- 26 S.J. Lin P.I. Chen. Automatic keyword prediction using Google similarity distance. *Expert Systems with Applications*, 37(3):1928 – 1938, 2010.
- 27 Shi-Jen Lin Ping-I Chen. Word AdHoc Network: Using Google Core Distance to extract the most relevant information. *Knowledge-Based Systems*, 34:370–383, 2011.
- 28 P. Vitanyi R. Cilibrasi. A New Quartet Tree Heuristic for Hierarchical Clustering. <http://arxiv.org/abs/cs.DS/0606048>.
- 29 P. Vitanyi R. Cilibrasi. Automatic meaning discovery using Google,. <http://xxx.lanl.gov/abs/cs.CL/0412098>, 2004.
- 30 P. Vitanyi R. Cilibrasi. Clustering by compression. *IEEE Trans. Information Theory*, 51(4):1523–1545, 2005.
- 31 P. Vitanyi R. Cilibrasi, R. de Wolf. Algorithmic clustering of music based on string compression. *Computer Music J.*, 28(4):49–67, 2004.
- 32 Paul M.B. Vitányi Rudi L. Cilibrasi. The Google Similarity Distance. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):370–383, 2007.
- 33 S. Bawa S. Batra. Semantic categorization of Web services. *International Journal of Recent Trends in Engineering*, 3(2):19 – 23, 2009.
- 34 C. E. Shannon. A mathematical theory of communication. *Bell Systems*, (27):379 – 423, 1948.

- 35 E. Terra and C. L. A. Clarke. Frequency Estimates for Statistical Word Similarity Measures. *HLT/NAACL 2003, Edmonton, Alberta*, pages 37 – 162, 2003.
- 36 P. Kearney X. Chen, S. Kwong and H. Zhang. An information-based sequence distance and its application to whole mitochondrial genome phylogeny. *Bioinformatics*, 17(2):149 – 154, 2001.
- 37 M. Ishizuka Y. Matsuo. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(1):157 – 169, 2004.
- 38 N. Zhong. Representation and construction of ontologies for Web intelligence. *International Journal of Foundation of Computer Science*, 13(4):555 – 570, 2002.
- 39 N. Zhong Z.Y. Lu, Y.Y. Yao. Web log mining. *Web Intelligence*, pages 174 – 194, 2003.

4 Text Variant Graph and its construction

Guangping Li

Abstract

Variation problem is a pervasive problem in linguistic. To realize the coexistence of multiple versions of the same text in digital form, some representations are developed. Text Variant Graph is a highly readable and widely used representation among them, for whose construction a lot of tools and algorithms have been developed. In this paper, we will see how to represent the variations of text in Text Variant Graph (Section 1.1). This paper focuses on the construction of a Text Variant Graph including the algorithms of linear alignment (Section 1.2.2) and the basic ideas of algorithms in transposition detection (Section 1.2.3).

4.1 Introduction

4.1.1 Variation in linguistic

Variation is a characteristic of linguistic. There is more than one way of expressing the same thing. Speakers may vary pronunciation (accent), word choice (lexicon), or morphology and syntax (grammar) [6]. Textual variation is a problem that has affected literary texts almost since the invention of writing. Variations present in different copies of literary texts and texts in linguistic. It leads to an old conundrum: what exactly is the text in variation? With the development of the digital medium, the old conundrum gave way to the problem of how to realize the coexistence of different versions of the same text.



■ **Figure 1** [1] Different versions of the book Bible.

One of the examples of text variation are hundreds of different translations of the Bible into English, which are based on different manuscripts. It is important to find a way to save the versions of the same text and make version research operations like reading a single version or comparing two versions possible. In Figure 1 you can see different versions of the Bible.

4.1.2 Representations of text in Text Variant Graph

In the following, I will introduce a way for the visual representation of textual variation. This representation uses a directed graph model to represent variation in literal texts. For convenience we call this a *Text Variant Graph* or in short *Variant Graph*.

The Text Variant Graph

A *directed graph* is an abstract structure in computer science, which contains a

set of nodes $v_1 \dots v_n$ and a set of edges $e_1 \dots e_m$, each of which connects two nodes. An edge can be represented by an ordered pair $e_q = (v_i, v_j)$ in a directed graph, where edge e_q is directed from v_i to v_j . *In-degree* of a node is the number of edges that end adjacent to the node and the *out-degree* is the number of edges that start from the node. A path in a directed graph is a sequence of edges that connect a sequence of nodes.

The *Text Variant Graph model* is an acyclic directed graph. This graph contains contents of different versions of the same text. Each edge is labeled by an index of one version. The nodes are labeled with non-empty strings. There are three types of nodes: start nodes, normal nodes and end nodes. A start node contains the start part of the content. A start node has in-degree 0. Analogously, an end node contains the end part of the content and has out-degree 0. A normal node is a node with non-zero in-degree and non-zero out-degree. A normal node labeled by a certain version is connected by an edge to a node with same relevant version. Analogously, this normal node is also connected by an edge from a node with same relevant version. In Figure 2 you can see a Text Variant Graph of the seven versions of Genesis1:1.



■ **Figure 2** [2] A short example from the Bible visualization project BibleViz based on seven versions of the Genesis 1:1. The direction of edges is left to right.

The directed edges indicate the order of the words in the variants. For each version of the text, there exists a single path from a start point to an end point in the Text Variant Graph, in which all the nodes are part of this version and the concatenation of the contents of the nodes in the path from start to end represents the content of this version.

The common parts of the versions are stored in one node in this graph model. This graph model saves the memory by solving the high redundancy problem in variation. It also allows the basic operations in version research: reading a single version, comparing two versions, etc.

Having labeled the nodes with version information, reading a single version only needs to find a path from the start node to an end node, which contains exactly the nodes relevant to this version. By going along the path and concentrating the contents in the nodes of this path, the content of a single version is outputted. Reading a version in a Text Variant Graph is clearly $O(n)$, where n is the number of nodes in the graph. Comparing two versions in Text Variant Graph and finding the differences of versions are corresponding to compare the paths of the two versions. The common parts of them are represented by common nodes in the graph and the differences are the nodes, which belong to only one of these two versions. So this operation is also $O(n)$.

4.2 Construction of Text Variant Graph

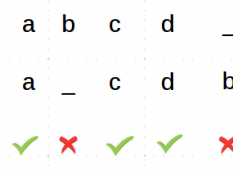
The question we consider in this section is how to construct the Text Variant Graph from a set of different versions of the same text.

4.2.1 The Gothenburg model

A collation summit and a collation workshop were held in Gothenburg and Brussel in 2009 [3]. The immediate result of these meetings was the Gothenburg model. The Gothenburg model is an agreement of modularization of the collation process into four basic well-defined steps: tokenization, alignment, transposition detection, visualization.

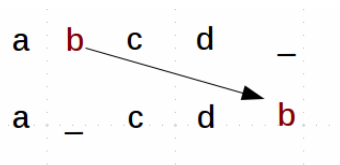
The first step is the *tokenization* of digital plain texts. In this pre-processing step, the texts can be split at any level of granularity. For example, splitting plain text based on whitespace. Normalization is an optional part of this step. After text splitting, the tokens can be normalized by a user-defined equivalence relation. For example, all the words can be converted to lower case. In Text Variant Graph model construction, each version is represented as a sequence of words as tokens. Each token is a vertex. Thus a version is a path from its start node to its end node.

The second step is *alignment*, in which a set of matched tokens are aligned via insertion of gaps such that the token sequences of all versions are lined up¹. For example, the optimal alignment for string “abcd” and “acbd” is “abcd_” and “a_cdb” (see Figure 3). In this alignment, three tokens, *a*, *c* and *d* are matched. After this step, the vertices are merged when corresponding words are aligned²



■ **Figure 3** An alignment of “abcd” and “acbd”.

The third step is *transposition detection*. A *transposition* is a permutation which exchanges two elements and keeps others remained. A transposition means same content in different positions in different versions. Some sentences are transposed in versions. The corresponding sentences in different version need to be found. If the alignment algorithm in step 2 aligns two irrelevant sentences, the alignment score is normally bad. Detecting transposed parts is a further analysis based on the alignment result. It will find the transposed sentences and the alignment algorithm can align the relevant sentences. Transposition detection is much harder than the linear alignment. Wagner has stated that transposition detection is NP-hard [14]. In Figure 4 you can see a transposition of “abcd” and “acbd”.



■ **Figure 4** Transposition of *b* in the example above.

¹ We use the word “line up” to describe the alignment of two characters in two sequences.

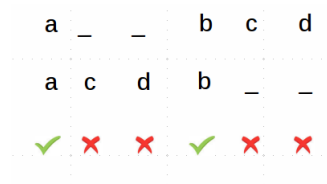
² Alignment here has been described on the level of words. For Text Variant Graph the alignment is on the level of sentences.

The fourth step and final step is the *output and visualization* of the collation results [3]. Based on different building principles, layouts and notation conventions are suggested. After construction of the graph, some output formats are provided for printing or saving the graph in digital form.

From the algorithmic point of view, alignment and transposition detection are most interesting steps. In the following, several widely used algorithms for alignment and the basic idea of transposition detection algorithms are introduced.

4.2.2 Text alignment and Text Variant Graph

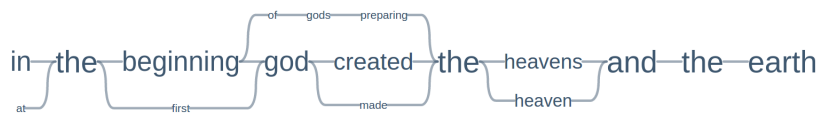
Linear alignment is the core of 4-steps Gothenburg collation workflow. In this step, the token sequences of all versions are lined up by the insertion of gaps. After insertion of gaps, the sequences must contain the same number of the tokens inclusive gaps. Alignment of two sequences is not unique. Recall the example of Figure 3, here “*abcd*” and “*acdb*” are aligned. An alternative way to align these sequences is illustrated in Figure 5. This alignment matches *a* and *b* while the previous matches *a*, *c*, *d*, and therefore is superior. Actually, the alignment in Figure 3 is optimal, in the sense that it aligns the most characters. So this alignment is not optimal. So how to find the optimal alignment is the problem to be solved in this step.



■ **Figure 5** Another alignment of “*abcd*” and “*acdb*”.

Base-Text and Oder-independence of alignment algorithm

This section gives an overview of the whole process of linear alignment, which is the core step of the construction of Text Variant Graph. In linear alignment, a version is chosen as the base text, which will be compared with other versions. Alignment algorithm is reduced to pairwise comparisons of two versions at one time. Each time when a new version is added in the collation, it must be compared with the base text. During the construction of the graph, each version must be compared with the base text and the result of alignment is merged consecutively.



■ **Figure 6** [2] Text Variant Graph with ASV as the base text.



■ **Figure 7** [2] Text Variant Graph with YLT as the base text.

So the alignment result depends on the choice of the base text and the order in which the pairwise comparisons are merged. Different alignment results lead to different forms of the Text Variant Graph. To find a suitable order of merging of the pairwise alignment results is very important. It affects the effort of merging and also the readability of the graph. A Hypothesis: More similar to the basic text, the version should be compared and merged earlier [13].

The result of linear alignment will be stored in a data structure, for example, JASON format.

After introducing the process of alignment, we will see two widely used pairwise alignment algorithms.

Needleman-Wunsch algorithm

Needleman-Wunsch algorithm is a widely used alignment algorithm. This algorithm uses dynamic programming to align two sequences of tokens [11].³

Dynamic programming is an important algorithmic technique which solves a complex problem by using the solutions of subproblems. In our alignment algorithm, the optimal alignment of two sequences is found by searching the best alignment of the subsequences. In the following, we use $A_{0\dots i}$ to represent the subsequence of A which contains the characters from the first to the $(i + 1)$ -th. The condition, to determine how $A_{0\dots i}$ and $B_{0\dots j}$ are aligned, are illustrated in the following table.

condition	Gap added	Line-up	Alignment
1	No gap at end of the alignment	A_i and B_j	$A_{0\dots i-1}$ and $B_{0\dots j-1}$.
2	One gap at the end of A	B_j and gap	$A_{0\dots i}$ and $B_{0\dots j-1}$.
3	One gap at the end of B	A_i and a gap	$A_{0\dots i-1}$ and $B_{0\dots j}$.

In this algorithm, a scoring schema S is used to evaluate the alignments. The optimal alignment is considered to be the alignment with the highest score based on the scoring schema. The scoring schema can be defined by the user himself. In this way, the user can get the optimal alignment on his demand. For example, a scoring schema can be like this: for a match one score is awarded, and for a non-match one score is reduced. If a gap is lined up with another gap (gap-matching), the score is unchanged. In our example (see Figure 5), the alignment “ $abcd_{\square}$ ” and “ $a_{\square}cdb$ ” has score 1 for 3 matches and 2 non-matches.

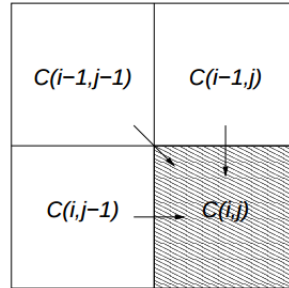
Apart from the scoring schema, a matrix C is used to record the highest score of alignments of sub-sequences A and B. The scoring matrix is an $N \times N$ matrix, in which $N = \max\{|A|, |B|\} + 1$.⁴ This matrix is also called *scoring matrix*. Each entity C_{ij} contains the highest score of Alignment of $A_{0\dots i}$ and $B_{0\dots j}$. So the entity at the bottom right corner will contain the highest score of the alignments of sequence A and B. For example, the C_{0i}

³ In this description, we use the character as the granularity.

⁴ A_0 and B_0 are considered to be a leading gap in sequences.

contains the highest score of alignments of A_0 and $B_0\dots i$. Because A_0 and B_0 are a gap-match and i gaps needs to be added after A_0 to align with $B_1\dots i$, the value of C_{0i} is equal to $-i$. In this way, the first row and the first column are filled in the scoring matrix.

After filling the first column and first row in C , the next step is to fill the internal entities.



■ **Figure 8** Three directions of filling up the internal cell.

Beyond our thoughts in dynamic programming, we can get the value in entity C_{ij} by considering three directions corresponding to the three conditions (see Figure 8). Under the first condition, the optimal alignment of $A_0\dots i$ and $B_0\dots j$ with the additional condition of the line-up of A_i and B_j is based on the optimal alignment of $A_0\dots i-1$ and $B_0\dots j-1$. The score of the optimal alignment depends on the score of this optimal alignment of $A_0\dots i-1$ and $B_0\dots j-1$ ($C_{i-1, j-1}$) and if A_i and B_j a match or not. So it is called diagonal direction. As for the second condition, to add a gap at the end of A is related in the scoring matrix the up direction. This value of the best alignment of $A_0\dots i$ and $B_0\dots j$ with adding a gap after A_i is dependent on the score of optimal alignment of $A_0\dots i$ and $B_0\dots j-1$ ($C_{i, j-1}$). Because of a gap insertion, a non-match exists. So the total score from the direction is $C_{i, j-1} - 1$. Similar to the second condition, the third condition corresponds to the left direction. The score of alignment with lining up a gap with A_i is $C_{i-1, j-1}$. From these three directions or three conditions, we will get three candidates for C_{ij} .

$$q_{diag} = C_{i-1, j-1} + S(i, j) \text{ (align } A_0\dots i-1 \text{ and } B_0\dots j-1, A_i \text{ and } B_j)$$

$$q_{up} = C_{i-1, j} - 1 \text{ (align } A_0\dots i \text{ and } B_0\dots j-1, \text{ gap and } B_j)$$

$$q_{left} = C_{i, j-1} - 1 \text{ (align } A_0\dots i-1 \text{ and } B_0\dots j, \text{ gap and } A_i)$$

The highest one of these three values will be used as the value of C_{ij} .

A matrix D called direction matrix with the same size as the scoring matrix is used to record the directions. The entity D_{ij} records the direction be chosen in filling the entity C_{ij} . The process of filling up the direction matrix is synchronous to the process of filling up the scoring matrix. After comparing the values of q_{up} , q_{left} and q_{diag} for C_{ij} , the chosen direction is recorded in D_{ij} .

After filling up the two matrices C and D , C_{nn} contains the score of the optimal alignment of sequences A and B and a traceback from D_{nn} can provide the process of getting this score. And according to the process, an optimal alignment is constructed. A diagonal direction corresponds to lining up to characters of A and B , a left direction corresponds to adding a gap to A and an up direction to adding a gap to B .

	-	a	b	c	d
-	0	-1	-2	-3	-4
a	-1	1	0	-1	-2
c	-2	-1	0	1	0
d	-3	-2	-1	0	2
b	-4	-3	0	-1	1

■ **Figure 9** Optimal alignment of “abcd” and “acdb” illustrated using the entrances of the direction matrix D.

There are variants of the Needleman-Wunsch algorithm with a different scoring schema. In our example, we give a score for a match and reduce one score for a mismatch [11]. Users can define their own score scheme based on their requirement on alignment. For example, when users prefer to compact alignment and try to avoid excessive gaps in the alignment, they can add gap penalty in the scoring schema. When a large deletion occurs, it is more convenient to be a large gap rather than a continuous set of small deletions. Hence, a continuous set of small gaps should have a worse score than a big gap. A common way to solve this problem is to add an extra gap-start score reduce in the scoring schema. The pseudo-code for the Needleman-Wunsch algorithm based on the scoring scheme (match +1, mismatch -1) is in

Algorithm 1.

```

Input : Sequence  $A$  and  $B$ 
Output : Alignment of Sequence  $A$  and  $B$ 
for  $i=0$  to  $length(A)$  do
  | for  $j=0$  to  $length(B)$  do
  | |  $C_{0j} \leftarrow j$ 
  | end
end
for  $i=1$  to  $length(A)$  do
  | Match  $\leftarrow C_{i-1, j-1} + S(A_i, B_j)$ 
  | Delete  $\leftarrow C_{i-1, j} - 1$ 
  | Insert  $\leftarrow C_{i, j-1} - 1$ 
end
AlignmentA  $\leftarrow \square$ 
AlignmentB  $\leftarrow \square$ 
 $i \leftarrow length(A)$ 
 $j \leftarrow length(B)$ 
while  $i > 0 \vee j > 0$  do
  | if  $i > 0$  and  $j > 0 \wedge D_{ij} = 1$  then
  | | AlignmentA  $\leftarrow A_i +$  AlignmentA
  | | AlignmentB  $\leftarrow B_j +$  AlignmentB
  | |  $i \leftarrow i - 1$ 
  | |  $j \leftarrow j - 1$ 
  | end
  | else
  | | if  $i > 0 \wedge D_{ij} = 2$  then
  | | | AlignmentA  $\leftarrow A_i +$  AlignmentA
  | | | AlignmentB  $\leftarrow \square +$  AlignmentB
  | | |  $i \leftarrow i - 1$ 
  | | end
  | | else
  | | | AlignmentA  $\leftarrow \square +$  AlignmentA
  | | | AlignmentB  $\leftarrow A_j +$  AlignmentB
  | | |  $j \leftarrow j - 1$ 
  | | end
  | end
end

```

Algorithm 1 : Needleman-Wunsch algorithm

The time complexity of the algorithm is $O(n^2)$. The algorithm reads in two sequences and compares each pair of the cells of the two sequences. So the run time depends on the maximum number of the cells of the two sequences and the comparison strategy of the cells. We consider here only the normal string comparison in the cell. So the time cost is $O(n^2)$, in which n is the maximum number of characters in sequences A and B .

MEDITE alignment algorithm

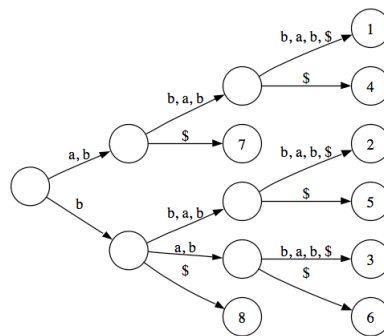
MEDITE alignment algorithm is a linear alignment algorithm developed and used in MEDITE by Julien Bourdaillet and Jean-Gabriel Ganascia [8].

The process of the alignment algorithm MEDITE has two steps. The first step is to find

the longest matching in the two sequences and align the found match exactly regardless of the rest part of the sequences. The second step is to align the rest part of the sequence with MEDITE alignment algorithm literally or other alignment algorithms. In the second step, the exact line up of the longest matching remains.

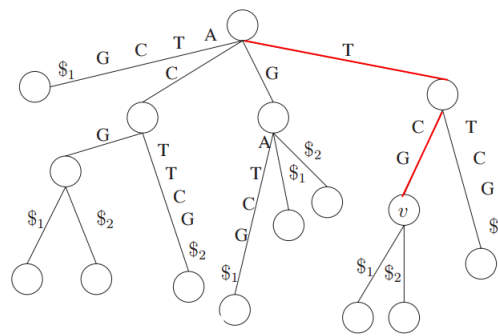
In the first step, to find the maximum match, MEDITE alignment algorithm uses the suffix tree. In linguistics, a *suffix* (also sometimes termed postfix or ending) is an *affix* which is placed after the stem of a word [4]. Informally, a suffix is the end part of a word, which can be attached to a start part to get the original word. For example, the word “APPLE” has suffixes: *E*, *LE*, *PLE*, *PPLE*, *APPLE*, \square . The last two are special suffixes, one with empty start part and one with the original word as its start part.

The Suffix tree is a space-efficient tree data structure, which is widely used in string comparison and other fields of textual analysis. A suffix tree is an ordered tree data structure which is used to store exactly the suffixes of a string. The keys (the strings) are stored in the edges. More formally, the suffix tree $T(S)$ of string S is a rooted tree with labeled edges. A path from the root to a leaf responds to a suffix of S . Two suffixes with same start part must share the corresponding start part in the path. Therefore, the form of the suffix tree of one string is unique. The construction of the suffix tree of a string S can be done in linear time [10].



■ **Figure 10** [5] Suffix tree of string “abbabbab”.

A *Generalized suffix tree* is the suffix tree of a sequence, which is concatenated by a set of sequences. The sequences involved in a Generalized suffix tree are concatenated, separated from a special sequence separator from each other. For example, the Generalized suffix tree of strings “GATCG” and “CTTCG” is the suffix tree of string “GATCG\$₁CTTCG\$₂”.



■ **Figure 11** [7] Generalized suffix tree of strings “GATCG” and “CTTCG”.

With this generalized suffix tree, we can find the longest matching, which corresponds to the longest path from the root to a node, whose subtree contains all the sequence separator symbols. The concentration of the labels of the edges in the path presents the longest common string of two sequences.

The maximal match is not unique. Multiple maximal matches can be found in a generalized suffix tree. After aligning the maximal matches found in step 1, the regions between the maximal matches will be aligned in step 2. A heuristics variation is to choose the alignment strategy in Step 2 based on the condition of the subsequences needed to be aligned.

The MEDITE alignment algorithm is not limited in pairwise comparison. A generalized suffix tree of multiple strings can be used to find the maximal match of strings. The version with multiple comparisons is often used in Computational Biology, for example for Detection of RNA Elements.

4.2.3 Transposition detection

Sentences sometimes change their positions in variation problem. The step 2 may align two sequences with a very bad score, a reason of this is: The aligned sentences are not meant to be different versions of the same sentence. The transposition detection, is to find the set of versions of same sentences.

Compared with the linear alignment, transposition detection is a much harder problem. It is a further analysis of alignment result by adding transpositions. Schmidt D. concludes his analysis of the transposition detection problem by stating detection of arbitrarily transposed words an NP-hard problem [12]. Besides this, the words are not transposed literally but often combined with small changes.

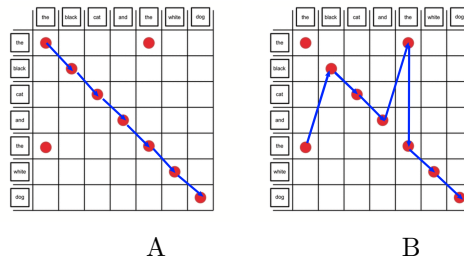
Match table

In this section, a concept by which most textual comparison tools detect transpositions will be introduced. A data structure we use here to explain this concept is *match table*. It is a version to version matrix, whose index of rows and column represents the tokens of the sequence. The cells in the match table are scored if the corresponding tokens are a match.

	the	back	of	and	the	when	dog
the	•					•	
back		•					
of			•				
and				•			
the	•				•		
when						•	
dog							•

■ **Figure 12** [13] A match table without variation.

A path of the scored cells from the upper left corner to the bottom right corner deviates a transposition interpretation of the sequences. Two paths of our example are shown. Obviously, the path in picture A (Figure 13) is more consistent with the assumption of human beings and the path in Fig B (Figure 13) is invalid. The path in picture B, which assumes the first “*the*” in sequence 1 is transferred to the second “*the*” in sequence 2 and the first “*the*” in sequence 2, which is in the same position as the first “*the*” in sequence 1, is yet transferred from the second “*the*” in the first sequence.



■ **Figure 13** [13] Two paths in match table.

Token transposition and diagonal transform

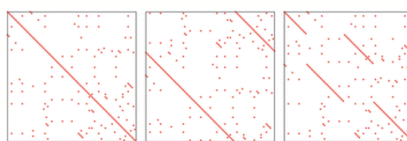
After introducing the match table, we consider a situation of transposition. The paper “Computer-supported collation of modern manuscripts: CollateX and the Beckett Digital Manuscript Project” offers an example of match table based on a six-sentence quote from Samuel Beckett’s *Stirrings Still*.

If we compare two identical copies of the text, we will arrive at the result shown in picture A (Figure 15). As our explained example without variation, we will get a path through the main diagonal. Then, we consider a situation of a transposition by moving the last sentence to the first. If we compare the original text with the text with our artificial transposition, we get the result in picture B (Figure 15). A diagonal in the top right corner depicts the swap of last sentences. If we create multiple transpositions, the corresponding match table is in picture C (Figure 15).

One night as he sat at his table head on hands he saw himself rise and go. One night or day.
 For when his own light went out he was not left in the dark. Light of a kind came from the one
 high window. Under it still the stool on which till he could or would no more he used to mount
 to see the sky. Why he did not crane out to see what lay beneath was perhaps because the
 window was not made to open or because he could or would not open it.

Text A: excerpt from Samuel Beckett’s *Stirrings Still*.

■ **Figure 14** [13] A six-sentence quote from Samuel Beckett’s *Stirrings Still*.



■ **Figure 15** [13]

A B C

*We add some vertical dotted line in the match table to get the optimal path of transposition.

A basic idea of transposition detection is: The transposition in variation is performed by breaking main diagonal into smaller diagonals in match table. Thus, the problem of finding transpositions is transformed to finding break-up path of main diagonal in match table. There are two steps in algorithms using this idea. At first, the match table is calculated by comparing tokens of two versions. In this step, the parts with best scores in alignment are ignored. The time complexity of this step is $O(n^2)$. The second step is finding transposition approximately. This step relies on a mathematical approximation of finding a set of sub-diagonals in a matrix. Each of this smaller diagonal corresponds two parts in two versions. If a small diagonal is not on the main diagonal, the corresponding two parts will be supposed to be a transposition in two versions. These algorithms, which rely on matrix algorithms, can also be used to provide transposition detection in multiple versions by using a multidimensional matrix as match table.

4.3 In closing

The variation problem is a pervasive problem in linguistic. In the process of collation of modern manuscripts, an efficient model is needed. The graph model, Text Variant Graph is a compact, efficient and highly readable form of visualization of multiple versions of the same text. The graph model allows several edit operation. In this work, the steps to Text Variant Graph are explained. The key steps are linear alignment and the transposition detection. For linear alignment, there are several algorithms with different ideas and data structures. We present two most used algorithms among them: Needleman-Wunsch algorithm and MEDITE alignment algorithm. For transposition detection, the basic idea of most detection algorithms is finding diagonals in match table is explored.

References

- 1 Different versions of bible.
- 2 The holy bible in 7 various english translations (verse by verse). <http://www.informatik.uni-leipzig.de:8080/BibleViz/>. Accessed: 2016-07-14.
- 3 Project collatex. <http://collatex.net/doc/>. Accessed: 2016-07-14.
- 4 Suffix. <https://de.wikipedia.org/wiki/Suffix>. Accessed: 2016-07-14.
- 5 Suffix tree. <https://de.wikipedia.org/wiki/Suffixbaum>. Accessed: 2016-07-14.
- 6 Variation (linguistics). [https://en.wikipedia.org/wiki/Variation_\(linguistics\)](https://en.wikipedia.org/wiki/Variation_(linguistics)). Accessed: 2016-07-14.
- 7 Srinivas Aluru, Pang Ko, and Srinivas Aluru. Suffix tree applications in computational biology. In *Handbook of Computational Molecular Biology*, pages 6–1. Chapman and Hall/CRC, 2005.
- 8 Julien Bourdaillet and Jean-Gabriel Ganascia. Medite: A unilingual textual aligner. In *Advances in Natural Language Processing*, pages 458–469. Springer, 2006.
- 9 Dan Gusfield. Linear-time construction of suffix trees. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, 1997.
- 10 Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.
- 11 Desmond Schmidt. Merging multi-version texts: a general solution to the overlap problem. In *Balisage: The Markup Conference*, 2009.
- 12 J van Zundert, R Haentjens Dekker, D Van Hulle, Vincent Neyt, and Gregor Middell. Computer-supported collation of modern manuscripts: Collatex and the beckett digital manuscript project. *LLC: the journal of digital scholarship in the humanities*, 30(3), 2014.
- 13 Robert A. Wagner. On the complexity of the extended string-to-string correction problem. In *Proceedings of Seventh Annual ACM Symposium on Theory of Computing*, page 218–223. Albuquerque, New Mexico, USA, 2006.

5 Block Edit Models for Approximate String Matching

Moritz Klammler

Abstract

Approximate string matching means identifying and rating similarities between strings. This is usually done by considering how much editing would be required to transform one string into another. Traditional models only consider character-by-character editing but this doesn't take the higher-level structure into account. Unfortunately, adding more powerful editing operations quickly makes the problem become \mathcal{NP} -hard. It is therefore essential to consider refinements that maintain polynomial solvability and approximation algorithms. In this article, we will look at the traditional Levenshtein distance which is based on character-wise editing and is in \mathcal{P} , editing with move operations which is \mathcal{NP} -complete, minimum common string partitioning which is also \mathcal{NP} -complete and, finally and foremost, the model of block-level approximate string matching introduced by Lopresti and Tomkins which is \mathcal{NP} -complete if constrained and in \mathcal{P} under certain relaxations. In order to manage the complexity of editing with move operations, we will investigate a restricted version thereof that can be solved in polynomial time and the greedy heuristic, which is a very fast approximation algorithm. For minimum common string partitioning, we will discuss its fixed parameter tractability. Finally, we will show the idea of the proof that constrained block-level approximate string matching is \mathcal{NP} -complete and present the idea of the polynomial algorithms to solve the relaxed problems. We will conclude our survey by an estimation of the potential benefits and drawbacks of some models.

5.1 Introduction

The basic problem that is studied in this article, *approximate string matching*, can be thought of as the following task: given a *source string* $A \in \Sigma^*$ and a *target string* $B \in \Sigma^*$ over an alphabet Σ with $|A| = m$ and $|B| = n$, transform A into B by applying a sequence of yet to be defined *editing operations*. In general, there will be more than one way to achieve the task. Among these, we seek an “optimal” one. In order to remove the quotation marks around the word “optimal”, we assign a *cost* – expressed as a, usually integral, number – to each edit operation and associate with an edit sequence the accumulated costs of its constituting operations. An optimal editing sequence is one with minimum cost. The *edit distance* between A and B is the cost of an optimal edit sequence to transform A into B . Instead of “transform into” we also say “match”.

In order to preserve space and improve readability, we will use the variables A , B , m , n and Σ throughout the remainder of this article with the meaning given to them above without defining them again every time.

We shall investigate different sets of allowed edit operations, called a *model*, with associated costs and study the algorithmic difficulty of finding optimal editing sequences and therefore edit distances in these models.

Before we look into more manageable edit models, let's get an intuition by considering an edit model that matches very closely how a human editor would nowadays think about transforming text. Pick your editor of choice and define the set of allowed operations as the operations provided by that editor. The cost function can be defined in several ways – for example, a cost could be assigned to each key-stroke. “Many” people consider the challenge of finding narrow upper bounds for optimal editing sequences in this model a joyful mental

exercise and entire web-sites¹ are dedicated to this “sport”, often referred to as *golfing*, with the obvious analogy to classical golf sport². Given that a powerful text editor will come with a Turing complete scripting language, it is clear that edit distance in that model will be undecidable as it basically boils down to Kolmogorov’s complexity.³ We will therefore want to restrict ourselves to less powerful models that are at least computable. Ideally so with feasible computational effort.

Scientific applications of approximate string matching arise for example in genomics where biologists might ask how “close” the DNA (or RNA for that matter) of one species is to the DNA of another species and thus draw conclusions about possible evolutionary relationships. (Recall that DNA is composed of linear strands of molecules drawn from a finite set of nucleic acids so the textual representation of DNA as strings is straightforward.) Lopresti and Tomkins [9] show an example of such analysis and cite another source [6, p.96] which claims that – at least by the time (1991) – manual inspection of a graphical visualizations was / is the go-to strategy for identifying similarities between DNA sequences.

Many papers on the subject seem to be motivated by genetics. It is also worth noting that DNA strands can be quite long – potentially much longer than English sentences – so algorithmic complexity might be an ample concern of this discipline.

Other applications involve the analysis of natural-language text where edit distances are used as a low-level tool in order to construct so-called *text variant graphs* that visualize similarities and variations among short texts [11]. In literary science, these visualizations can be used to supplement a comparative discussion of the versions or to discover even new aspects of the correlations between them.

Tichy [13] has studied a very similar problem with a focus on reducing the amount of storage required for revision control systems. The systems he considered would store the history of an evolving (software) project by keeping a base version (that could either be the initial or the most recent version) and defining other versions in terms of recursively applied *deltas* to that version. (Called *forward* or *backward* deltas depending on which revision was used as the baseline.) Since a delta can be thought of as a sequence of edit operations, finding an optimal sequence would have helped reducing the required amount of storage (and presumably speeding up application of the deltas).

5.2 Character-Based Editing (Levenshtein Distance)

A classical similarity measure for strings is the so-called *Levenshtein distance* which has become a standard item in the algorithm engineer’s toolbox. Since the original publications from the 1960s are not very easily accessible, the reader is instead referred to a more recent publication by Navarro [10] that compares several character-based distance functions, including the Levenshtein distance. The primary work may be found via that paper’s references.

¹ <http://vimgolf.com/>

² <https://en.wikipedia.org/wiki/Golf>

³ For a fixed Turing-complete description language over Σ (such that the description can be fed into a universal Turing machine to simulate the described machine) and a string $s \in \Sigma^*$, the *Kolmogorov complexity* of s is the length of a shortest description of a Turing machine that, when started with an empty tape, outputs s and then halts. The Kolmogorov complexity relates to the entropy in a string. It is not computable but more or less tight upper bounds can be determined for certain strings.

■ **Table 1** The Levenshtein editing model.

operation	cost function
insert single character	$\delta_{\text{ins}} : \Sigma \rightarrow \mathbb{N}_0$
delete single character	$\delta_{\text{del}} : \Sigma \rightarrow \mathbb{N}_0$
replace single character	$\delta_{\text{sub}} : \Sigma \times \Sigma \rightarrow \mathbb{N}_0$

In the Levenshtein model, there are three allowed operations which are rated by three corresponding cost functions as shown in table 1 where the understanding is that an editing operation, say, inserting a character $\sigma \in \Sigma$ will have cost $\delta_{\text{ins}}(\sigma)$.

One might often find it sufficient to assign unit cost to all operations but arbitrarily sophisticated choices are possible. For example, assigning a lower cost to replacing a character by its corresponding lower- or upper-case pendant could make the model partially or completely case-insensitive. Likewise, assigning reduced or zero cost to the insertion and deletion of punctuation characters might be desired (if they are not pruned from both strings upfront). As a third example, δ_{sub} might be defined such that correcting common typos (such as slipping fingers to an adjacent key on the keyboard or confusing the letters “Y” and “Z” by switching from English to German keyboard layouts, where their locations are inverted) is made cheaper.

Finding the edit distance in the Levenshtein model is straight-forward and is a popular textbook exercise in dynamic programming, also known as *Wagner-Fischer algorithm*. See again Navarro [10], transitively including references therein if desired. The algorithm always finds the optimal solution and runs in time $\mathcal{O}(mn)$, so approximate string matching in the Levenshtein model is in \mathcal{P} .

5.3 Editing with Move Operations

While simple and efficient, the Levenshtein model falls short for modeling how the human mind approaches similarity of strings. For example, if merely the order of words or even larger chunks of text is permuted, we would like the strings to be considered still fairly “close together”. The Levenshtein model cannot provide this because there is no way to move sub-strings around other than deleting and re-inserting each character at a time.

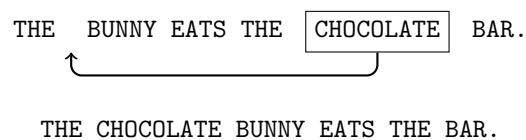
Therefore, it might seem natural to add move operations to the set of allowed operations. Of course, moving a single character doesn’t provide much benefit over deleting and inserting it again elsewhere. It will be more interesting, however, to look at *block moves*: relocation of blocks of text of arbitrary length. Such a block move is visualized in figure 1. There is a distinction to be made between block moves that draw from the original source string and such moves that take character sequences generated by previous edit operations. The latter type is referred to as a *recursive move* and visualized in figure 2.

We drop the concept of cost *functions* and instead associate a fixed cost with insertion and deletion of any character as well as moving of a block of text of arbitrary size. Note that we have also abandoned the concept of replacing individual characters (which is trivially compensated by a deletion followed by an insertion). The author does not know whether and what properties of this edit model that are discussed in the following would have to be revised, would these simplifications not be made. Table 2 summarizes the new edit model.

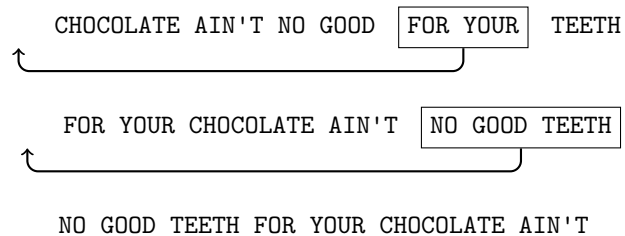
■ **Table 2** The editing model with block moves.

operation	cost
insert single character	$c_{\text{ins}} \in \mathbb{N}_0$
delete single character	$c_{\text{del}} \in \mathbb{N}_0$
move sequence of characters	$c_{\text{mov}} \in \mathbb{N}_0$

■ **Figure 1** A block move operation.

THE BUNNY EATS THE CHOCOLATE BAR.

 THE CHOCOLATE BUNNY EATS THE BAR.

■ **Figure 2** Editing with recursive block move operations. Note how in the second step, the string “no good teeth” is moved, which doesn’t appear in the original string.

CHOCOLATE AIN'T NO GOOD FOR YOUR TEETH

 FOR YOUR CHOCOLATE AIN'T NO GOOD TEETH
 NO GOOD TEETH FOR YOUR CHOCOLATE AIN'T

5.3.1 Complexity

Shapira and Storer [12] give a proof that approximate string matching in this model is an \mathcal{NP} -complete problem. Their proof uses a reduction of the *bin packing* problem⁴, which is known to be \mathcal{NP} -complete in the strict sense⁵.

In the same paper, they show that if $\text{dist}(A, B)$ is the distance between strings A and B for some fixed values of c_{ins} , c_{del} and c_{mov} , then the distance using the same parameters but disallowing recursive moves is bounded above by $3 \times \text{dist}(A, B)$. Consequently, the model is \mathcal{NP} -complete, regardless of whether recursive moves are allowed.

Ann et al. [1] show that the “problem can be solved by the polynomial-time optimization algorithms if some reasonable restrictions are applied. The restricted variations [...] involve character insertions, character deletions, block copies and block deletions.” Note that there are no block moves but instead copies and deletions which can simulate moves.

Their “reasonable restrictions”, which they adopted from another paper by Ukkonen [14], involve partitioning the string into a dynamically changing *active* suffix and *inactive* prefix. Initially, the entire string is active. As editing proceeds, the active part shrinks until the entire string eventually becomes inactive. Edits are only allowed to modify prefixes of the active part which subsequently become inactive.⁶

The paper considers three kinds of copies, referred to as *external*, *internal* and *shifted*. External copies insert portions of the original (unedited) string while internal copies are what we have previously defined as recursive moves except that the sub-string is copied instead of moved. Shifted copies allow character-wise shifts of the copied text. For example, if the alphabet is the ASCII character set and spaces and punctuation are ignored for a moment, then a shift of -32 means conversion from lower-case to all-caps.

In addition to the three copying alternatives, the paper also considers three alternative cost measures referred to as *constant*, *linear* and *nested*. The constant cost measure assigns the same cost to any copying or deletion operation, no matter how long the copied or moved sub-string is, while the linear measure assigns it a cost that is a constant base cost plus a variable cost linear in the length of the sub-string. The nested model allows additional modification of the copied strings and sets the cost to a fixed base cost plus the character-based edit distance of the edit.

Three combinations are evaluated in the paper and polynomial algorithms using dynamic programming with additional pre-processing are presented. The pre-processing step involves preparing a suitable data structure for the actual algorithm to use. The results are shown in table 3.

⁴ The *bin packing* problem is defined as follows: given a set of n “items” $X = \{x_1, \dots, x_n\}$, each with an associated “weight” given by a function $w : X \rightarrow \mathbb{N}$, and a “per-package weight limit” $k \in \mathbb{N}$, as well as an integer $m \in \mathbb{N}$, does there exist a “packaging”, that is, a partitioning of X into at most m sub-sets $P = \{p_1, \dots, p_m\}$ such that for the individual weight of each “package” p , the inequality $\sum_{x \in p} w(x) \leq k$ holds. The version of the problem used in the proof actually requires that $\sum_{x \in p} w(x) = k$ but it is not hard to see that this additional requirement is easily dealt with by adding $mk - \sum_{x \in X} w(x)$ additional items of unit-weight.

⁵ A problem is *\mathcal{NP} -complete in the strict sense* if it remains \mathcal{NP} -complete even if a unary encoding is used for all numbers in the input.

⁶ The reader who is inclined to read up on this in the cited paper shall be warned that the authors use the variables A and B for different purposes than we do here.

■ **Table 3** Worst-case complexity of the algorithms presented by Ann et al. [1]. The column of allowed block-copy operations is meant as a disjunction. That is, more than one kind of block-copy operation is allowed. The “pre-processing” and “dynamic programming” columns do not refer to alternatives but two mandatory steps that both have to be carried out sequentially.

allowed block-copy operations	cost measure	pre-processing	dynamic programming
external, internal, shift	constant	$\mathcal{O}(n + m^2)$	$\mathcal{O}(nm)$
external, internal	linear	$\mathcal{O}(n + m^2)$	$\mathcal{O}(nm \log(m))$
external, internal	nested	$\mathcal{O}((n + m)m^2)$	$\mathcal{O}(nm^2)$

5.3.2 Greedy Heuristic

Another approach to combat the \mathcal{NP} -complete nature of the problem is to leave the problem definition unaltered but accept sub-optimal solutions, that is, *approximations*. One particular approximation that has received a lot of attention is the so-called *greedy heuristic* (algorithm shown in figure 3) which was first suggested by Shapira and Storer [12] in 2002. The idea is very straight-forward: by repeatedly replacing longest common sub-strings in both the source and the target string with unique single symbols, the problem is converted into one where both strings do not contain identical sub-strings beyond single characters. This instance is then solved by applying the standard Wagner-Fischer algorithm (which doesn’t consider block moves). Move operations can later be re-identified and used to correct the computed distance. The main step of the algorithm is visualized in figure 4. The last loop in that algorithm is called `check_move` by the authors. It looks for pairs of insertions and deletions of the same symbol and replaces them with a move operation. (This only makes sense of course if it results in a reduction of the net cost which is assumed here.) For this post-processing step to be efficient, additional arrays (called η_{ins} and η_{del} in the listing) are used that keep track of the number of insertions and deletions. The Wagner-Fischer algorithm has to be modified to populate those arrays accordingly. The maximum number of symbols in the strings given to the Wagner-Fischer algorithm (and therefore the length of these arrays) is bound by the length of the input (and probably much lower).⁷

When a suffix tree is used for finding the longest common sub-strings, and a new tree is constructed in each iteration of the main loop, the algorithm has complexity $\mathcal{O}(mn)$ [12]. In 2014, Goldstein and Lewenstein [5] presented a major improvement of the algorithm by re-using the suffix tree instead of re-constructing it anew in each iteration. This cuts down the complexity of the pre-processing step to $\mathcal{O}(m + n)$ which seems impossible to be lowered any further.

Shapira and Storer [12] alleged that the greedy algorithm would achieve an approximation quality of $\log(m)$ but this claim turned out to be based on flawed reasoning. Chrobak, Kolman, and Sgall [3] could show that the actual approximation quality is rather m^γ with $0.43 \leq \gamma \leq 0.69$. Kaplan and Shafir [7] could improve their result and show that $\gamma \leq 0.46$ is a tight lower bound. Their proof assumes that $|\Sigma| \in \mathcal{O}(\log(m))$.

⁷ The paper by Shapira and Storer [12] is not very clear here about its use of variables for the required storage. The numbers mentioned in the text are partially based on own reasoning. It is clear that the greedy step might *increase* the size of the alphabet, although not the length of either string. Since characters that never occur in any string need not be considered, the number of characters in both input strings together appears to be a safe upper bound.

■ **Figure 3** The greedy heuristic suggested by Shapira and Storer [12]. For simplicity, unit cost per operation is assumed.

```

PROCEDURE Greedy
INPUT
  A : STRING of length m
  B : STRING of length n
OUTPUT
  δ : INTEGER
VARIABLES:
  ηins[1, ..., m + n] : INTEGER
  ηdel[1, ..., m + n] : INTEGER
  i : INTEGER
BEGIN
  WHILE TRUE DO
    Find longest common sub-string s of A and B.
    IF |s| ≤ 1 THEN BREAK FI
    Choose arbitrary character x not currently present in either A or B.
    Replace the same number of occurrences of s in A and B with x.
  DONE
  CALL StandardDistance(A, B, ηins, ηdel, δ)
  FOR i = 1 TO m + n DO δ ← δ - min(ηins[i], ηdel[i]) DONE
END

```

■ **Figure 4** The repetitive substitutions of longest common sub-strings with arbitrary new symbols during the main loop of the greedy heuristic illustrated by example.

CHOCOLATE WITH GARLIC

α WITH GARLIC

α WITH β

αγβ

GARLIC WITHOUT CHOCOLATE

GARLIC WITHOUT α

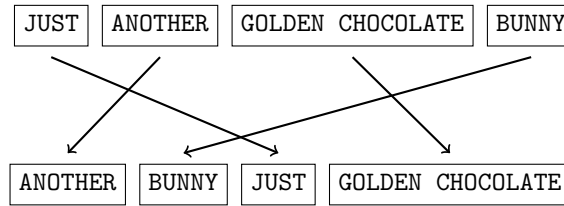
β WITHOUT α

βγ OUT α

■ **Table 4** The editing model behind minimum common string partition.

operation	cost
move sequence of characters	1

■ **Figure 5** An instance of MCSP with $k = 4$.



The algorithm was modified by Kolman and Waleń [8] to obtain an $\mathcal{O}(k^2)$ approximation (independent of the input length) for the special case that each input string contains at most k repetitions of any given character. (Note how this conflicts with the previously mentioned assumption that the alphabet size is logarithmic in the length of the strings.)

5.4 Minimum Common String Partition

Moving further along with our survey of different edit models, let's consider the case next where the only supported operation is moving blocks of text at unit cost while recursive moves are disallowed. Obviously, if we can neither insert nor delete any characters, then A and B had better be permutations of the same multi-set of characters to begin with or the distance cannot be finite. The edit model is formally defined in table 4. Edit distance in this model is also referred to as *minimum common string partition* or MCSP for short. If $\text{dist}(A, B) = k$, then both A and B are permutations of the same k -tuple of text blocks. This is illustrated in figure 5.

5.4.1 Complexity

It is not hard to see that MCSP can be viewed as a special case of editing with move operations if the cost for insertions and deletions is set to infinity and the cost of moving to one. More interestingly, Shapira and Storer [12] could also show that – if insertion, deletion and movement all have unit cost – an instance of string editing with block moves can be transformed into an instance of MCSP that has the same cost. Note that it was not required on the original problem that the two strings be permutations of the same multi-set of characters.

As a consequence, it is clear that MCSP will be \mathcal{NP} -complete, too.

5.4.2 Fixed Parameter Tractability

Let $k \in \mathbb{N}$ be an upper bound for the number of blocks in the optimal solution and $r \in \mathbb{N}$ the maximum number of consecutive repetitions of any given sub-string in either string, called

the *repetition number*.

$$r = \max \{i \in \mathbb{N} : \exists v \in \Sigma^+ : \exists u, w \in \Sigma^* : (A = uv^i w) \vee (B = uv^i w)\} \quad (1)$$

Furthermore, let $l \in \mathbb{N}$ bet the minimum distance between two “cuts” (length of a smallest block) in either string in an optimal solution. Define $d = n/l$ as the *distance ratio*. Note that $d \geq k$.

Damaschke [4] showed in 2008 that MCSP is *fixed parameter tractable* in (d, r) as well as in (k, r) . For fixed $(d, r) \in \mathbb{N}^2$, an optimal solution can be found in $\mathcal{O}(n)$ and in $\mathcal{O}(n \log(n))$ for fixed $(k, r) \in \mathbb{N}^2$.

While the algorithm is not exactly straight-forward, the basic idea for the case that (d, r) is fixed is simple. Since the distance between any two “cuts” in a string is at least $l = n/d$, we can divide each string into fixed-sized segments. None of these segments can contain more than one cut. After guessing an initial sub-set of the segments that contain cuts (referred to as *fragile* segments by Damaschke), the remaining segments that are not expected to contain cuts (referred to as *conserved*) have to be *aligned*. If the segment size was chosen sufficiently small (but still linear in l) then between each pair of matched blocks, there must be at least one segment that can be aligned to an exact match. The alignment need not be unique (consider repetitions in the string) so additional guessing is required. This alignment step corresponds to guessing a matching permutation. If an alignment turns out to be not possible, then the initial guess was wrong and has to be discarded. Otherwise, the algorithm proceeds by constructing a set of linear equations that either have as solution the exact positions of the cuts in a valid solution or no solution if one of the guesses was wrong. It is not too hard to see that the number of possible guesses only depends on d and r . The paper also proves that solving the remaining linear system is possible in $\mathcal{O}(n)$ which shows the overall result. If we fix (k, r) instead of (d, r) , the reasoning becomes more complicated.

It is intuitively clear why fixing k is useful. The reason why the solution must also depend on r is less clear. In fact, Damaschke wonders whether the restriction is actually necessary at all and points out that a similar \mathcal{NP} -complete problem – minimum common integer partitioning – is in fact fixed parameter tractable only in k . Yet, nobody seems to have found a way to show the same for MCSP so far.

Meanwhile, however, Bulteau et al. [2] were able to lower the complexity of solving the problem for fixed (k, s) where s is the maximum number of occurrences of any letter in either input string⁸ to $\mathcal{O}(n)$ as well. The “constant” (due to fixing k and s) factor hidden by the “ \mathcal{O} ” is bound above by $\mathcal{O}(ks^{2k})$. While their algorithm is not at all obvious, the paper explains it in good detail and they made a prototype implementation available for public inspection⁹.

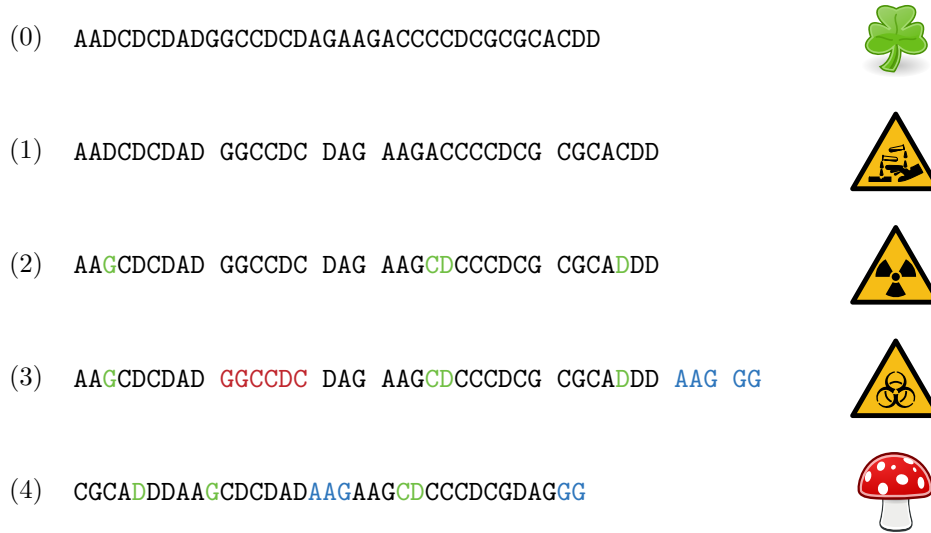
5.5 Block-Level Approximate String Matching

As a last edit model, we turn our attention to the oldest – and perhaps most intuitive – block-editing model for the scope of this article. In 1997, Lopresti and Tomkins published

⁸ Beware that Bulteau et al. use the letter d instead of s which may confuse the hell out of you if you have just previously read Damaschke’s paper.

⁹ It can be downloaded from <http://fpt.akt.tu-berlin.de/mcsp/> although, unfortunately, they omitted to mention any kind of software license so the code cannot really be used legally for anything beyond an academic reading.

■ **Figure 6** A thought experiment to motivate the block edit model for approximate string matching. (0) Some random DNA strand. (1) The DNA strand is broken into pieces by the application of chemicals. (2) The DNA undergoes some local mutations due to exposure to radioactive radiation. (3) The pieces of DNA mix with other biologic material. (4) Finally, the pieces of DNA re-combine in different order and some DNA is lost. A new species is formed.



■ **Table 5** The editing model corresponding to the work of Lopresti and Tomkins [9].

operation	cost
insert single character	$c_{\text{ins}} \in \mathbb{N}_0$
delete single character	$c_{\text{del}} \in \mathbb{N}_0$
break into block (sub-string)	$c_{\text{blk}} \in \mathbb{N}_0$
fuzzy matching of blocks	$\text{dist} : \Sigma^* \times \Sigma^* \rightarrow \mathbb{N}_0$

a model [9] that might be understood by an analogy from genetics. Interpreting the source and target strings as DNA sequences, one transits from A to B by cutting A into smaller pieces, possibly mutating some pieces somewhat, losing some of them, adding new material and finally re-combining all the pieces in a potentially different order. This informal idea is visualized in figure 6. Their model allows sub-strings to be matched in a *fuzzy* manner – corresponding to the mutation step in the aforementioned analogy. Table 5 summarizes the model but the table does not describe it as crisply as it did the other models.

Breaking the source and target strings into blocks may be subject to certain constraints.

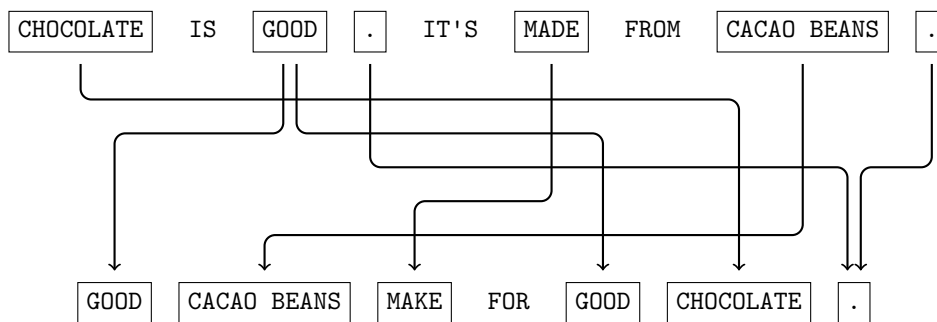
- The blocks form a *cover* if no characters are “left over”, that is, each character is contained in *at least* one block.
- The blocks are *disjoint* if no characters are “re-used”, that is, each character is contained in *at most* one block.

All four combinations of “cover” and “disjoint” (visualized by example in table 6) are

■ **Table 6** Examples for the four possible combinations of breaking the string “chocolate” into blocks with regard to their property of being *disjoint* or a *cover*.

	disjoint	not disjoint
cover	{CHOCO, LATE}	{CHOC, OCL, OCLATE}
no cover	{HOC, ATE}	{HOC, CAT}

■ **Figure 7** Block-level approximate string matching. Note how “made” is matched to “make” and how the blocks are neither disjoint nor a cover for either strings.



possible and define meaningful constraints. These may be applied to the source and target string individually. For example, we might require a disjoint cover for A and only disjoint blocks for B (or any other combination).

5.5.1 Complexity

As it turns out, if we require that the breaking into blocks of both A and B shall be constrained to disjoint covers, set $c_{\text{ins}} = c_{\text{del}} = \infty$, $c_{\text{blk}} = 1$ and define

$$\text{dist}(x, y) = \begin{cases} 0, & x = y \\ \infty, & x \neq y \end{cases}$$

for $x, y \in \Sigma^*$ then we obtain an MCSP instance. Necessarily, we conclude that block-level approximate string matching with arbitrary costs and distance function is \mathcal{NP} -complete if both, the source and the target string, are constrained to disjoint covers.

Actually, as Lopresti and Tomkins [9] prove, the problem is \mathcal{NP} -complete if and only if both strings are constrained in *some* way. If either of them is unconstrained, the distance can be computed by a polynomial-time algorithm. Table 7 summarizes the results of their paper.

5.5.2 \mathcal{NP} -Completeness

The proof of \mathcal{NP} -completeness given by Lopresti and Tomkins [9] works via a reduction of *sequencing with release times and deadlines* which is strictly \mathcal{NP} -complete (c.f. footnote 5). The problem is defined as follows: given a set of *jobs*, each with an associated *duration*, *re-*

■ **Table 7** Complexity of block-level approximate string matching depending on the constraints placed upon the breaking into blocks of source A and target string B where we still have $|A| = m$ and $|B| = n$.

source text	target text			
	unconstrained	disjoint	cover	disjoint cover
unconstrained	$\mathcal{O}(m^2n^2)$	$\mathcal{O}(mn^2)$	$\mathcal{O}(m^2n^2)$	$\mathcal{O}(mn^2)$
disjoint	$\mathcal{O}(m^2n)$	$\mathcal{N}\mathcal{P}\mathcal{C}$	$\mathcal{N}\mathcal{P}\mathcal{C}$	$\mathcal{N}\mathcal{P}\mathcal{C}$
cover	$\mathcal{O}(m^2n^2)$	$\mathcal{N}\mathcal{P}\mathcal{C}$	$\mathcal{N}\mathcal{P}\mathcal{C}$	$\mathcal{N}\mathcal{P}\mathcal{C}$
disjoint cover	$\mathcal{O}(m^2n)$	$\mathcal{N}\mathcal{P}\mathcal{C}$	$\mathcal{N}\mathcal{P}\mathcal{C}$	$\mathcal{N}\mathcal{P}\mathcal{C}$

lease time and *deadline*, is there a non-overlapping (single-processor) scheduling that ensures that each job is started at or after its release time and completes before its deadline?

For the proof, the characteristics of a job are encoded in a unary fashion as are the available time-slices. For each of the cases listed as $\mathcal{N}\mathcal{P}$ -complete in table 7 (exploiting symmetry), a distance function dist is crafted that has the property that a zero-cost matching will exist if and only if a valid scheduling exists.

5.5.3 Polynomial Algorithm

For the remaining cases (where at least one string is unconstrained), Lopresti and Tomkins [9] present exact polynomial algorithms.

The algorithms use dynamic programming to find the optimal solution recursively. As an example, we will discuss the case where A is unconstrained and B is constrained to be a disjoint cover. The other polynomial cases listed in table 7 can be solved in a similar manner by adjusting the cost functions and lookup tables. See the cited paper for details.

The algorithm begins by computing a $m \times m$ lower triangular cost-matrix W^1 with

$$W^1[i][j] = c_{\text{blk}} + \min_{1 \leq k \leq l \leq n} \left\{ \text{dist}(A[i : j], B[k : l]) \right\} \quad (2)$$

for $1 \leq i \leq j \leq m$. The element $W^1[i][j]$ stores the cost to optimally match the sub-string $A[i : j]$ to any sub-string of B plus the constant cost c_{blk} for creating that block. If one is interested not only in the distance but also the actual matching, the matrix (or a supplementary data structure) should also record the indices k and l .

Then the m -element array \mathcal{M} is defined recursively via

$$\mathcal{M}[i] = \min_{j < i} \left\{ \mathcal{M}[j] + W^1[j+1][i] \right\} \quad (3)$$

where $\mathcal{M}[0] = 0$. The element $\mathcal{M}[i]$ gives the distance between the prefix $A[1 : i]$ and B . Consequently, $\mathcal{M}[m] = \text{dist}(A, B)$. Again, if the actual matching is to be re-constructed, the values of j should be stored somewhere as well.

If W^1 is already given, evaluating $\mathcal{M}[m]$ is possible in $\mathcal{O}(m^2)$ steps using a straightforward iterative approach.

How much does it take to computing W^1 ? There are $\mathcal{O}(m^2)$ non-zero entries that need to be computed. Approaching the task naïvely, for each of these, $\mathcal{O}(n^2)$ possible sub-strings of B have to be considered. If dist is the Levenshtein distance, then evaluating each of those costs another $\mathcal{O}(mn)$ steps. In total, this leads to a complexity of the naïve approach of $\mathcal{O}(m^3n^3)$ steps. While still polynomial, the growth of this function is quite steep in practice.

■ **Table 8** Summary of allowed operations in various edit models.

	char.	char.	char.	block moves		fuzzy
	ins.	del.	sub.	normal	recursive	
Levenshtein	✓	✓	✓	✗	✗	✗
edit w. move	✓	✓	✗	✓	(?)	(?)
MCSP	✗	✗	✗	✓	✗	✗
blk. approx.	✓	✓	✗	✓	✗	✓

Fortunately, at least if dist is the Levenshtein distance, the computation can be greatly optimized down to only $\mathcal{O}(m^2n)$ by re-using results that were already computed. How this can be done is explained in Lopresti and Tomkins [9]. Given that the sheer size of the result matrix which cannot be changed already contributes a factor of m^2 , the complexity is quite remarkable. In any case, computing W^1 dominates the computation of \mathcal{M} and therefore defines the overall complexity of the algorithm.

Very similar algorithms can be formulated for the cases where other restrictions are applied to at most one of the strings and are accessible in Lopresti and Tomkins [9].

5.6 Conclusion

Approximate string matching is finding or rating similarities between strings. Simple character-based models like the Levenshtein distance exist but yield results that are “too localized” by failing to consider similarities of distant blocks of text.

Allowing block edit operations quickly makes the problem \mathcal{NP} -hard. Since solving an \mathcal{NP} -hard problem will be unfeasible for all but the tiniest inputs, ways have to be sought out of this. There are basically two approaches to get back to polynomial algorithms.

- Alter the problem definition such that it becomes manageable again.
- Accept approximations of the optimal solution.

The first approach is taken by Ann et al. [1], Damaschke [4] and in a broader sense also by Lopresti and Tomkins [9]. The approximation path is followed by Shapira and Storer [12]. The allowed operations in the various editing models are summarized in table 8.

Even though papers about block-edit models for approximate string matching have appeared decades ago (such as the 1984 paper by Tichy [13]), the topic remains a field of active research up to the present day. The breakthrough on the grounds of the greedy heuristic presented by Goldstein and Lewenstein [5] in 2014 is a good example of a substantial recent advance.

The oldest work that was considered for the purposes of this article dates back to 1997 but there are works predating it, too. The model presented by Lopresti and Tomkins [9] is very intuitive and leads to elegant solutions that, in the author’s opinion, match very closely how humans would naturally approach similarity of sequences. Interestingly, the paper is often cited in later works but preferably so in “related work” sections. The author did not find any work that built directly upon the foundation built by Lopresti and Tomkins. One possible reason for this might be that while simple and elegant, the polynomial algorithms still have quite steep curves and might not be able to compete with the greedy heuristic – even more so after introduction of the improvement by Goldstein and Lewenstein [5] –

especially for long strings such as DNA which seems to be where most of the research effort originates from.

Bibliography

- [1] Hsing-Yen Ann et al. “Efficient algorithms for the block edit problems”. In: *Information and Computation* 208.3 (2010), pp. 221–229. DOI: <http://dx.doi.org/10.1016/j.ic.2009.12.001>.
- [2] Laurent Bulteau et al. “Algorithms in Bioinformatics: 13th International Workshop, WABI 2013, Sophia Antipolis, France, September 2–4, 2013. Proceedings”. In: ed. by Aaron Darling and Jens Stoye. Berlin, Heidelberg: Springer, 2013. Chap. A Fixed-Parameter Algorithm for Minimum Common String Partition with Few Duplications, pp. 244–258. ISBN: 978-3-642-40453-5. DOI: 10.1007/978-3-642-40453-5_19.
- [3] Marek Chrobak, Petr Kolman, and Jiří Sgall. “The Greedy Algorithm for the Minimum Common String Partition Problem”. In: *ACM Trans. Algorithms* 1.2 (2005), pp. 350–366. DOI: 10.1145/1103963.1103971.
- [4] Peter Damaschke. “Algorithms in Bioinformatics: 8th International Workshop, WABI 2008, Karlsruhe, Germany, September 15–19, 2008. Proceedings”. In: ed. by Keith A. Crandall and Jens Lagergren. Berlin, Heidelberg: Springer, 2008. Chap. Minimum Common String Partition Parameterized, pp. 87–98. ISBN: 978-3-540-87361-7. DOI: 10.1007/978-3-540-87361-7_8.
- [5] Isaac Goldstein and Moshe Lewenstein. “Quick greedy computation for minimum common string partition”. In: *Theoretical Computer Science* 542 (2014), pp. 98–107. DOI: 10.1016/j.tcs.2014.05.006.
- [6] Michael Gribskov and John Devereux. *Sequence Analysis Primer*. New York: Springer, 1991.
- [7] Haim Kaplan and Nira Shafir. “The greedy algorithm for edit distance with moves”. In: *Information Processing Letters* 97.1 (2006), pp. 23–27. DOI: <http://dx.doi.org/10.1016/j.ipl.2005.08.010>.
- [8] Petr Kolman and Tomasz Waleń. “Approximating reversal distance for strings with bounded number of duplicates”. In: *Discrete Applied Mathematics* 155.3 (2007), pp. 327–336. ISSN: 0166-218X. DOI: 10.1016/j.dam.2006.05.011.
- [9] Daniel Lopresti and Andrew Tomkins. “Block Edit Models for Approximate String Matching”. In: *Theor. Comput. Sci.* 181.1 (1997), pp. 159–179. DOI: 10.1016/S0304-3975(96)00268-X.
- [10] Gonzalo Navarro. “A Guided Tour to Approximate String Matching”. In: *ACM Comput. Surv.* 33.1 (2001), pp. 31–88. DOI: 10.1145/375360.375365.
- [11] Desmond Schmidt. “Merging Multi-Version Texts: a Generic Solution to the Overlap Problem”. In: *Proceedings of Balisage: The Markup Conference 2009*. Vol. 3. Balisage Series on Markup Technologies. Montréal, Canada, 2009. DOI: doi:10.4242/BalisageVol3.Schmidt01.
- [12] Dana Shapira and James A. Storer. “Combinatorial Pattern Matching: 13th Annual Symposium, CPM 2002 Fukuoka, Japan, July 3–5, 2002 Proceedings”. In: ed. by Alberto Apostolico and Masayuki Takeda. Berlin, Heidelberg: Springer, 2002. Chap. Edit Distance with Move Operations, pp. 85–98. ISBN: 978-3-540-45452-6. DOI: 10.1007/3-540-45452-7_9.

- [13] Walter F Tichy. “The string-to-string correction problem with block moves”. In: *ACM Transactions on Computer Systems* 2.4 (1984), pp. 309–321.
- [14] Esko Ukkonen. “International Conference on Foundations of Computation Theory Algorithms for approximate string matching”. In: *Information and Control* 64.1 (1985), pp. 100–118. DOI: 10.1016/S0019-9958(85)80046-2.

6 Latent Dirichlet Allocation

Florian Becker

Abstract

Due to the vast amounts of texts which are produced and digitized on a daily basis, it becomes more and more difficult to find what we are looking for. Topic models can be used as a tool to discover *topics* in large collections of texts. Each text is considered to be a distribution over topics and topics are, in turn, distributions over words. Topic model algorithms receive unlabeled texts as input and produce the topic distribution as output. Latent Dirichlet Allocation is a generative probabilistic topic model. The basic idea is that documents are mixtures of latent topics distributed according to a Dirichlet prior.

6.1 Introduction

When seeking digitized and hyper-linked information, we use searching tools that weigh each result according to some centrality measure. Such techniques are often sufficient, provided that we only want to find some result and do not care about the underlying topic structure which might give us more insight into a document or even a collection of many documents. Topic models enable us to not only search documents by the themes that they contain, but also give us a tool with which we can automatically organize and structure text corpora. It is assumed that a document exhibits multiple topics and each is a distribution over words. Consider, for instance, a corpus with texts about digital humanities. A topic model algorithm might output that the text corpus contains the topics *Computer Science*, *Linguistics* and *Philosophy*. But since every topic is a distribution over words, the algorithm will not just output '*Computer Science*', but will rather give a distribution over words, which we then can identify as the topic '*Computer Science*': e.g. *algorithm*, *program*, *complexity*, *machine*, *Turing*, *artificial intelligence*, etc.

Topic Models can also be thought of as a way of clustering similar documents. Scientific papers, books, tweets, blogs etc. can be organized according to the themes they feature. With topic models organizing a large corpus becomes feasible.

Latent Dirichlet Allocation (LDA) is a generative topic model. In topic models the topic and word distribution of a document can be explained by a generative model. In machine learning generative models are used to *simulate* data points. Given some data (e.g. a text corpus) in a generative model one will assume that there exists some process in the background which has generated the data. In our topic model context, this means that the process first decided what to pick as a topic and then decided which word to pick from that topic. The generative process has certain *parameters* and it is assumed that there must be a certain configuration that is most likely to have generated the observation. The goal is then to infer those hidden parameters. Hence, inference can be seen as reversing the generative process. That means that we want to uncover the most likely configuration that has produced a document. In the case of Latent Dirichlet Allocation the hidden variables are the per-document topic distribution, the per-document per-word topic assignments and the topics themselves.

Generally in *discriminative* machine learning data-driven algorithms are building a model. This process is called *training* or *learning* and refers to the fact that the model adapts iteratively to the training data, such that eventually the model can explain the target variables. *Explaining* the data in a supervised model means, that the model delivers a label

to a query. In unsupervised learning, a model represents the structure of unlabeled data. In a generative (probabilistic) model however, we understand something else under the term *model*. A *model* in this context means something that actively models the data/observations. Section 6.4 and 6.5 discuss the differences between discriminative and generative models in more detail.

Latent Dirichlet Allocation was introduced by David Blei, Andrew Ng, and Michael Jordan in 2003 [2]. Since then it has been applied in various different domains to many different sorts of texts. It is not restricted to operate on ‘classical’ documents. It was applied, for instance, in the domain of public health, to analyze tweets with LDA in order to track illnesses over time [8]. It was also just recently applied in bioinformatics for annotating genomes with a feature term that describes a feature with LDA [9].

As the intersection of computing and humanities grows, topic models will become more important. Humanities profit from algorithmic methodologies applied on large text corpora for obvious reasons; manually structuring those is mostly not an option.

LDA can also be seen as a way of reducing the dimensionality of a corpus, since the topic distribution is a low dimensional representation of the documents [7]. Thus, one may gain insight about the semantic levels when examining the low-dimensional structure of a text corpus.

After a section about preliminaries and notation this report will introduce plate notation in section 6.3 and the generative model in section 6.5. Section 6.4 will discuss LDA in terms of machine learning and will therefore also introduce some terminology. In section 6.6 plate notation will be used in order to introduce a concise way to formulate Latent Dirichlet Allocation as a statistical model. Afterwards, the problem of Bayesian inference is discussed and a solution, the Gibbs Sampling algorithm, is presented.

6.2 Preliminaries and Notation

Throughout this report the notation of the original paper by Blei et al. will be used and is summarized in the following table.

■ **Table 1** Notation, as used in [2].

symbol	description
α	parameter of the Dirichlet prior on the per-document topic distributions
β	parameter of the Dirichlet prior on the per-topic word distribution
θ_i	topic distribution for document i
φ_k	word distribution for topic k
z_{ij}	topic for the j -th word in document i , and
w_{ij}	specific word
\mathbf{w}	a sequence of words (a document)
\mathcal{D}	a corpus of M documents; $\mathcal{D} = \{w_1, \dots, w_M\}$

In order to understand in what way topics are assumed to be distributed the multinomial and Dirichlet distribution including important properties will be summarized briefly.

6.2.1 Multinomial Distribution

The binomial distribution is a discrete distribution over events with two outcomes. It gives the probability of obtaining k successes out of n trials, where each success has a probability

of p . The probability mass function (PMF) is given by (Equation 1).

$$\Pr(X = k) = \binom{n}{k} p^k (1 - p)^{n-k} \quad (1)$$

The binomial distribution is a special case of the multinomial distribution. The multinomial distribution models events which have k outcomes, where each has a fixed probability. The probability mass function is given by

$$f(x_1, \dots, x_k; n, p_1, \dots, p_k) = \frac{n!}{x_1! \dots x_k!} p_1^{x_1} \dots p_k^{x_k} \quad (2)$$

6.2.2 Dirichlet Distribution

The Dirichlet distribution $\text{Dir}(\alpha)$ is a probability distribution parameterized by a positive real valued vector $\alpha = (\alpha_1, \dots, \alpha_K)$. The probability density function is given by:

$$f(x_1, \dots, x_K; \alpha_1, \dots, \alpha_K) = \frac{\Gamma\left(\sum_{i=1}^K \alpha_i\right)}{\prod_{i=1}^K \Gamma(\alpha_i)} \prod_{i=1}^K x_i^{\alpha_i-1} \quad (3)$$

where Γ is the Gamma function.

The Dirichlet distribution is often also expressed with the inverse of the Beta function as a normalizing constant. $B(\alpha)$ is constant as it only depends on the fixed parameter α .

$$f(x_1, \dots, x_K; \alpha_1, \dots, \alpha_K) = \frac{1}{B(\alpha)} \prod_{i=1}^K x_i^{\alpha_i-1} \quad (4)$$

where

$$B(\alpha) = \frac{\prod_{i=1}^K \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^K \alpha_i\right)} \quad (5)$$

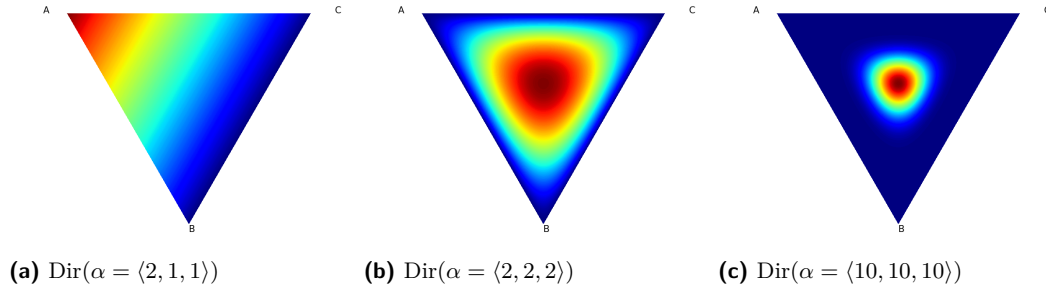
The expected value of the Dirichlet distribution is simply computed by:

$$E[X_i] = \frac{\alpha_i}{\alpha_0}, \quad \alpha_0 = \sum_{i=1}^K \alpha_i. \quad (6)$$

The support of the Dirichlet distribution of order K is the $(K - 1)$ -simplex. For $K = 3$ the 2-simplex is given by a regular triangle, where each corner is an event. The vertices are $(0, 0, 1)$, $(0, 1, 0)$ and $(1, 0, 0)$. A point $\vec{p} = (p_1, p_2, p_3)$ on this probability simplex corresponds to a certain event.

6.3 Plate Notation

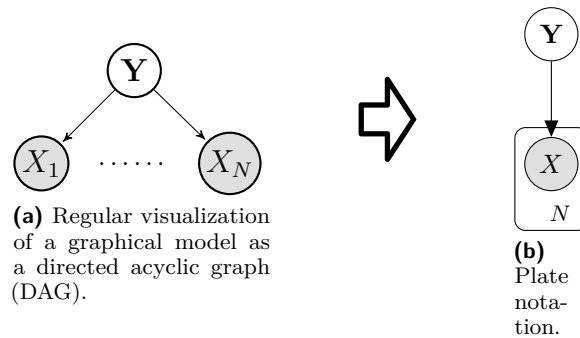
Generally, Bayesian networks (or probabilistic directed acyclic graphical models) represent conditional dependencies with a directed acyclic graph (DAG). A directed edge from a



■ **Figure 1** Dirichlet distributions in the two-dimensional simplex with different α . The corners A , B , C correspond to topics. Regions of higher probability are shown by a heat map, i.e. (dark) red corresponds to the most probable sample.

random variable Y to a random variable X means that Y *causes* X .

In a probabilistic graphical model conditional dependencies between various random variables are often depicted in plate notation. Plate notation is a concise method for visualizing the factorization of the joint probability. Typically, shaded nodes stand for *observed* random variables. The plates represent the repeated structure.



■ **Figure 2** Plate notation as a way to summarize conditional dependencies. Shaded nodes represent *observed* variables. Here, the X_i are conditionally dependent on Y . In plate notation all the X_i are summarized as one plate.

The joint probability of the probabilistic graphical model depicted in (Figure 2) is

$$\begin{aligned} \Pr[X_1, \dots, X_n] &= \prod_{i=1}^n \Pr[X_i | \text{parent}(X_i)] \\ &= \prod_{i=1}^n \Pr[X_i | Y] \end{aligned} \quad (7)$$

The plate notation will be used in the next section to have a concise representation of Latent Dirichlet Allocation and especially the conditional dependencies.

6.4 Machine Learning and Topic Models

In this section Latent Dirichlet Allocation will be discussed in terms of machine learning. Therefore, the distinction between supervised and unsupervised as well as the distinction between discriminative and generative model will be clarified.

Machine learning algorithms or problems are usually classified into two broad categories: supervised and unsupervised learning¹. In supervised learning the task is to find a hypothesis or model from labeled data such that the model can correctly classify the data. Given a set of *training examples* $\{x_{1:N}, y_{1:N}\}$, where $x_i \in X$ is an instance or *feature vector* and $y_i \in Y$ is its corresponding label a supervised machine learning algorithm must find a hypothesis $h : X \rightarrow Y$ such that $\forall i : h(x_i) = y_i$. In contrast to that, in an unsupervised machine learning problem there are no labels. Clustering, for instance, falls into the domain of unsupervised learning. Given data vectors $\{x_{1:N}\}$ the task is to label them. The underlying premise of clustering algorithms (such as *K-Means* for example) is that data that belongs to the same cluster is more *similar* to each other than to data from other clusters. Given an input space X and two clusters $V = \{v_{1:N} | v \in X\}$ and $W = \{w_{1:M} | w \in X\}$ and some notion of similarity $s : X \times X \rightarrow \mathbb{R}$, it must hold that $\forall v_1, v_2 \in V$ and $\forall w \in W : s(v_1, v_2) > s(v_1, w)$. The objective function is to maximize the similarities within one cluster and minimize the similarities between clusters. Latent Dirichlet Allocation is an unsupervised model, which can be seen as a clustering algorithm. The input space X consists of documents, which exhibit K topics. Unlike *K-Means*, in LDA a document can be part of more than one cluster (see also Figure 6).

6.4.1 Discriminative Models

Discriminative algorithms, as opposed to generative ones, directly optimize a model for a classification task [6]. Given an observed variable x and unobserved variable y , discriminative models will give the conditional distribution $P(y|x)$. *K-Means* for instance is a similarity-bases, unsupervised, discriminative approach [11]. Examples of discriminative models include *linear regression*, *support vector machines* and *nerual networks*. However, LDA is a generative model. The assumption is that a corpus is generated by a process, which will be discussed in the next section.

6.5 Generative Process

Generative models generate data values according to a probability distribution. In machine learning generative models are used to estimate the joint probability distribution $Pr(X, Y)$ of observed data X and labels Y . In the context of probabilistic topic models, Y corresponds to the latent variables (i.e. to the unknown parameters) and X to the (observable) words in a corpus. Latent Dirichlet Allocation assumes the following generative process for a corpus:

¹ Reinforcement learning does not fall into this categorization. Since it is not of importance in this context, it won't be considered here.

Input : Number of words N , Number of topics K
 Dirichlet parameters $\alpha \in \mathbb{R}_+^K, \beta \in \mathbb{R}_+^N$
Output : D documents (bag-of-words) with N words each

1. Choose $\theta_i \sim \text{Dir}(\alpha)$,
2. Choose $\varphi_k \sim \text{Dir}(\beta)$

for each word position i, j **do**

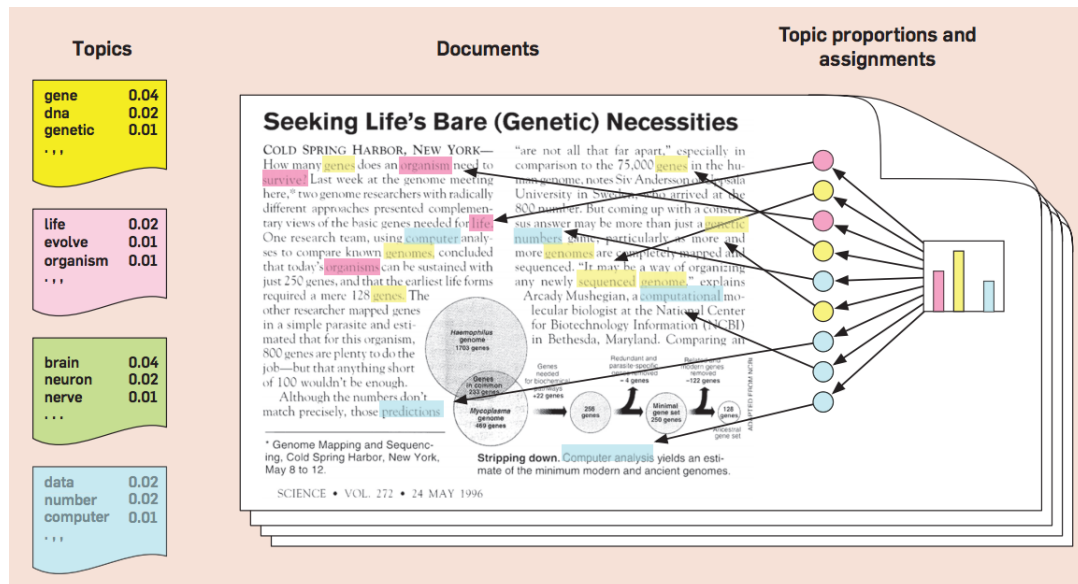
- (a) Choose a topic $z_{i,j} \sim \text{Multinomial}(\theta_i)$.
- (b) Choose a word $w_{i,j} \sim \text{Multinomial}(\varphi_{z_{i,j}})$

end

Algorithm 1 : Generative Model

The generative model contains the main ideas of LDA. Data is treated as observations coming from a process which samples words from a (hidden) distribution over topics. Every word is generated independently from any other word. The generative process does not take word order into account. Hence, the generative model produces a bag-of-words, where word order does not play a role.

If three topics are chosen, e.g. *Linguistics*, *Philosophy* and *Computer Science*, and if equal probability is given to all of them, then the generative process might produce bag-of-words resembling a text from *Digital Humanities*. By putting more weight on *Linguistics* and *Computer Science* the result would resemble documents from the domain of *Computer Linguistics*. The generative process of LDA is depicted as a Bayesian probabilistic model in Figure 4.

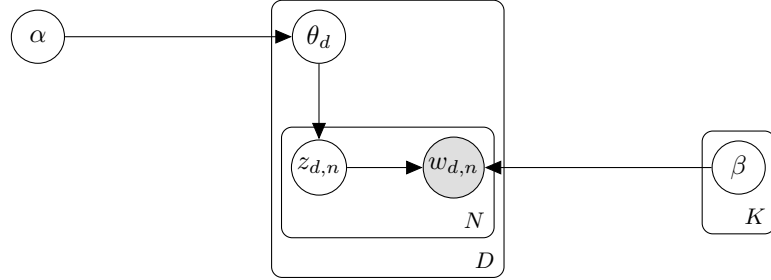


■ **Figure 3** The generative process. Every word originates from a topic coming from a distribution over topics. The topics and their word distributions are depicted on the far left. E.g. the *yellow* topic is made up of the word *gene* by 4%. Taken from [1]

In Figure 3 the generative model is illustrated. The topic proportions are plotted as a

bar chart, the colored coins are the topic assignments for each word.

6.6 LDA as a graphical model



■ **Figure 4** Plate notation for Latent Dirichlet Allocation corresponding to the generative process. Every vertex depicts a random variable. The shaded node $w_{d,n}$ stands for the observed words. Everything else is not observed and must be inferred.

From the Bayesian probabilistic model one can derive the probability of a corpus $\mathcal{D} = \{\mathbf{w}_1, \dots, \mathbf{w}_n\}$. First, the joint distribution of a mixtures of topics θ , N topics \mathbf{z} and N words \mathbf{w} can be calculated by considering the conditional structure given in the graphical model:

$$p(\theta, \mathbf{z}, \mathbf{w} | \alpha, \beta) = p(\theta | \alpha) \prod_{n=1}^N p(z_n | \theta) p(w_n | z_n, \beta) \quad (8)$$

In order to arrive at the marginal distribution of one single document \mathbf{w} , one must integrate over the (continuous) random variable θ and sum over the topic variable z .

$$p(\mathbf{w} | \alpha, \beta) = \int p(\theta | \alpha) \left(\prod_{n=1}^N \sum_{z_n} p(z_n | \theta) p(w_n | z_n, \beta) \right) d\theta \quad (9)$$

The probability of the corpus $p(\mathcal{D} | \alpha, \beta)$ is now obtained by multiplying all the marginal probabilities of the M documents:

$$p(\mathcal{D} | \alpha, \beta) = \prod_{d=1}^M \int p(\theta_d | \alpha) \left(\prod_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn} | \theta_d) p(w_n | z_{dn}, \beta) \right) d\theta_d \quad (10)$$

6.7 (Bayesian) Inference

In Bayesian statistics, one is interested in computing the degree of belief $p(h|D)$ of a hypothesis $h \in \mathcal{H}$ given data D , where \mathcal{H} is the space of all hypotheses. Bayes theorem states:

$$p(h|D) = \frac{p(h)p(D|h)}{p(D)} \quad (11)$$

$p(h|D)$ is also called the posterior.

Inference means computing the *posterior* and corresponds to ‘reversing’ the generative model. In other words, inference is about fitting a generative model (or rather the hidden

parameters), such that it explains the observed data. In Bayesian statistics, maximum likelihood estimation is used in order to estimate the parameters of a model given data. Let τ be the parameter of a statistical model and let $\mathbf{x} = x_1, \dots, x_n$ be the observations, then the maximum likelihood estimator is defined to be [3]:

$$\arg \max_{\tau} \mathcal{L}(\tau; \mathbf{x}) = \arg \max_{\tau} P(\mathbf{x}|\tau) \quad (12)$$

Many methods solving the problem of maximum a posteriori estimation have been proposed. The original paper on LDA [2] suggested a Variational Bayesian method, which directly maximizes $P(\mathbf{w}|\varphi, \theta)$, which corresponds to equation 12 and can be understood as an extension of the expectation-maximization algorithm [4]. Gibbs Sampling as a way to compute the posterior was proposed by Steyvers and Griffiths [10]. They considered the posterior distribution over the assignments of words $w_{1:D}$ to topics $z_{1:D}$:

$$p(\beta_{1:K}, \theta_{1:D}, z_{1:D} | w_{1:D}) = \frac{p(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D})}{p(w_{1:D})} \quad (13)$$

The posterior is intractable to compute. More precisely, the denominator of (Equation 13), the marginal probability, is not feasible to compute. To compute $p(w_{1:D})$ one would have to sum up the joint distribution over all possible instances of the hidden topic structure.

6.7.1 Gibbs Sampling

Gibbs Sampling, named after the physicist Josiah Willard Gibbs, is a Markov Chain Monte Carlo Algorithm (MCMC) and can be used to approximate the marginal and posterior distribution by constructing a Markov chain which converges to the target distribution. A Markov chain represents a random process, where going from one state to another is determined by a transition matrix P . An entry $p_{i,j}$ corresponds to the probability of going from state i to state j . The goal of Gibbs Sampling is to integrate out the per-document topic proportions θ . In order to assign a word to a topic, the Gibbs sampler computes the probability of a topic $z_{d,n}$ if it is assigned to a word $w_{d,n}$, where all other word-to-topic assignments are given or fixed. $z_{-d,n}$ is defined to be the notation for all topic assignments except for for $z_{d,n}$

Formally:

$$p(z_{d,n} = k | z_{-d,n}, w, \alpha, \beta) = \frac{p(z_{d,n} = k, z_{-d,n} | w, \alpha, \beta)}{p(z_{-d,n} | w, \alpha, \beta)} \quad (14)$$

It was shown [5] that Equation 14 can be calculated by:

$$p(z_{d,n} = k | z_{-d,n}, w, \alpha, \beta) \propto \frac{C_{n,k}^{WT} + \beta}{\sum_{n=1}^W C_{n,k}^{WT} + W\beta} \cdot \frac{C_{d,k}^{DT} + \alpha}{\sum_{t=1}^T C_{d,t}^{DT} + T\alpha} \quad (15)$$

where T is the number of topics, and W the number of unique words. $C_{n,k}^{WT}$ and $C_{d,k}^{DT}$ are matrices. $C_{n,k}^{WT}$ is the number of times topic k was assigned to word n . On the other hand $C_{d,k}^{DT}$ is the number of times topic k was assigned to words in document d . The first step of the Gibbs Sampling algorithm is to randomly assign every word of all documents to a topic. By this the count matrices $C_{n,k}^{WT}$ and $C_{d,k}^{DT}$ are filled with values. Gibbs Sampling will

now sample topic assignments according to Equation 15. The first term of the product in Equation 15 also corresponds to an estimate of φ_i , and the second term to an estimate of θ_j

$$\hat{\varphi}_i = \frac{C_{i,j}^{WT} + \beta}{\sum_{k=1}^W C_{k,j}^{WT} + W\beta} \quad (16)$$

$$\hat{\theta}_j = \frac{C_{d,j}^{DT} + \alpha}{\sum_{t=1}^T C_{d,t}^{DT} + T\alpha} \quad (17)$$

► **Example 1.** The following step-by-step example illustrates how Gibbs Sampling can be used to assign a topic to a word. Consider the sentence (which represents one document out of a corpus):

Text mining algorithms can be used to find structure in text corpora like Plato's dialogues

A priori we choose the number of topics $K = 3$. The first step is to remove stop words and randomly assign each word to a topic (Table 2). This is done to all documents.

■ **Table 2** random initialization of topic assignments to words.

1	3	2	1	2	1	2
text	mining	algorithms	structure	corpora	Plato	dialogues

The counts for all documents is shown in Table 3.

■ **Table 3** word counts for every topic

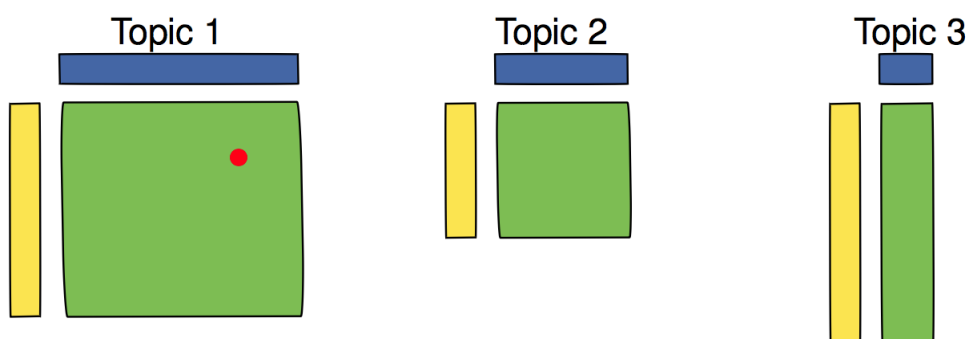
	1	2	3
text	65	54	59
mining	21	4	12
algorithms	100	74	122
structure	20	12	14
corpora	5	2	12
Plato	35	33	42
dialogues	24	27	31

The goal is now to resample the word *algorithm*.

■ **Table 4** We first choose *algorithm* to be resampled.

1	3	???	1	2	1	2
text	mining	algorithms	structure	corpora	Plato	dialogues

The assignment of the word *algorithm* is determined by sampling according to the topic distribution in the document and the word distribution over the corpus. In Figure 5 the sampling is depicted. The blue bars correspond to the topic distribution, the yellow bars to the word distribution over topics. Accordingly, it is most likely that *algorithm* is assigned to topic 1, since the green area for topic 1 is the largest. After assigning the word, the word counts for each topic get updated and the first iteration of the Gibbs sampling algorithm is completed. This process is repeated several times, until the assignments do not change anymore.



■ **Figure 5** Sampling according to the green area. Blue bars represent the topic distribution, the yellow bars to the word distribution over topics.

6.8 Experiments and Applications

To fully grasp what LDA will produce as output, this section will provide an example with *real world* data.

Gensim² is a tool for unsupervised semantic modeling. The idea is to define three topics by using Wikipedia articles. Then, all those Wikipedia articles are put into one corpus to see whether LDA can reproduce those three original topics. Basically this is a clustering task with a predefined number of clusters K . Similar to *K-Means* LDA will use the training set (the predefined Wikipedia articles) and will find the clusters that maximize the model parameters.

With Gensim the model can be trained very easily³:

■ Listing 1 Training the LDA model

```
K = 3
```

² <https://radimrehurek.com/gensim/index.html>

³ the complete ipython notebook can be found here: https://github.com/flobeck/algorithm-seminar/blob/master/ipython-notebook/lda_mixture.ipynb

```
lda = ldamodel.LdaModel(CORPUS, id2word=dictionary, num_topics=K,
update_every=1, chunksize=1, passes=350)
```

Gensim uses the Gibbs sampling procedure that was discussed in section 6.7.1. CORPUS contains the following preprocessed Wikipedia articles. The preprocessing steps comprise removing stop words (i.e. most common words) and transforming the documents to a bag-of-words.

- *Computer science* Algorithm, Computation, Computer Programming, Programming Language, Computational complexity theory, Computability theory, Artificial Intelligence
- *Philosophy* Epistemology, Metaphysics, Continental philosophy, Ancient Greek, Ethics, Aesthetics, Art, Phenomenology (philosophy), Pythagoras, Plato
- *Linguistics* Language, Semantics, Syntax, Phonology, Grammar, Phonetics, Pragmatics, Corpus linguistics, Linguistic prescription, Linguistic description

The output of a call of LDA with $K=3$ is summarized in table 5.

Given a set corpus of Wikipedia articles, LDA finds three topics which resemble the topics *computer science*, *philosophy* and *linguistics*.

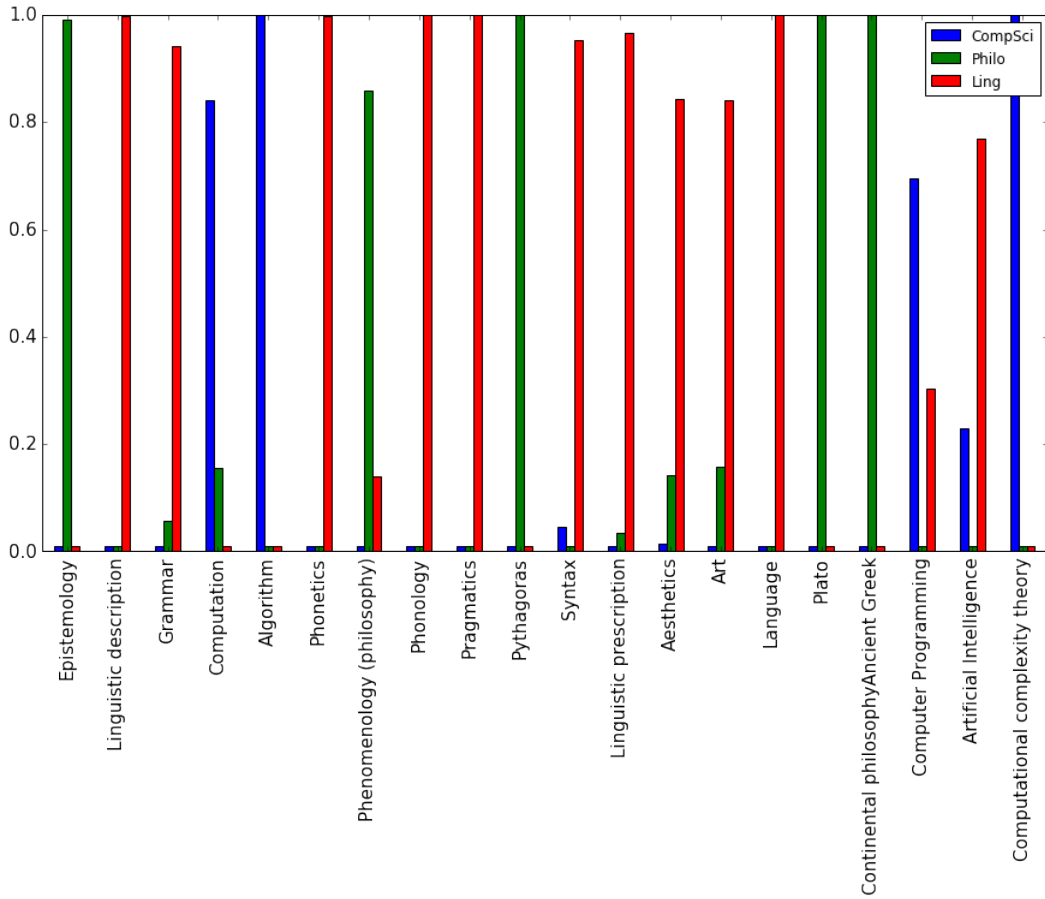
Furthermore, it is possible to query where on the topic simplex an unseen document is located:

■ Listing 2 Query for a document

```
noam = wikipedia.page("Noam Chomsky").content
bow_vector = dictionary.doc2bow(tokenize(noam))
print lda[bow_vector]

>> [(1, 0.032599745516170064),
(2, 0.30504589551791855),
(3, 0.66235435896591133)]
```

In this case (Listing 2), the Wikipedia article on the linguist *Noam Chomsky* is made up of 66% Topic 3 (*Linguistics*), 30% Topic 2 (*Computer Science*) and 3% Topic 1 (*Philosophy*). Blei et. al showed that one can train a model for document classification with Latent Dirichlet Allocation [2]. Instead of using individual words as features, one can use the topic proportions for each document. A document can then be classified by using LDA to extract the topic proportion feature vectors of the training set and then a finding decision boundary with a Support Vector Machine (SVM) or some other linear classifier.



■ **Figure 6** Topic proportions for some of the Wikipedia articles that were used to train the model. The Wikipedia article on ‘Pythagoras’, for instance, is classified as a document in the category ‘philosophy’ with very high confidence. Note that albeit the confidence is very high it is not 100%. Due to the model every document exhibits all the K topics, but to a very different degree.

■ **Table 5** Each of the three topics is a distribution over the words. E.g. topic 1 resembles *computer science*. The table only shows the most relevant words for each topic.

TOPIC	REL. FREQUENCIES	WORD
1	0.010	programming
	0.009	turing
	0.008	problem
	0.007	machine
	0.006	problems
	0.006	algorithms
	0.006	algorithm
	0.006	set
	0.006	complexity
	0.005	time
2	0.008	plato
	0.006	plato's
	0.006	knowledge
	0.005	pythagoras
	0.005	philosophy
	0.004	phenomenology
	0.004	greek
	0.004	according
	0.004	socrates
	0.003	theory
3	0.019	language
	0.009	languages
	0.007	art
	0.005	meaning
	0.005	linguistic
	0.005	human
	0.004	ethics
	0.004	speech
	0.004	study
	0.004	grammar

6.9 Conclusion

Topic models enable scholars to find the hidden topic structure in large text corpora. The hidden structure must be uncovered in order to gain information about the low-dimensional structure behind a text corpus. A topic modeling algorithm takes a corpus as an input and will output the topics running through that corpus. The generative process is the essential idea behind Latent Dirichlet Allocation. Words come from a distribution over topics, which are again distribution over words. Reversing the generative model by applying Gibbs sampling yields a solution of the inference problem, which can not be solved analytically.

References

- 1 David M Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012.
- 2 David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.
- 3 Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.
- 4 Charles W Fox and Stephen J Roberts. A tutorial on variational bayesian inference. *Artificial intelligence review*, 38(2):85–95, 2012.
- 5 Thomas L Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National academy of Sciences*, 101(suppl 1):5228–5235, 2004.
- 6 Tony Jebara. *Discriminative, generative and imitative learning*. PhD thesis, Massachusetts Institute of Technology, 2001.
- 7 Tomonari Masada, Senya Kiyasu, and Sueharu Miyahara. Comparing lda with plsi as a dimensionality reduction method in document clustering. In *Large-Scale Knowledge Resources. Construction and Application*, pages 13–26. Springer, 2008.
- 8 Michael J Paul and Mark Dredze. You are what you tweet: Analyzing twitter for public health. *ICWSM*, 20:265–272, 2011.
- 9 Pietro Pinoli, Davide Chicco, and Marco Masseroli. Latent dirichlet allocation based on gibbs sampling for gene function prediction. In *Computational Intelligence in Bioinformatics and Computational Biology, 2014 IEEE Conference on*, pages 1–8. IEEE, 2014.
- 10 Mark Steyvers and Tom Griffiths. Probabilistic topic models. *Handbook of latent semantic analysis*, 427(7):424–440, 2007.
- 11 Shi Zhong and Joydeep Ghosh. Generative model-based document clustering: a comparative study. *Knowledge and Information Systems*, 8(3):374–384, 2005.

7 Syntax Trees and Syntax Tree Drawing

Jan Keim

Abstract

In the field of analyzing grammatical structures, syntax trees are an essential tool for visualization. With the help of syntax trees different properties like commonalities and differences of sentences, movement of words between sentences or ambiguities can be explored. Drawing syntax trees with pleasing aesthetic properties is therefore a necessity. Algorithms that fulfill these aesthetic properties algorithm like the Buchheim-Walker algorithm are required to allow for computer-aided or automatic drawing of syntax trees. In this work, the historical development along with principles of this algorithm is outlined. There are many tools that all provide different features and approaches in drawing these trees. The outputs of some of them are compared. Especially the tool TreeForm is highlighted as an easy and proper solution for drawing syntax trees.

7.1 Introduction

People that study grammatical structures, like linguists, rely on a proper visual representation for these structures. Syntax trees act as such a required visual model for grammar and are therefore a necessary tool. Hence, section 7.2 gives a basic introduction into syntax trees to get to know syntax trees and the application of them.

Although syntax trees can be drawn by hand, this solution is, for various reasons, unsatisfying. Automatic drawing of syntax trees with the help of algorithms can solve many issues. Section 7.3 looks into drawing conventions and properties of tree drawing algorithms before pointing out which properties are needed for drawing syntax trees. Later the Buchheim-Walker algorithm that is widely used for drawing syntax trees is presented along its historical development.

Finally, section 7.4 looks into different drawing tools with focus on the tool TreeForm as an example for an easy-to-use tool that produces good syntax trees.

7.2 Syntax Trees

People who want to explore and explain grammar need a model to represent grammatical structures and other parts of interest in grammar visually. One possible solution is to use the so called *bracket notation*, where words within a sentence are annotated and surrounded with brackets to limit the scope of the annotated grammatical structure. An example of bracket notation for the question “What did John eat?” with the corresponding answer “John ate bread.” can be found in example 1.

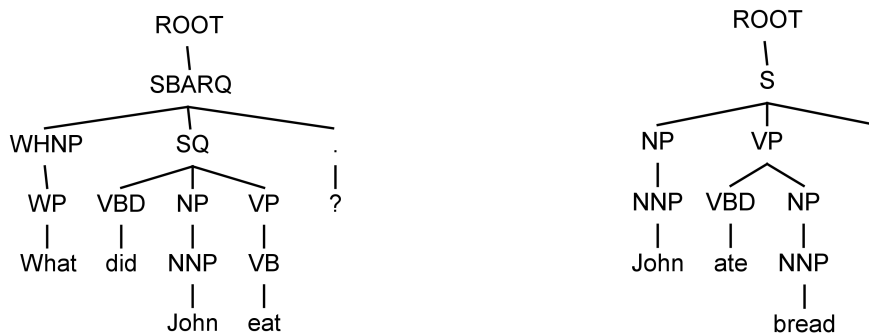
► **Example 1.**

```
((SBARQ (WHNP (WP What)) (SQ (VBD did) (NP (NNP John)) (VP (VB eat))) (. ?)))
((S (NP (NNP John)) (VP (VBD ate) (NP (NN bread)))) (. .)))
```

Bracket notation can carry all information needed, but is a rather poor visual representation as it is not particularly well readable by the viewer. It is especially hard for the user to see where the scopes of structures exactly are. This gets even harder for phrases like noun phrases, commonly abbreviated *NP*, or verb phrases, commonly abbreviated *VP*, because

those phrases usually aggregate multiple structures, which results in an even more confusing representation.

A solution for this problem is the usage of Syntax Trees. Syntax trees are made to give the viewer an excellent visual representation for grammatical structures. Examples for syntax trees can be found in Figure 1. The sentences are the same as in example 1 for bracket notation and the exact same grammatical structure is represented. Instead of having confusing brackets the tree visualizes neatly how the grammar in the sentence is structured. The depiction of the depth of grammatical structures with the help of levels in syntax trees is especially beneficial. Structures like phrases, that made bracket notation more confusing, are easily visualized with sibling nodes under a common parent. For example in Figure 1b there is a verb phrase (VP) consisting of the verb (VB) “ate” and as the object of the sentence a noun phrase (NP) solely consisting of the noun (NNP) “bread”. Verb phrases generally contain at least a verb and can be extended with further details like adverbs, objects and the like. Similar to a verb phrase, a noun phrase contains at least a noun (NN*) and can be extended by descriptive information like adjectives (JJ), pronouns (PRP) etc. Both are the most important structures in syntax trees as a common sentence has at least a subject, which is represented by a noun phrase, and a verb, which is represented by a verb phrase. There are plenty of labels on clause level, phrase level and word level, but the detailed meaning of each possible label is not important for this write-up. Further, those labels are not standardized, which results in a huge amount of labels. Therefore further labels will not be explained here if the explanation are not needed for understanding ¹.



(a) “What did John eat?”

(b) “John ate bread.”

■ **Figure 1** Example syntax trees

In this write-up the focus lies primarily on the usage of syntax trees in the linguistic field, so mainly syntax of natural language is shown. However, syntax trees are not tied to grammar in natural language but can theoretically be used for every other grammar like the ones for parser and compilers in computer science.

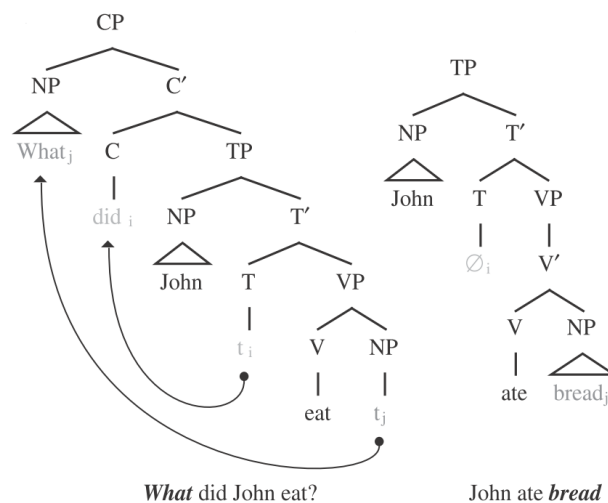
Generating a syntax tree is done with the help of a grammar that is used to parse the sentence. By exchanging the underlying grammar the appearance and focus of a syntax tree can be altered, because the grammar defines how structures look like and what phrases

¹ A popular set of labels with explanations can be found at <https://gist.github.com/nlothian/9240750>

exists, what they can contain etc. An example would be one grammar that treats all verbs the same while another grammar makes a difference in active and passive verbs. Additionally one grammar may put adjectives in a verb phrase together with the verb while another one would not cover this.

Besides just representing syntax, these trees also offer the possibility to point out dependencies between structures of sentences. Once one or multiple syntax trees are drawn one can start to observing the special points of interest like commonalities between different sentences.

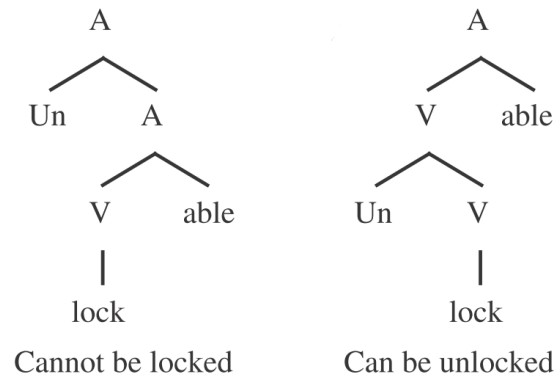
By adding further information and structures into syntax trees, theories can be examined even better. One of these structures are movement lines that can be seen in Figure 2. The sentences there are the same as above from example 1 and figure 1, but the parsing grammar is different. This syntax tree illustrates how a question and its answer are related to each other. This resemblance is visible in the subtree at the node *TP* on the left that is basically the same as the syntax tree on the right. This shows how the question is closely related to the answer. The movement lines support this statement by showing how parts of the sentences were dragged to the beginning to form the question. Here, the past tense auxiliary verb “did” was moved to the front and the question word “what” can be derived from “bread”, the main subject of the answer of the question.



■ **Figure 2** Example for syntax trees with movement lines [6]

Another application for syntax trees is their usage to show differences in ambiguous words and sentences. Figure 3 shows the syntax trees for the ambiguous word “unlockable”. The left meaning “Cannot be locked” can be easily explained with the help of the syntax tree. The verb “lock” is grouped with the word “able”, which builds the adjective “lockable” as main meaning, that then gets negated with the prefix “un”. On the right hand side the meaning “Can be unlocked” can be explained with its syntax tree as easily. Now the verb gets negated at first to form main meaning “unlock” before it then gets transformed into an adjective with the help of the suffix “able”.

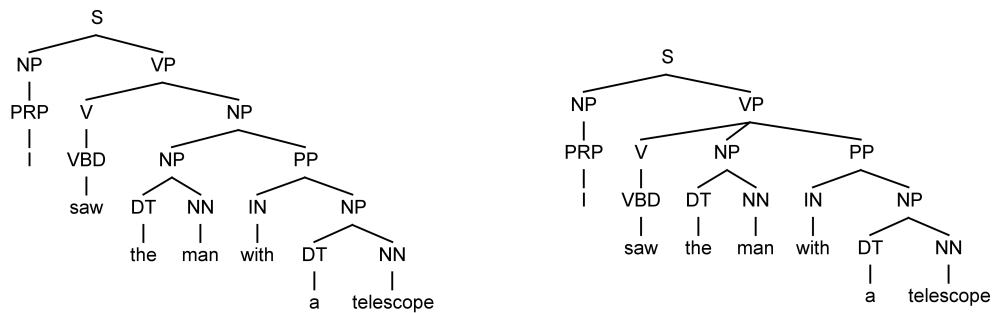
Besides words, sentences can be ambiguous as well and those ambiguities can be explained in a very similar way. Figure 4 shows the two syntax trees for the ambiguous sentence “I saw the man with a telescope.” Here, the meaning is also easily explainable with the help of those syntax trees. In figure 4a there is a verb phrase (*VP*) consisting of a Verb (*V*) and a



■ **Figure 3** Example for syntax trees for the ambiguous word “unlockable” [6]

noun phrase (*NP*). The noun phrase itself consists of the noun phrase “the man” and the prepositional phrase (*PP*) “with a telescope”. It is easily visible that the phrase “the man with a telescope” is here the intended coherent meaning. The syntax tree on the right shows a slightly different structure. The noun phrase from above was split here and the verb phrase now consists of the verb, a noun phrase and a prepositional phrase. This prepositional phrase is not bound to the noun phrase anymore but rather bound to the verb phrase and thus emphasizes the verb. This results in the meaning that a telescope was used to see the man.

To go even further the sentence can be altered with the little phrase “on a hill” to form the sentence: “I saw the man on a hill with a telescope.” This would result in up to 5 different meanings that could be extracted from this sentence. Syntax trees would help again to differentiate most of these meanings.



(a) I saw a man who was with a telescope.

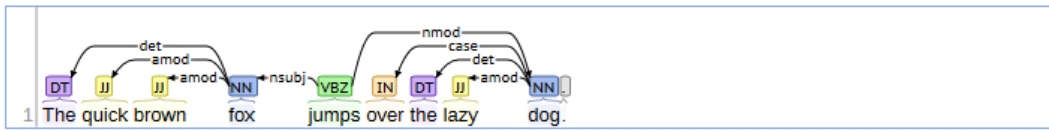
(b) Using a telescope I saw a man.

■ **Figure 4** Syntax trees for the ambiguous sentence “I saw the man with a telescope.”

7.2.1 Dependency Trees

Similar to syntax trees there are so called *Dependency Trees*. They use different grammars to serve a slightly different purpose to show dependencies within one sentence. In contrast to syntax trees, here the depth of the structures is not that important anymore because the focus lies more on the edges instead of the nodes. A basic example can be seen in Figure 5.

The root of dependency trees is usually the verb of the sentence. In contrast to syntax



■ **Figure 5** Example for a dependency tree, created with Stanford’s CoreNLP [8]

trees phrases are distinguished based on their dependencies and task in the sentence. For example in a syntax tree “the quick brown fox” and “the lazy dog” would each be noun phrases and it would be slightly harder to tell which one of them is the actual subject within this sentence. Within the dependency tree there is the edge *nsubj* from the verb to the subject of the sentence, here from “jumps” to “fox”.

Such dependencies are designed to be useful in many situations, especially if you need to process the information of the dependencies within a sentence instead of the actual (grammatical) structures [5][7]. A good example for that is trying to convert a natural language text into code. There, the information about subject, predicate and object(s) of the sentence can be used to transform them into the class or object that calls a function with some parameters.

Both, dependency trees and syntax trees each have their own applications where they are more beneficial to use than the other and the user judges which represents the required information better.

7.3 Tree Drawing Algorithms

There is a wide variety of tree drawing algorithms [1][10]. Each of them has its own characteristics and area of application and not every algorithm for drawing trees is automatically applicable for drawing syntax trees.

Syntax trees have special drawing requirements as they are for instance designed to provide a good visual representation. To figure out which of these algorithms are applicable for syntax trees a short introduction into drawing conventions of tree drawing algorithms is given in section 7.3.1. After this overview a deeper look into the actual needs of syntax trees is taken in section 7.3.2.

7.3.1 Properties

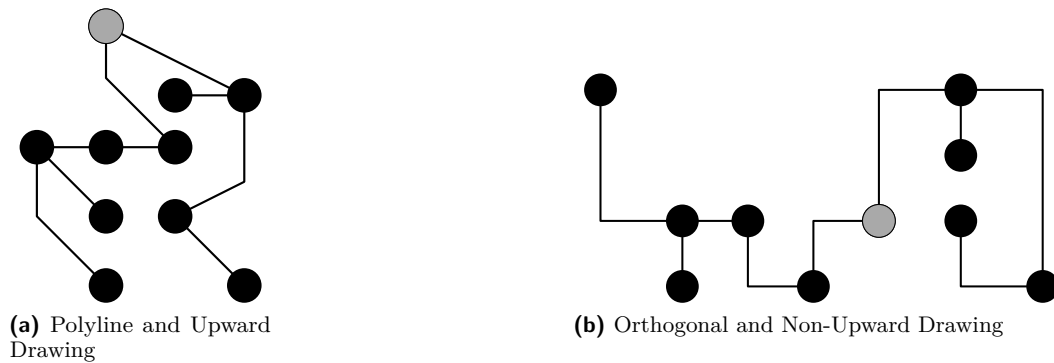
According to Battista et al [1], properties or *drawing conventions* are basic rules that a drawing of a graph must satisfy to be admissible. These properties define some aspects of how the resulting trees will look like.

In a *Polyline Drawing* of a tree, edges are represented as polylines with *bends*. An example for this can be found in Figure 6a.

A similar but more strict version of polyline drawings are *Orthogonal Drawings*, where the edges consist of only vertical and horizontal segments like the ones visible in Figure 6b.

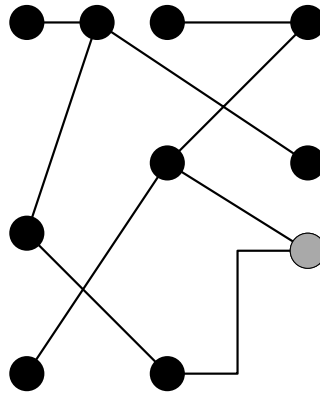
If we look into syntax trees both drawing conventions are not desired as bends can confuse the viewer. Additionally, with these bends the order of nodes cannot be assured which is a rather disadvantageous trait for syntax trees.

The next kind of drawings are *Planar Drawings*, where the lines of the edges must not cross each other. The trees of Figure 6 show examples for this property while the tree in



■ **Figure 6** Trees drawn with different properties [10]

Figure 7 does not fulfill this property. In regard to syntax trees it is essential for the clarity of the visualization that the used algorithm creates a planar drawing.



■ **Figure 7** Tree drawn in a non-planar way [10]

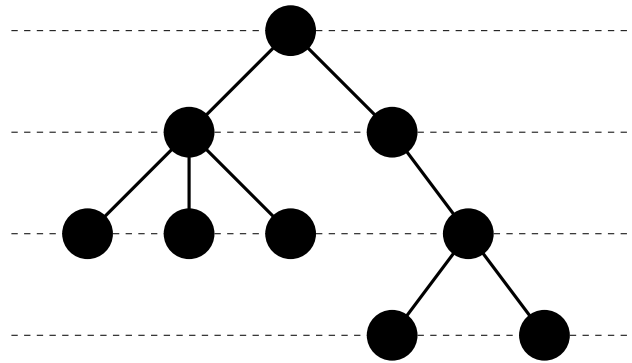
Drawing algorithms can also have a property to create an *Upward Drawing* like the tree in Figure 6a. An upward drawing tree has the characteristic that no child node is drawn above its parent node. In contrast, Figure 6b shows a non-upward drawing, where no limitation regarding the position in reference to the parent node is made.

In Figure 6a some children nodes are drawn on the same level with their parents. *Level-based approaches* are a more strict version of upward drawings that do not allow this. In a level-based approach nodes that have the same distance from the root node are drawn on the same horizontal level. This is illustrated in Figure 8. It is easy to see that such a level-based approach is desired for syntax trees as this means that grammatical structures that have the same depth within the sentence will be drawn on the same level. This also means that it is easily visible where higher-ranking structures are and what they do contain.

7.3.2 Drawing Conventions for Syntax Trees

To understand which drawing conventions match the syntax trees we take a closer look on their definition.

► **Definition 2.** A Syntax Tree is a rooted, ordered tree of unbound degree.



■ **Figure 8** Tree drawn with a level-based approach [1]

The definition says that a syntax tree is rooted meaning that a single vertex of a tree is designated as a root. Further, the definition states that the tree must be ordered, which means the relative position of each node in the tree is predefined. In the case of syntax trees this means that the order of the leaves is defined by the order of the input, thus the leftmost part of the sentence will also be the leftmost part in the syntax tree. The last part states that syntax trees have unbound degree, that is, nodes can have arbitrarily many children.

Thus, definition 2 and other considerations lead to the following drawing conventions [2]:

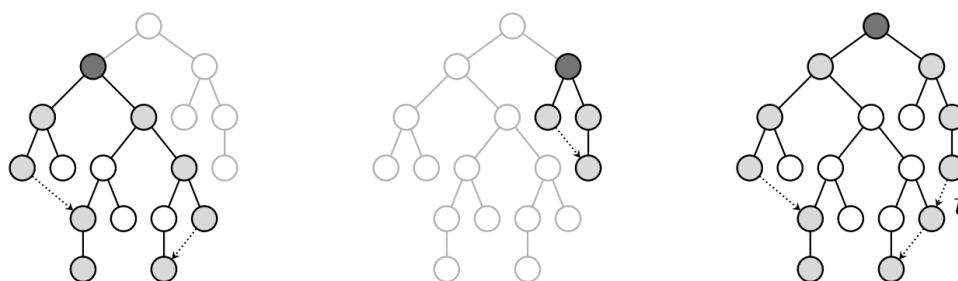
- The first aesthetic property demands that the layout of the tree needs to display the hierarchical structure. This property can be assured with the help of the aforementioned level-based approach.
- The second aesthetic property states that edges must not cross, which means the algorithm must produce a planar drawing.
- The third property requires that the drawing of a subtree does not depend on its position in the tree. This assures that isomorphic subtrees will look equal.
- To include the “ordered” leaves property from definition 2, the fourth aesthetic property defines that the order of the children must be displayed.
- Last but not least, the algorithm must work symmetrically. This means, that the reverse input would result in a mirrored tree. This property additionally secures the “ordered” property.

7.3.3 Buchheim-Walker Algorithm

The Buchheim-Walker algorithm is a tree drawing algorithm that fulfills the required aesthetic properties for drawing syntax trees and grew historically. The initial Reingold-Tillford-Algorithm [9] from 1981 for binary trees was adapted in 1990 by Walker [13] to support trees of unbound degree and lastly improved in 2002 by Buchheim [2] to achieve linear time for drawing.

Reingold and Tillford [9] designed their algorithm to draw binary trees in a bottom-up fashion. Leaves and subtrees are independently placed at first before they get merged under a common parent. The parent in such a merge is drawn one level above its children and placed horizontally between them. Merging two subtrees in this way causes the need of avoiding collisions of these subtrees. Reingold and Tillford introduced contours to overcome this problem. A contour of a (sub)tree are the nodes that lie on the “boundary” of its drawing. Contour nodes of the tree in Figure 9 are colored gray. Subtrees are placed as close to each other as possible such that the contours do not collide (and have a minimum distance in

between them). Contour nodes on the same level of subtrees are checked against each other for collisions by comparing their positions and in the case of a collision the whole subtree is shifted. To get the contour of a subtree the leftmost and rightmost nodes are walked. This creates a problem if the leftmost (rightmost) path cannot be walked down to the lowest level. In this case each time a calculation of the leftmost (rightmost) node on the level below would be needed. To overcome this computational overhead so called “threads” are introduced to leaves to save a pointer to the next node of the contour. These threads are visible as dotted arrows in Figure 9. A naive implementation of this algorithm would run in quadratic time as shifting all nodes would result in quadratic time. However, Reingold and Tilford used a technique presented by Wetherell and Shannon [14] and introduced new modifier values to the nodes. While traversing the tree these values are set “on the fly” and are representing the amount of shifting of a node. At the end of the algorithm the tree only have to be traversed once more to compute all real positions in linear time.



■ **Figure 9** Grey marked nodes represent the contours, dotted arrows represent the threads [2]

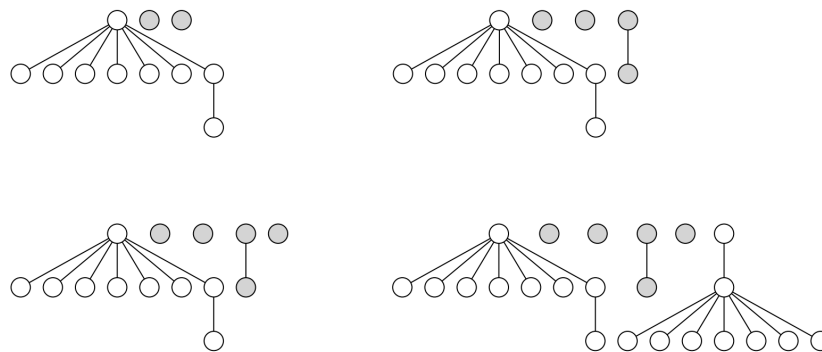
In 1990 Walker [13] adapted the Reingold-Tilford algorithm to be able to draw trees of unbound degree. This adaption made it possible to use this algorithm for syntax trees.

Walker adapted the former algorithm by stating that subtrees can be processed left to right. For each subtree it is evaluated if there is a collision with one of the subtrees left of it and the subtree is then placed accordingly. If the placed subtree needs to be shifted to avoid collisions all subtrees left of it need to be shifted so the subtrees are all spread evenly. Figure 10 illustrates the placing and shifting of subtrees.

The calculation for collisions on the one hand causes much computation, especially because the number of subtrees that each subtree contains needs to be calculated and traversed. On the other hand, the shifting of the subtrees causes the other major computational time. According to [2] this results in the algorithm needing more than linear time for drawing: In total, traversing the contour has a runtime of $\Omega(n^{3/2})$, finding ancestors and summing up modifiers has a runtime of $\Omega(n^2)$ and counting and shifting of nodes needs $\Omega(n^{3/2})$ time.

To solve the problem of Walker’s algorithm being a non-linear drawing algorithm Buchheim et al. improved it in 2002 [2]. Most improvements consisted of saving calculated information rather than calculate that information each time. This needs more memory for saving that information but the recalculation time is significantly improved.

At first threads were reintroduced to get contours easily, like in the original Reingold-Tilford algorithm. Another major problem in Walker’s algorithm was the runtime of finding ancestors and summing up modifiers. These informations are now calculated once and then stored in the corresponding nodes. Additionally, nodes now carry information about their ancestors and the counting of smaller subtrees. This results in a linear runtime instead of a



■ **Figure 10** Placing of subtrees with shifts in Walker's Algorithm [2]

quadratic runtime for these operations.

One other big change was to change when and how nodes are shifted. When checking for collisions and needed shifting, only the currently added subtree is shifted, while the information of how far this subtree needed to be shifted is stored. These values are similar to the modifier values in the original Reingold-Tillford algorithm. Once every subtree is added the values of how far each subtree needs to be shifted are calculated and the shift operations are carried out. This reduces the amount of shift operations for each subtree from a linear to a constant amount.

Looking at the costly operation of Walker's algorithm these improvements erase the problematic parts that run in non-linear time. Placing a subtree is now possible in constant time, which overall makes the algorithm to run in linear time.

The Buchheim-Walker algorithm with its linear time and visually good results is therefore a solid choice for drawing syntax trees.

7.4 Syntax Tree Drawing Tools

Before using syntax trees to describe and explore grammar, these trees need to be drawn. There is a variety of tools that all offer various features and use different approaches to draw syntax trees.

Additional features include coloring of nodes and text and the addition of movement lines like the ones in Figure 2. Some tools also provide the facility to check created syntax trees with the help of provided grammar to make sure there are no mistakes regarding the grammar.

Different drawing tools and techniques can be distinguished in different ways. One major difference is which technique and input is used to produce syntax trees. Besides, tools differ in the amount of features they offer as well as the amount of exporting formats they provide. Common export formats are document formats like PDF and image formats like PNG and JPEG.

In section 7.4.1 an overview of different tools is given before a deeper look into the syntax tree building tool TreeForm is given in section 7.4.2. Finally, a critical look into missing features of TreeForm and a proposal for an alternative is given in section 7.4.3.

7.4.1 Comparison of Tools

The first obvious category of tools is drawing syntax trees by hand. This technique can be quite error-prone which collides with the fact that erasing wrongly drawn parts can be more difficult than in digital versions as sometimes the only option is to redraw the whole tree. However, the major disadvantage of these hand-drawn trees is that they are not eligible for publication.

The next possibility is the technique to draw syntax trees with the help of *special fonts* like *Arboreal* [3] and *Moraic* [4]. According to Derrick et al. [6], most linguists use word processors with these fonts to draw their syntax trees. Drawing syntax trees with fonts is done by writing the labels on every other row. The rows in between will then be filled with special characters that represent the edges. In Figure 11 tree (a) was made with this manual typesetting. Advantage of this technique is its easy approach and that the insertion of the syntax trees into documents is easy. Problem with this approach is that it is error-prone and time-consuming. The user has to figure out manually each problem that an algorithm would solve, like the positioning of subtrees, spacing and others.

\LaTeX macros form the next category. Among the different macros are *qtree* and *synttree*. One of the major problem of \LaTeX macros is a by-product of \LaTeX the absence of a graphical user interface (GUI). This often causes less experienced \LaTeX users to struggle using these macros. Additionally, some macros leave some tasks like spacing to the user, which makes them even less user-friendly.

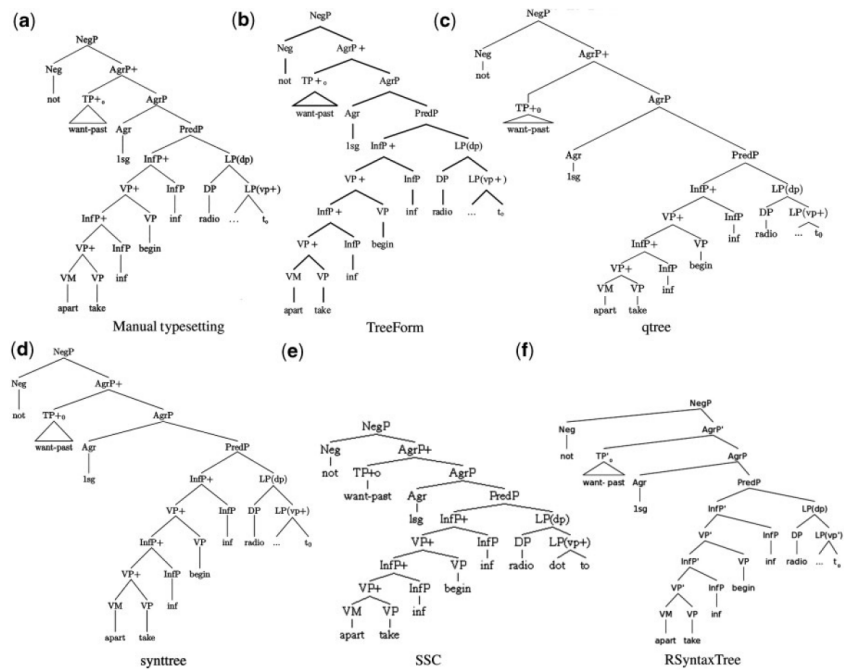
Some \LaTeX macros like *qtree* are *transforming bracket-notation* into syntax trees. Similarly, there are some tools unrelated to \LaTeX that just take the bracket notation of a syntax tree as input and build the tree out of it. One of these is the web-based tool *phpSyntaxTree* [12] where one can enter trees using bracket-notation and the resulting syntax trees can then be exported as PNG or SVG.

Another category of tools are *basic graphic tools*. With almost any graphical software nodes and lines can be drawn and text can be added. Put together in the correct way one can easily form a syntax tree. Similar to e.g. the special fonts, all algorithmic decisions need to be taken by the user, which makes this technique again error-prone and time-consuming.

Better suited to solve these algorithmic decisions are conventional *graph visualization tools* like *Graphviz*, *yEd*, *Pajek*, *visone* and many more. Problematic with these tools is that they are not adopted to syntax trees and therefore are usually not suitable for drawing syntax trees.

To overcome most of the problems stated above specific *Syntax Tree building tools* like *TreeForm* [6] and *TreeBuilder* [11] were developed. Such tools provide an easy accessible way to create syntax trees in no time. These tools usually use an algorithm like the Buchheim-Walker algorithm to draw pleasing syntax trees.

Figure 11 shows a comparison of the results of different tools and techniques. They can be divided into two groups. The first group involving tree a,b and e creates good looking syntax trees. The other group with the rest of the trees have weird looking results as the spacing and positioning of subtrees are less favorable. The weird looking spacing is caused by the following property of the algorithms: Each subtree is drawn left of the leftmost node of the right subtree (and the other way round), without taking into account the level of the nodes. In contrast, better solutions like the Buchheim-Walker algorithm take the contours into account and place subtrees closer to each other, which results in more compact and clearly arranged trees.



■ **Figure 11** Comparison of different Syntax Tree Drawing tools [6]

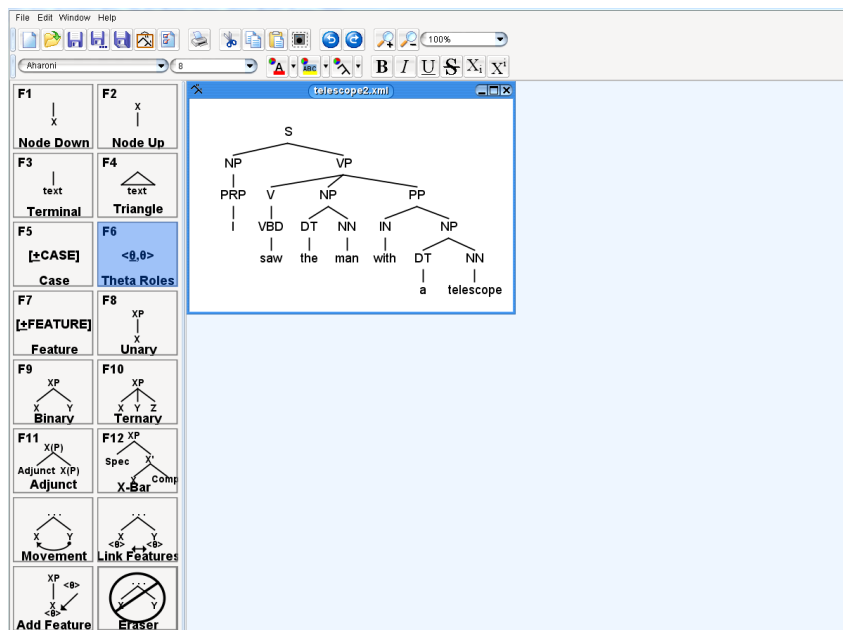
7.4.2 TreeForm

TreeForm is an easy to use *What-you-see-is-what-you-get* (WYSIWYG) syntax tree building tool and provides the user with a clear graphical interface that is shown in Figure 12. The user can utilize simple drag-and-drop elements to build up the syntax tree. Resulting syntax trees are drawn with the help of the Buchheim-Walker algorithm. With this tool first syntax trees can be created in no time without a steep learning curve. Besides building syntax trees TreeForm offers features such as multiple colors and font types, indices and other text formatting features. Additionally, TreeForm supports drawing of movement lines.

The syntax trees can be saved, printed and exported in the image formats PNG and JPEG and as a vector graphic in the document format PDF. On top of that, there is the possibility to copy the tree into the clipboard to then be able to directly insert the tree into word processors like Microsoft Word or its free pendant LibreOffice.

The major benefit of TreeForm is its accessibility and usability. Syntax trees can be easily drawn and the WYSIWYG property underlines this even more. As it is visible in Figure 11 TreeForm also produces good looking syntax trees which is naturally essential. Last but not least TreeForm is also platform independent as it is written in Java.

In contrast to all those benefits there are also some disadvantages. TreeForm supports movement lines reasonably well, but sometimes the positioning of these lines is not beneficial as they might cross nodes or edges. Another disadvantage is that it is impossible to change the structure or position of a subtree without deleting and rebuilding it. TreeForm is good at adding nodes etc. to the tree but offers no other solution to change them except deletion. This can be frustrating for the user when the user only wants to make slight changes but has to rebuild the majority of the tree. Finally, the biggest disadvantage right now is that this tool is not supported anymore and thus gets no more updates. This is problematic as some features are broken when using a current version of Java. The last Java version that was



■ **Figure 12** Screenshot of the graphical user interface of TreeForm

officially supported by this tool was Java 1.6, that lost support in 2013 and thus received no public updates since then.

7.4.3 Missing Features for Treeform and Alternatives

TreeForm can be improved in some aspects to provide an even better experience for the user. At first the disadvantages of TreeForm could be removed. This can be done by at first adding the functionality to move or copy subtrees to enable modification without deletion. Additionally, releasing the source code to open source to make it possible for the community to repair broken features and introduce new functionalities would be beneficial. One of these new features could be an optional automatic generation of syntax trees. An input sentence could be transformed into a syntax tree with tools and libraries like Stanford's CoreNLP [8]. Such an automatic generation of syntax tree could reduce the time needed for creating syntax trees and also would help beginners to get into syntax trees.

An alternative to TreeForm is TreeBuilder [11]. TreeBuilder is a commercial syntax tree building tool that offers the same features like TreeForm and has some additional functionalities as well. The tool itself uses menus to create further elements in the syntax tree in comparison to TreeForm's drag-and-drop solution. In contrast to TreeForm this tool enables the user to edit the tree by moving and inserting parts. It also provides the user the possibility to set a specific grammar and check whether the tree is sufficient regarding that grammar. The biggest disadvantage is that it is commercial, but there is also a free trial version to test this tool.

7.5 Summary

Syntax trees aid in understanding and explaining aspects in grammar. They are an essential tool especially for linguists and other people involved in analyzing grammar. With the help of syntax trees tasks like comparison of differences and commonalities in grammatical structures within sentences as well as e.g. ambiguities can easily be done.

For automatically drawing syntax trees proper algorithms are needed. These algorithms need specific aesthetic properties to create a satisfactory visual representation of syntax trees. One appropriate algorithm is the Buchheim-Walker algorithm that creates syntax trees in linear time. The Buchheim-Walker algorithm was developed out of the Walker-Algorithm to solve the issue of the non-linear time complexity. The Walker-Algorithm in turn was developed out of the Reingold-Tillford algorithm to be able to fix the problem of only supporting binary trees.

Drawing tools implement algorithms to help users create syntax trees. Various tools and techniques have been mentioned in this writeup. Most tools have either the problem of being error-prone and time-consuming or being hard to use and less user friendly. However, syntax tree building tools like TreeForm and TreeBuilder provide the user with a solid and easy to use solution to draw syntax trees that then can be exported to various formats.

References

- 1 Di Battista G, Peter Eades, Ioannis G Tollis, and Roberto Tamassia. *Graph drawing: algorithms for the visualization of graphs*. 1999.
- 2 Christoph Buchheim, Michael Jünger, and Sebastian Leipert. Improving walker’s algorithm to run in linear time. In *International Symposium on Graph Drawing*, pages 344–353. Springer, 2002.
- 3 Cascadilla. Arboreal for mac and arborwin (arboreal for windows). <http://www.cascadilla.com/arboreal.html>, 2016. [Online, last accessed 09.2016].
- 4 Cascadilla. Moraic for macintosh and moraicwin (moraic for windows). <http://www.cascadilla.com/moraic.html>, 2016. [Online, last accessed 09.2016].
- 5 Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454, 2006.
- 6 Donald Derrick and Daniel Archambault. Treeform: Explaining and exploring grammar through syntax trees. *Literary and linguistic computing*, page fqp031, 2009.
- 7 Stanford Natural Language Processing Group. Stanford dependencies. <http://nlp.stanford.edu/software/stanford-dependencies.shtml>, 2016. [Online, last accessed 09.2016].
- 8 Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.
- 9 Edward M. Reingold and John S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, (2):223–228, 1981.
- 10 Adrian Rusu. Tree drawing algorithms. 2004.
- 11 SITTAC. Treebuilder. <http://sittac.com/>, 2016. [Online, last accessed 09.2016].
- 12 IronCreek Software. phpsyntaxtree. <http://www.ironcreek.net/phpsyntaxtree>, 2016. [Online, last accessed 09.2016].
- 13 John Q Walker. A node-positioning algorithm for general trees. *Software: Practice and Experience*, 20(7):685–705, 1990.
- 14 Charles Wetherell and Alfred Shannon. Tidy drawings of trees. *IEEE Transactions on Software Engineering*, (5):514–520, 1979.

8 Graph-based Topic Labelling Using DBpedia

Germaine Götzelmann

Abstract

In text mining and natural language processing, topic modelling is a way to enlist the subjects appearing in text. However, topic models are not easily understood by human users. Given a topic model for a text or a corpus of documents, how can we find fitting labels describing the clustered topics by exploitation of an external knowledge base? The described approach makes use of the graph-based structure of DBpedia, which is a semantic web database derived from Wikipedia articles. Topic words are mapped to their corresponding DBpedia concepts and then concepts that are semantically closely related to them are used to generate subgraphs of DBpedia. With the help of graph centrality measures the top-k labelling candidates are chosen from all nodes in such a graph. This report describes the approach in detail, assesses the evaluation for the explained approach and supplements a possible use case in the humanities.

8.1 Introduction

Topic models have become an increasingly popular way to analyse text corpora since the proposal of the Latent Dirichlet Allocation (LDA) in 2003 [9] (see report 6 in this collection for details). A standard approach for presenting said topic models are simple term lists which contain the top-k topic words of the model [17]. It can be quite difficult to interpret these term lists correctly. Especially the readability for human users can be improved if a semantically fitting word or phrase could be found instead, to describe those top-k words of a given topic. This word or phrase is called a *topic label*.

A good label is characterized as understandable for the user, fitting for all or most given topic words and good for distinguishing topics. The last-mentioned trait excludes labels that are too broad from being good candidates. Previous research indicates that single words tend to be rather generic while whole sentences on the other hand may be too specific to subsume a given topic [18]. The most promising approach seems to be one that does not limit the labels to single terms but also considers short phrases as fitting candidates.

The following report focuses on a paper from 2013 with the title *Unsupervised Graph-based Topic Labelling using DBpedia* by Iona Hulpuş, Conor Hayes, Marcel Karnstedt and Derek Greene [13]. It is a shortened version of chapter 6 of Ioana Hulpuş' PhD thesis *Semantic Network Analysis for Unsupervised Topic Linking and Labelling*, published in 2014 [12]. The basic idea of the presented research is to select topic labels by clustering the content of one or multiple documents into sets of topic words and to link those topic words to concepts in the Semantic Web database DBpedia. From the DBpedia concepts two-hop-neighbourhood graphs are constructed to find the best label candidates among DBpedia concepts with the aid of graph centrality measures. The document processing can be roughly divided in the following steps: extraction of topics from the document(s) (step 1), word-sense disambiguation of the topic words with support of information from DBpedia (step 2), extraction of subgraphs from DBpedia (step 3), selection of topic labels based on the subgraphs (step 4). This report focuses on step 3 and 4.

Within the scope of the PhD thesis, a framework was developed to implement the presented approach. In the paper it is called *Canopy*, the name was later changed to *Kanopy*.

There is a web demo of the framework available online¹ which can be used with custom text input. A short paper with an overview over the framework was published additionally in 2013 [14].

The following report starts with the place of the approach in current scientific research (section 8.2). Afterwards we describe the used structure of DBpedia as well as the core of the graph-based topic labelling in detail (section 8.3). Section 8.4 focuses on the evaluation that was conducted by the authors of the paper. The last part of this thesis discusses the adaptability of the given approach for usage in the humanities (section 8.5).

8.2 Related Work

The related work given by Hulpuş, Hayes, Karnstedt and Greene can be basically divided into two parts. The first one contains work that uses a text-based approach for topic labelling. The approaches in the second part use an external knowledge base for the labelling process. Most of the approaches either exploit Wikipedia and/or a web search engine like Google or Bing. Instead of repeating the related work information in the paper we complement the existing list with more recent papers from 2013 to 2016.

A new text based approach was announced 2015 ‘using Word Vectors and Letter Trigram Vectors’ [16]. Different kinds of vectors are generated from candidate labels within the documents. In 2014, an approach on event extraction based on tweets focused on the problem of the description of events that are happening at the same moment the texts about them are created [7]. Obviously an approach based on static knowledge bases cannot be sufficient for such a task. Summarization techniques (Sum basic, Hybrid TFIDF, Maximal marginal relevance, TextRank) are used instead to exploit the information in the stream of tweets itself.

In 2014 as well, a different unsupervised graph-based approach was presented [4]. In this case, topic words were the basis for a query in a web search engine. A graph is constructed from the result words and then the label candidates in said graph are ranked with PageRank.

Schuhmacher and Ponzetto [22] make use of DBpedia, but mainly for a graph-based computation of document semantic similarity. They adapt Hulpuş’ list of stop-URIs to get rid of unwanted shortcuts in the DBpedia concept relations. 2015, Allahyari and Kochut announced another graph-based topic labelling approach making use of the DBpedia ontology [5] (which was described in more detail most recently in [6] – paper under review). They propose the OntoLDA model, an ontology-based topic model. DBpedia concepts describing topic words are constructed as a graph based on their relations in the DBpedia ontology. The connected subgraph with the largest number of nodes is called the *dominant thematic graph*. The concept nodes are weighted accordingly to their distribution in the topic as well as based on their ‘authoritative’ character (according to the HITS algorithm) in the graph. A semantic relevance scoring function is applied to the gained label candidates. Overall OntoLDA shares a lot of similarities with the approach by Hulpuş et al.

Additionally, an attempt was made to use images for labelling purposes instead of words or phrases[3]. Candidate images are retrieved via a web search with graph-based processing of the given image meta data. With images, a user maybe could identify the topic more quickly than with textual labels and would not depend on a specific language for the topic

¹ Kanopy demo version 1: <http://kanopy.insight-centre.org/KanopyDemo.html>,
updated version 2: <http://kanopy.insight-centre.org/V2/KanopyDemo.html>

labels. In 2015, an evaluation paper compared said image representation of topics with textual labels as well as a list of the top-k terms of a topic model [2]. The results indicate that textual topic labels are a better way to represent topics of document collections than image labels, and do not have significant disadvantage regarding precision compared to term lists.

The Kanopy framework, on which we will concentrate in the following, did not receive a lot of attention. It has been taken note of [11, 22, 4, 2, 5, 6, 10], but mainly as related work only. In 2014, the characteristics of DBpedia graphs regarding semantic relations assessed by Hulpuş et al. were used for a more generic named entity recognition task by Prokofyev et al. [20]. Specifically, they reused the two hop threshold for the graph extraction as well as the list of stop-URIs (see p. 96 for details). Most recently the labelling step by Hulpuş et al. was used in a modified version for the labelling of topic clusters for reader comments of online news articles [1]. We will present evaluation results by this paper in subsection 8.4.

8.3 Graph-based Topic Labelling

For the topic labelling step in the Kanopy framework, the technical prerequisite is to link and disambiguate the words in a topic cluster, extracted from a document, to concept nodes in the DBpedia graph. We consider this task to be done and do not focus further on the challenges that come with it. A detailed explanation of the proposed procedures for the unsupervised word-sense disambiguation (WSD) can be found in chapter 4 and 5 of the PhD thesis [12]. To better understand the graph extraction and the labelling step we first have to have a closer look at the general semantic structure of data in DBpedia.

8.3.1 DBpedia Concepts

DBpedia is a structured knowledge base containing extracted information from Wikipedia articles. There are over 4.5 million entities (like places, persons, species, films, etc.) in the English version of DBpedia alone. The contents of DBpedia are stored as RDF (Resource Description Framework) triples. RDF triples are built in the form of subject – predicate – object and represent simple semantic relations, similar to a simple sentence in natural language with the same structure. All subjects, predicates and objects have unique Uniform Resource Identifiers (URIs) and represent entities of the real world.² The result is a graph database in which subjects and objects of said triples are modelled as nodes and properties form the edges between them. The data can be queried with the SPARQL Protocol and RDF Query Language, which simulates semantic questions like ‘retrieve all musicians born in Berlin, Germany’. SPARQL endpoints for querying are available online or can be set up with a local version of DBpedia. Encyclopaedic information can be retrieved this way without additional preprocessing of unstructured text (as it would be needed to get similar information from Wikipedia articles directly).

To create a well-defined data set, controlled vocabularies (e.g. taxonomies, ontologies) give information about the overall knowledge structure of the database. Typical namespace prefixes for external ontologies used in DBpedia are `rdf` for standard rdf syntax³ and `rdfs`

² Objects can also be literals instead of resources with an URI, but the distinction is not important for explanations in this report.

³ <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

for rdf schema, owl for the Web Ontology Language⁴, skos for the Simple Knowledge Organization System⁵ and dcterms for the extended Dublin Core ontology⁶.

The DBpedia project itself implements mainly two knowledge structures: the DBpedia ontology on one hand and Wikipedia categories on the other. The DBpedia ontology (namespace dbo) is a manually created class hierarchy. It is based on the most common infoboxes in Wikipedia articles and contains over 500 concepts as nodes in a directed-acyclic graph.⁷ The root node for all ontology classes is owl:Thing. Classes link to their parent(s) in the hierarchy with the property rdfs:subClassOf. They also hold information about the child classes (is rdfs:subClassOf of).

The second hierarchical structure in DBpedia stems from Wikipedia article categories (namespace dbc). It is extracted automatically and contains over 700k categories as nodes. A category refers to its parent via the skos:broader property and also has knowledge about the reversed relation (is skos:broader of, sometimes also called skos:narrower). While the relations broader and narrower indicate a semantic hierarchy, however, the “categories do not form a proper topical hierarchy, as there are cycles in the category system and as categories often only represent a rather loose relatedness between articles” [8].

Both hierarchical structures only kind of describe *ideas* or *concepts* of the real world but not specific *things* or *named entities*. Those are represented by DBpedia resources. Resources are the equivalent of Wikipedia articles in structured form. They both link to the DBpedia ontology as well as to the Wikipedia categories. Therefore resources have DBpedia ontology classes as rdf:type and are dcterms:subject in a Wikipedia category (and categories link to their related resources with a ‘is dcterms:subject of’ property vice versa).

Moreover DBpedia is closely linked to the YAGO classification schema⁸. YAGO is a semantic knowledge base extracted from Wikipedia, WordNet and GeoNames. It is characterized by “deepness”, “encoding of much information in one class” and “high accuracy in general” [8]. It is linked to DBpedia resources and structured the same way as the DBpedia ontology: it can be retrieved via the rdf:type property and traversed with rdfs:subClassOf edges.

8.3.2 Topic Graphs

In step 2 of the Kanopy framework topic words get linked to their matching DBpedia concepts. In step 3 so called *sense ego-networks* are extracted for those DBpedia concepts.

► **Definition 1** (Sense Ego-Network). Given a set of DBpedia relation types R , we denote by $DBpedia^R$ the subgraph of the DBpedia semantic network formed only by the relations of types included in R . Then we define the sense ego-network or ego-graph of k^{th} degree of a sense s_i , as an undirected graph $G_i^k = (V_i^k \cup \{s_i\}, E_i, s_i, R)$, where V_i^k is the set of DBpedia concepts lying at most k steps away from s_i in $DBpedia^R$, and E_i is the set of undirected edges between pairs of concepts in $V_i^k \cup \{s_i\}$ that are connected by an edge in $DBpedia^R$. The node s_i is called the **seed** concept or seed node of graph G_i^k . [12, p. 95]

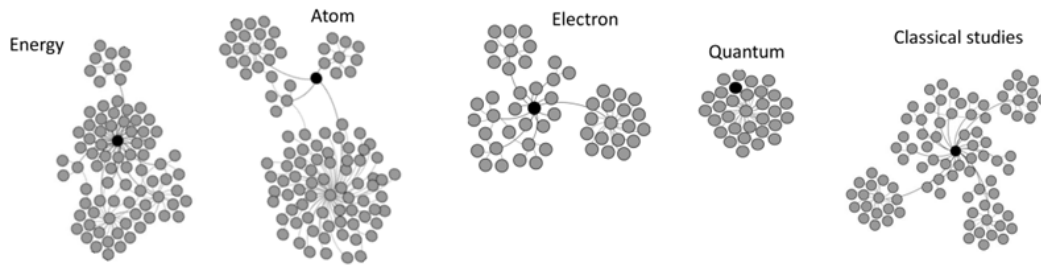
⁴ <http://www.w3.org/2002/07/owl#>

⁵ <http://www.w3.org/2004/02/skos/core#>

⁶ <http://purl.org/dc/terms/>

⁷ See <http://wiki.dbpedia.org/services-resources/ontology>.

⁸ <http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/>



■ **Figure 1** Visualization of a Sense Ego-Network for the DBpedia concepts Energy, Atom, Electron, Quantum and Classical studies (Source: [12, Fig. 6.2])

While there are more properties to a DBpedia concept than the ones we described before, the approach focuses on those relations between resources, classes and categories. So the set R in the definition is limited to the properties described above. With the relations `dcterms:subject`, `skos:broader`, `rdfs:subClassOf` and their reciprocal properties as well as `rdf:type`, the DBpedia ontology classes, YAGO classes and Wikipedia categories are traversed starting from a seed node.⁹

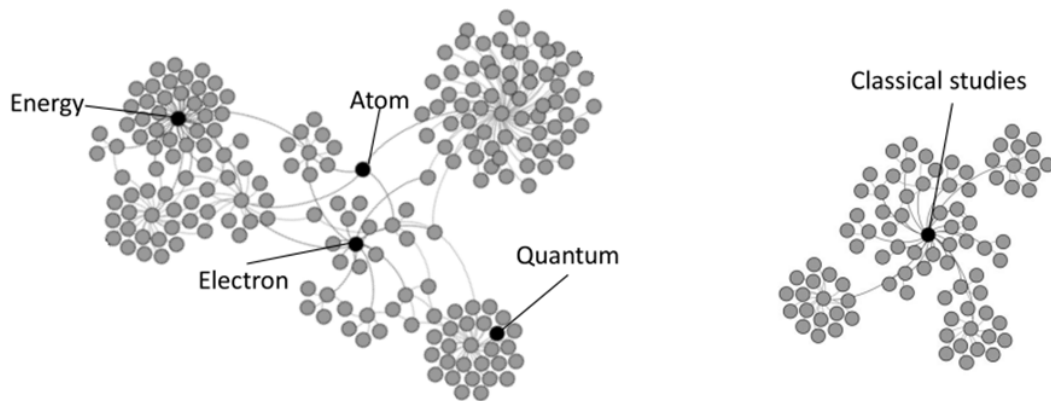
While strictly speaking the properties in R are unidirectional, the graph is constructed with undirected edges. For the sake of simplicity there are no parallel edges between vertices even if they have more than one connection in R . The used ego-networks are constructed with $k = 2$. This was preceded by experiments with 3-hop neighbourhoods but “the potential gain from the analysis of the bigger network does not justify the cost of the computational difference” [12, p. 95]. A visualization of a sense ego-network for the DBpedia concepts ‘Energy’, ‘Atom’, ‘Electron’, ‘Quantum’ and ‘Classical studies’ can be seen in Figure 1. All concepts were disambiguated from topic words of a single topic.

With the sense ego-networks we get semantic concepts in one graph for each topic word that could be linked to a DBpedia concept in the step before. To interlink the topic words of one topic with each other, all ego-networks are merged into one *topic graph*. The topic graph is the result of step 3 and the input for step 4 of the Kanopy framework. A visualization of a topic graph for the concepts ‘Energy’, ‘Atom’, ‘Electron’, ‘Quantum’ and ‘Classical studies’ can be seen in Figure 2.

► **Definition 2 (Topic Graph).** Let $C^\theta = \{C_1, \dots, C_n\}$ be the set of DBpedia concepts corresponding to the disambiguated senses of the words in topic θ . Let R^θ be a set of semantic relations defined in DBpedia. Let $G_i^k = (V_i, E_i, C_i, R^\theta)$ be the sense ego-graph of degree k corresponding to $C_i, i \in 1, \dots, n$. Then, $G = (V, E, C^\theta, R^\theta)$ is the topic graph or topic network of θ , where $V = \bigcup V_i$, and E is the set of undirected edges between nodes in V that correspond to the DBpedia relations R^θ . The concepts $C_i \in C^\theta$ are called the seed concepts or seed nodes of G . [12, p. 139]

In the creation of the ego-networks and the topic graphs two main challenges occur. The first one is, that a 2-step-path in the graph might be obstructed by the fact that oftentimes there are DBpedia resources and DBpedia concepts that share the same

⁹ To be precise additional Infobox properties are used in some of the experimental settings. This is not described here for the sake of brevity. Moreover “[t]he decision of considering only the Types and Category datasets, or also the Infobox properties makes almost no difference” regarding the contents of the resulting graphs (see [12, p. 112] for further details).



■ **Figure 2** Visualization of a Topic Graph for the DBpedia concepts Energy, Atom, Electron, Quantum and Classical studies (Source: [12, Fig. 6.2])

name. To give an example, the topic word *Helium* would be linked to the DBpedia resource *dbp:Helium*¹⁰. It has a *dct:subject* relation to *dbc:Helium*¹¹, which is linked via *skos:broader* to *dbc:Chemical_elements*¹². However the path to *dbc:Atoms*¹³ is already of length 3 but it could provide a viable connection to all kind of concepts within the range of Atomic Physics (i.e. Nuclear Fusion). To improve the results, resources and Wikipedia Categories with the same name are merged beforehand into one node that inherits all relevant relations from both original nodes. So in our example *dbp:Helium* and *dbc:Helium* would be merged and, as an implicit result, *dbc:Atoms* would become a part of the ego-network for *Helium*.

The second challenge for the construction of useful topic graphs are nodes that are closely connected to the seed nodes via property types in *R*, but are not relevant for the determination if two concepts are semantically close. They are instead creating unwanted shortcuts between concepts and can be roughly classified as administrative links (i), semantically irrelevant relations (ii) and high-level ontology links (iii). Examples for the first problem are *dbc:Articles_containing_video_clips*¹⁴ and *dbc:Place_of_birth_missing*¹⁵. (ii) contains Wikipedia categories like *dbc:New_Testament_Latin_words_and_phrases*¹⁶ or *dbc:Living_people*¹⁷. DBpedia concepts are also often linked to *owl:Thing*, even if they are of more specific ontology types at the same time. *owl:Thing* would in consequence shortcut nearly all seed concepts and is therefore an example for problem (iii). The shortcut problem can be minimized with a semi-automatically generated list of 865 stop-URIs which are ignored when creating the ego-networks. Evaluations of the word sense disambiguation results show in regard to the stop-URIs that “their removal greatly improves the performance of all tested measures” and that on the flipside “their tolerance in the semantic network

¹⁰ <http://dbpedia.org/resource/Helium>

¹¹ <http://dbpedia.org/resource/Category:Helium>

¹² http://dbpedia.org/resource/Category:Chemical_elements

¹³ <http://dbpedia.org/resource/Category:Atoms>

¹⁴ http://dbpedia.org/resource/Category:Articles_containing_video_clips

¹⁵ http://dbpedia.org/resource/Category:Place_of_birth_missing

¹⁶ http://dbpedia.org/resource/Category:New_Testament_Latin_words_and_phrases

¹⁷ http://dbpedia.org/resource/Category:Living_people

sometimes renders the path-based semantic relatedness measures unusable” [12, p. 132].

The topic graph of a set of seed concepts is not necessarily a connected graph. Ego-networks can stay separated from others if they share no neighbouring nodes with other seed concepts. This leads to the definition of core topic graphs.

► **Definition 3 (Core Topic Graph).** Let $G = (V, E, C^\theta, R^\theta)$ be the topic graph of θ . G consists of c connected components, with $c \geq 1$. Then a connected component $G_j = (V_j, E_j, C_j^\theta, R_j^\theta), j \in 1, \dots, c$. It follows that $\bigcup C_j^\theta = C^\theta$, and $\bigcap C_j^\theta = \emptyset$. We define a core topic graph any connected component G_j for which $|C_j^\theta| > 1$. The seed concepts that belong to a core topic graph are called core concepts. [12, p. 140]

Seed concepts that stay disconnected from the core topic graph are less related to the other concepts than those that connect with each other. This might indicate that something went wrong in one of the previous steps, so either something was added to a set of topic words by mistake or the linking and disambiguation step lead to the wrong DBpedia concept. All concepts that are not core concepts are eliminated from the labelling process. The core topic graph for the topic graph in Figure 2 consists only of the connected subgraph on the left (with the seeds ‘Energy’, ‘Atom’, ‘Electron’ and ‘Quantum’). The subgraph for the concept ‘Classical studies’ has been discarded. This is fitting, because the concept does not have anything in common with the other terms of the topic and most likely is the result of an erroneous disambiguation.

Every core concept, on the other hand, can be considered as a label candidate. Hulpuş states that “usually the topic graphs have only one core component, and the concepts that are not connect to it are isolated” [12, p. 140]. If there are multiple core graphs for one topic they are treated as separate topics from this point on. The impact of the stop-URIs are described as follows, regarding the structure of the core topic graph created for the evaluation process (for details of the evaluation see subsection 8.4): if no stop-URIs are used, 9 out of 10 core topic graphs are identical with the overall topic graph. Excluding stop-URIs instead, only 16% of the topic graphs still connect every seed concept. But the other 84% still connect approximately 2/3 of the seed concepts on average in one core topic graph [13]. This indicates that shortcuts exaggerate the connectivity and therefore hinder further analysis of the graph, while after a removal of the shortcuts a reasonable amount of seeds stay connected.

8.3.3 Centrality Measures

After the core topic graph is computed, the question is how to choose the most fitting label candidate(s) from all nodes in said graph. To state the problem more formally:

Problem Statement Let C^θ be the set of seed concepts in a core topic graph. We want to identify the concept C^* from all core concepts such that the relation $r(C^\theta, C^*)$ is optimised.

The given problem can be solved by the use of graph centrality measures. Centrality is a term from social network analysis and it aims to identify the most important or influential actor in a network. With centrality measures, a ranking of importance can be determined among the nodes of a graph. How this importance is defined depends highly on the characteristics of the given network as well as on the task you want to achieve. Hulpuş et al. chose four popular centrality measures for their experiments: closeness centrality, betweenness centrality, information centrality and random walk betweenness centrality. Moreover, they propose a focused variant for each of the centrality measures, which concentrates on the seed

nodes instead of the complete core topic graph. In the following all centralities are described both with their basic definition as well as in their focused variant. First of all, *closeness centrality* ranks a node as important in a network, if it has a short distance to all other nodes. It is computed as the inverse of the average shortest distance from a node to all others in the graph.

► **Definition 4** (Closeness Centrality). Let dist_{ij} be the shortest path between two nodes v_i and v_j in a graph $G = (V, E)$. Then the closeness centrality for v_i is calculated as:

$$\text{CC}_i = \frac{|V|}{\sum_{v_j \in V} \text{dist}_{ij}}$$

When finding a fitting label, we are not particularly interested in all distances, but especially in short distances from a node to the seed concepts. Because of that, a variant that is better suitable for the task at hand can be computed as the inverse of the average shortest distance from a node to all seed concepts in a core topic graph¹⁸. It is called the *focused closeness centrality*.

► **Definition 5** (Focused Closeness Centrality). Let $|C^\theta|$ be the number of seed concepts and C^θ the set of seed concepts in a core topic graph G . The focused closeness centrality can be calculated as:

$$\text{fCC}_i = \begin{cases} \frac{|C^\theta|}{\sum_{v_j \in C^\theta} \text{dist}_{ij}}, & v_i \notin C^\theta \\ \frac{|C^\theta| - 1}{\sum_{v_j \in C^\theta} \text{dist}_{ij}}, & v_i \in C^\theta \end{cases}$$

Information centrality measures importance based on the assumption that the amount of information flow on a given path is higher the shorter the path. Accordingly, two nodes are more closely related the more (preferably short) paths they have between them. Information centrality as the result is calculated as the harmonic mean of informations between the target node and all other nodes in G .

► **Definition 6** (Information Centrality). Let $I_{ij} = \sum_{p \in P} \frac{1}{\text{length}(p)}$ be the information between two nodes in G with P being the set of paths between v_i and v_j . Then the information centrality for v_i can be calculated as:

$$\text{IC}_i = \frac{|V|}{\sum_{j=1}^{|V|} 1/I_{ij}}$$

Focused information centrality limits the computation to the information flow between seed nodes instead.

► **Definition 7** (Focused Information centrality).

$$\text{fIC}_i = \begin{cases} \frac{|C^\theta|}{\sum_{v_j \in C^\theta} 1/I_{ij}}, & v_i \notin C^\theta \\ \frac{|C^\theta| - 1}{\sum_{v_j \in C^\theta \setminus v_i} 1/I_{ij}}, & v_i \in C^\theta \end{cases}$$

¹⁸The notations of the following definitions may differ slightly from the ones in [12], hopefully in a beneficial way.

The focused information centrality favours nodes that are passed by a high amount of paths between seed concepts regardless of the path length. This makes the focused information centrality biased towards high degree nodes. It is, in contrast to closeness centrality, computed using all paths in the core topic graph and not only the shortest ones.

Betweenness centrality is computed as the proportion of shortest paths through a node in relation to all shortest paths in the graph between all nodes. It is scaled by the total number of pairs of nodes in the undirected graph G , excluding v_i . Betweenness centrality ranks a node in a network by the amount of control the node has over communication in the network, if we assume that control flows over shortest paths (see [15]).

► **Definition 8 (Betweenness Centrality)**. Let g_{st} be the number of shortest paths between nodes v_s and v_t in G , and x_{st}^i the number of those shortest path that pass through v_i . Then betweenness centrality for v_i can be calculated as:

$$BC_i = \frac{\sum_{v_s, v_t \in V} \frac{x_{st}^i}{g_{st}}}{(|V| - 1)(|V| - 2)/2}$$

Betweenness centrality is biased towards nodes with a high degree, since those tend to lie on a higher number of shortest paths. For the labelling task we can once again focus on the shortest paths between seed concepts only. The *focused* variant of the *betweenness centrality* only computes the proportion of shortest paths between seed concepts through the target node v_i , instead of all shortest paths in G . The denominator is adjusted depending on v_i being one of the seed concepts or not.

► **Definition 9 (Focused Betweenness centrality)**. Let $|C^\theta|$ be the number of seed concepts and C^θ the set of seed concepts in a core topic graph G . The focused betweenness centrality can be calculated as:

$$fBC_i = \begin{cases} \frac{\sum_{v_s, v_t \in C^\theta} \frac{x_{st}^i}{g_{st}}}{|C^\theta|(|C^\theta| - 1)/2}, & v_i \notin C^\theta \\ \frac{\sum_{v_s, v_t \in C^\theta} \frac{x_{st}^i}{g_{st}}}{(|C^\theta| - 1)(|C^\theta| - 2)/2}, & v_i \in C^\theta \end{cases}$$

Experiments indicate that betweenness centrality often results in high ranks for seed concepts, while the focused betweenness centrality tends to favour nodes between the seed concepts, which does correspond better to intuitive label candidates.¹⁹

Random walk betweenness, as a variant of betweenness centrality, measures how often a node is passed on a random walk in the graph. The basic concept is, that a start node in a network wants to send a message to a target node but has no shortest path information. Instead it sends its message randomly to a neighbouring seed which continues to do so until the message eventually reaches the target node. The idea of a random walk betweenness centrality was introduced by [19]. While Hulpuş et al. call their computed measure Random Walk Betweenness (RWB), in fact they use a current-flow betweenness, which is equivalent to RWB for all vertices in a network and “is simple and intuitive and makes for easy calculations” [19]. For explanation of current-flow betweenness, we picture the given graph as a current flow network with identical electrical resistance at all edges. The sum of the current flowing

¹⁹For example, to label the topic words ‘Atom’ and ‘Electron’, one would expect that a label should describe both terms equally well, instead of using ‘Atom’ or ‘Electron’ themselves as a topic label.

into a node in the circuit is the same as the sum of the current flowing out of it (according to Kirchoff's current law). For a source s and a sink t we can calculate the throughput for a node v inbetween. If we consider all possible st -pairs, the fraction of throughput through v for all pairs (st) defines the current-flow betweenness centrality. For sake of consistency we keep the name RWB for this measure and for simplicity we substitute the definition used by Newman and by Hulpuş et al. with a definition given by [15].

► **Definition 10** (Random Walk Betweenness / Current-Flow Betweenness). Let G be a flow network with a current x between a source s and a sink t . Let $\delta(v) \subseteq E$ be the subset of edges that are incident to v . Moreover, we define b_{st} , the supply of one (abstract) unit that enters the network at s and leaves at t . It is defined as $b_{st}(s) = 1$, $b_{st}(t) = -1$ and $b_{st}(v) = 0$ otherwise. We define the throughput τ for a node v_i as:

$$\tau_{st}(v_i) = 1/2 \left(-|b_{st}(v_i)| + \sum_{e \in \delta(v_i)} |x(e)| \right)$$

and the centrality measure of v_i as following:

$$\text{RWB}_i = \frac{\sum_{v_s, v_t \in V} \tau_{st}(v_i)}{(|V| - 1)(|V| - 2)}$$

Note, that for the throughput both current flowing in and out of a node is summed up, and to account for this the result is divided in half afterwards.

Instead of accounting for all possible pairs of source s and sink t , a focused variant of the centrality measure only takes into account pairs of seed concepts in G to compute centrality for flow between them.

► **Definition 11** (Focused Random Walk Betweenness).

$$\text{fRWB}_i = \begin{cases} \frac{\sum_{v_s, v_t \in C^\theta} \tau_{st}(v_i)}{|C^\theta|(|C^\theta| - 1)}, & v_i \notin C^\theta \\ \frac{\sum_{v_s, v_t \in C^\theta} \tau_{st}(v_i)}{(|C^\theta| - 1)(|C^\theta| - 2)}, & v_i \in C^\theta \end{cases}$$

The result is biased towards nodes with high degree since they handle more flow in the current network and they are more likely to be used by a random walker.

Hulpuş et al. experimented with all given focused variants of centrality measures. The top-k ranked concepts in every core topic graph were extracted as candidates for the topic labels. They compare the centrality measures and calculate the Pearson correlation coefficient as shown in Table 1. The centrality measures fCC and fIC have a high correlation as well as fBC and fRWB. The measures fCC and fBC rely on shortest paths only, while fIC and fRWB take all paths in the core topic graph into account. As a result, fIC and fRWB are the preferred representatives for the closely correlated measures and chosen to be further evaluated.

8.4 Evaluation

To evaluate the results of automatic topic labelling, they can either be compared to corpora that were labelled manually beforehand, or by presenting the labelling candidates to human

■ **Table 1** Pearson correlation coefficients (PCC) of focused centrality measures

	fCC	fBC	fIC	fRWB
Degree	0.3365	0.4889	0.5072	0.6620
fCC	1	0.4432	0.7967	0.5118
fBC		1	0.4967	0.8923
fIC			1	0.6436
fRWB				1

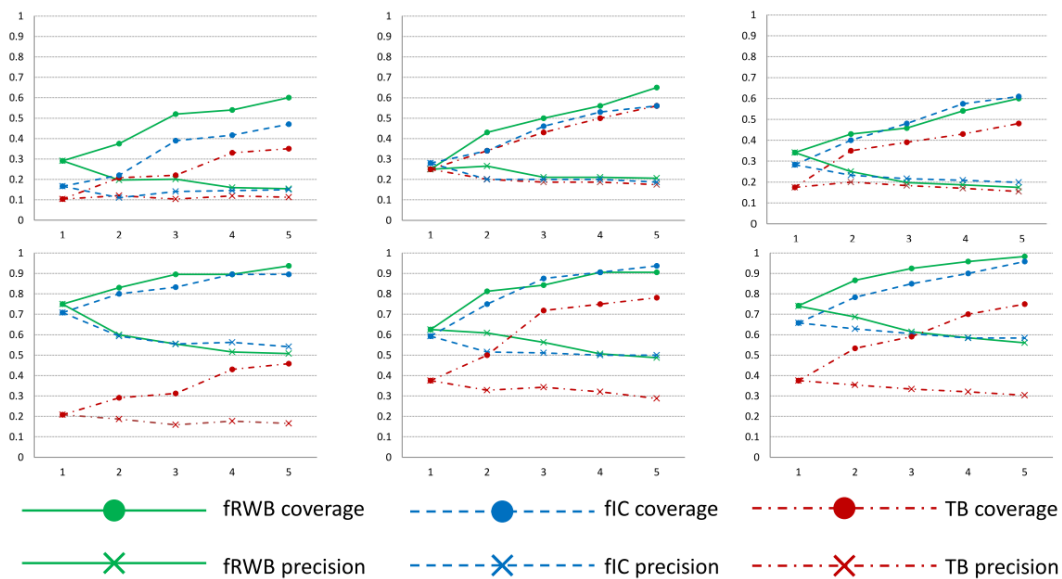
annotators to assess the quality of the labels. Hulpuş et al. decided to use the latter. They chose three corpora from very different domains and with different text styles: the British academic written English Corpus (BAWE), which is a corpus of assessed student writings from 35 different scientific disciplines; the BBC corpus, containing news website texts, and the Stack Exchange dataset with discussion threads from topics in wordpress, webmasters, web applications, photography, gaming, game development, android, cooking and bicycles. There were 54 users in the study (mostly PhD students and post-docs). Each automatically produced label candidate was shown to three users.

From BAWE 150 topics were extracted, 50 from BBC and 50 from Stack Exchange. Topics considered to be ‘hard to grasp’ or too domain specific for a user study were removed manually afterwards (30 topics were eliminated from BAWE, 18 from BBC and 2 from Stack Exchange). The topic labels were extracted using fIC and fRWB. For comparison additional labels were extracted with the text-based approach of [18]. The labels were presented to the annotators in a questionnaire in randomized order. The top-n topic words for the labelled topic were shown to the users. They had to choose one rating for every label: ‘good fit’, ‘too broad’, ‘related but not a good label’ and ‘unrelated’.

8.4.1 Results

From the results for the top-k labels *precision* was computed as $\frac{\# \text{Hits with rank} \leq k}{k}$ and *coverage* as $\frac{\# \text{topics with at least one hit at rank} \leq k}{\# \text{topics}}$. A label was considered to be a *hit* if at least two out of the three annotators marked it accordingly. From one and the same user study two results were computed. The first only took labels assessed as ‘good fit’ into account, the second shows results for all labels that were marked either as ‘good fit’ or as ‘too broad’. Results can be seen in Figure 3. We only show the results for the three corpora separately since they give more detailed information than the results combined.

In the comparison with labels that are considered as good fit only, the improvements over the text-based approach are only minimal except for the Stack Exchange corpus. Especially the fRWB approach boosts the results significantly in this case. On the other hand the performance is nearly identical when used on the BBC corpus. This might partially be the case because news texts tend to state their topics very clearly and in a lot of variation regarding the used terms. The typical use of buzzwords in journalistic texts can favour a text-based approach and limit the improvement due to an external knowledge base. The result is very different with labels that are a good fit or broader. The Kanopy framework outperforms the text-based approach significantly and reaches a maximum coverage of over 90% in all corpora.



■ **Figure 3** Evaluation results for the corpora. Top row: “good fit”, bottom row: “good fit or broader”. From left to right: Stack Exchange, BBC, BAWE. x axis: top-k labels, y axis: Precision and Coverage (Source: [12, Fig. 6.10])

8.4.2 Assessment of the Evaluation Results

Overall the evaluation results were presented in a comprehensible and structured way. There are some grounds for criticism. Two major points concern the baseline of the evaluation and the separation of the evaluation into results for good terms on one hand and for good and broader terms on the other.

First, Hulpuş et al. compare their approach to the ‘state of the art text-based approach’ [18] under the presumption that their own solution will be less sensitive to the text style and the domain specific characteristics of the tested corpora. The assumption itself would be fine, but while [18] compare two implementation alternatives, Hulpuş et al. chose the variant with an OpenNLP noun-chunker, which is trained on news texts. Therefore, it is not surprising that the baseline performs better on the BBC corpus than on the other two, and it is an unfair comparison in this regard.

Second, the separation into evaluation of ‘good fits’ and ‘good fits and broader’ emphasizes an improvement if broader labels are acceptable. The problem that we see with this indication is, that even a much simpler approach might be able to produce reasonably good results if broader labels are considered acceptable. Earlier in the paper/thesis, an approach which computes the least common subsumer in a hierarchical knowledge base (like the Wikipedia categories) is dismissed as a solution because it “would most often produce very generic terms” [12, p. 138]. A very generic term would be considered as a hit in the ‘good fit or broader’ evaluation. There is no grading for the labels that are assessed as broad and therefore no information is gathered about how acceptable those labels could be. For example the label *entertainment* for the topic [song, rock, single, lyrics, instrument] could be an acceptable one depending on the use case (while *music (industry)* could be more fitting), the label *life* for [health, healthcare, hospital, drug, professional] most likely could not. Both might get annotated as ‘too broad’ in a user study.

The problem with the acceptance of broader labels might have already been confirmed

by [1]. They combined a topic clustering approach with the topic labelling step by Hulpuş et al. for reader discussions/comments of online news. The results were evaluated as following: they presented 22 extracted comment clusters and their labels to three annotators. For every label three clusters were randomly selected to be suggested along with the one belonging to the label. The annotators chose one or multiple clusters as being described best by the current label. Precision, recall and F-score were calculated based on the annotation results. There was also the possibility to mark the label as NA (not annotated), if it was not fitting or “too generic or even very abstract”. The authors reported an overall acceptable precision (0.80 mean) but not only with a low recall (0.50), but also with a very high percentage in NA annotations of 40.9% (mean value).²⁰ The problem seems to be pretty subjective since the numbers differ vastly between the three annotators. One marked 13 labels out of 22 as NA while another one only marked two. Overall the results might indicate that broader labels can indeed mean that they are not accepted as a label at all by a human reader and therefore are rendered useless. Overall the study design of [1] seems to be more fitting to determine if computed labels are useful to a human user.

The given major points of criticism we would like to supplement with some smaller comments on details of the evaluation. Due to the lack of comparison with a different knowledge base, it is undecidable (as stated by [13] themselves) if the leaning towards labels that are too broad is linked to the method or to the specifics of the knowledge base DBpedia. Moreover, it would have been interesting to additionally present the seed concepts themselves to the annotators. This would have given more insight about the intuitive assumption that centrality measures with lower ranking of seed concepts are resulting in more fitting labels overall. It also could have provided more insight, if there are still DBpedia concepts in the core topic graphs that are falsely disambiguated (and marked as ‘unrelated’ by a human annotator as a result), and still influential enough in the topic graph to be among the top-5 label candidates.

The user study both eliminates domain specific topics and focuses on coherent topics only. This leads to some blind spots regarding more domain specific labelling tasks, as well as corpora with sparse content, which can make topic clusters less coherent. While this is understandable for the evaluation, we might encounter both challenges (domain specific texts and sparse content) on a regular basis if we want to use the approach for humanities use cases.

8.5 Adaptability for Digital Humanities

As was outlined in subsection 8.2, the Kanopy framework did not get a lot of resonance so far. There are no real world use cases known for the approach. In this section we discuss the general possibility for a usage in the range of humanities. We focus on the upcoming challenges first and then present an example for a use case that could have profited from automatic topic labelling.

²⁰Since they comment on possible links like *Articles containing video clips*, it seems that there was no list of stop-URIs used like it is described by Hulpuş et al. to solve the problem of shortcuts (see subsection 8.3.2). So worse results than necessary might have been produced due to noisy data.

8.5.1 Challenges

In general it is easy to see why a database like DBpedia is appealing for exploitation. It is a well-structured, well maintained information source. DBpedia as a knowledge base is derived from Wikipedia content. Wikipedia with a vast amount of contributors from all kinds of domains provides information about nearly every thinkable topic – often up-to-date and with potentially more than decent quality in at least some fields of interest. Overall there are two main problems for topic labelling tasks in the scope of humanities. The first one is about the adaptability for scientific use in general. Since Wikipedia is at least partially considered to be a unreliable source of information with low quality content in some domains, DBpedia naturally would share that categorization. Moreover, DBpedia and Wikipedia seem to be a universal, nearly inexhaustible data base, but this can obscure the fact that this sort of encyclopaedic world-knowledge naturally comes at a cost in details. Instead of aiming for completeness, Wikipedia has clear-cut relevance filters and a notability guideline to focus on the things that are ‘really important’. The problem with this are not only the resulting gaps, but also the a-priori assumption that the entities in Wikipedia/DBpedia are more important than the missing ones. By using a knowledge base not well suited for a scientific interest, we not only get bad results, we are also in danger of diminishing the research topic itself.

A second problem can occur with any research topic that deals with historic texts. There is no such thing as stability if it comes to knowledge and language. More specific: world knowledge as a whole is not historically stable; semantic meanings are not historically stable and also semantic relations between concepts are not historically stable. The phenomenon regarding semantic meanings is called *semantic shift* and it can produce a problem that in language learning is called *false friends*. The main problem with this is that it would result in false positives, which could only be eliminated in a fully automated labelling process with Kanopy, if they are not related well enough to the rest of the topic and – because of this – are not a part of the core topic graph.

These problems can likely be overcome if the interlinking potential of the Linked Open Data (LOD) Cloud is used. The ‘data hub’ DBpedia could be supplemented or even exchanged with more domain specific knowledge already being part of the LOD cloud. This also might provide an opportunity to encourage researchers in the humanities to publish their own data accordingly to the Linked Open Data standards, and thus to enrich the domain specific knowledge openly available.

8.5.2 Use Case Example

Overall automatic topic labelling might find interesting use cases in the range of humanities. We want to conclude this report with an example from German literature studies. It evolves around one of the most important awards in German literature, the *Ingeborg-Bachmann-Preis* (Bachmannpreis in short) which is awarded in a yearly television show called *Tage der deutschsprachigen Literatur* (aka *Festival of German-Language Literature*). The show airs on the Austrian network ORF and lasts 2–3 days. Up to 23 authors are reading a short story live on television and are rated by a jury afterwards. Since the award was first given out in 1977, the texts within the Bachmannpreis form a relatively large corpus. While some of the texts can only be accessed in archives, over 400 of the texts are published in anthologies and are readily available for research purposes. Since the Bachmannpreis can have quite an impact on the following (sometimes life-long) success of the competing authors, it is interesting to see if the winning texts themselves are deemed to be of a lasting quality. Usual hermeneutic methods in literature studies however are not suitable to accomplish such a

distant reading task. A PhD thesis, finalised in 2014, instead quantitatively analysed the corpus of all published texts of the Bachmannpreis from 1977 up until 2009 [21]. It focused on the question what kind of text the jury appreciated the most over the years and if there is such thing as a ‘prototypical’ Bachmannpreis text. The research method was a manually conducted content analysis. One of the questions in the so called code book asked for the topic of every text. 27 possible categories like ‘religion’, ‘love, relationship, marriage’, ‘family’, ‘friendship’, ‘illness and death’ were defined beforehand.²¹ Every text could be annotated with 1–3 topic labels. The resulting tables can be summarized as shown in Table 2 (in an abbreviated version).

■ **Table 2** Topics in texts of the Bachmannpreis. Percentages by decades in relation to texts.

Topic	1977–79	1980–89	1990–99	2000–09
	% of texts	% of texts	% of texts	% of texts
Current Affairs	21.1%	12.2%	5.7%	3.5%
Family	18.4%	16.8%	17.0%	25.6%
Illness,Death	13.2%	16.0%	11.3%	10.5%
Love	15.8%	18.3%	6.6%	10.5%
Daily Life	7.9%	6.9%	15.1%	9.3%
Work Life	5.3%	11.5%	10.4%	11.6%
History	5.3%	9.2%	9.4%	12.8%

To accomplish a task like this, automatic topic labelling could be highly beneficial. We would gain a more objective result, since the parameters for the approach could be made transparent and the results could be reproduced. The labelling process would be a lot faster than manual coding and it could be easily adapted and repeated in case the underlying hypothesis shifted. There would be no need to define the labels beforehand, which would minimize the risk to come across texts that cannot be subsumed under the chosen topics.

The obvious drawbacks are that this would only be useful for a very small part of the overall analysis. Moreover, fictional literature poses a particularly difficult task since it is highly polysemantic and ambiguous. An automatic tool cannot entirely substitute a human reader, let alone a domain expert. Nonetheless, it seems plausible that with such an approach a researcher could be supported in his/her study and be relieved of some of the simpler tasks in favour of concentrating on data analysis and interpretation.

8.6 Conclusion

The described approach indicates possible improvements in comparison with a text-based approach especially if labels are allowed to be broad, and if coverage is more important than precision. However, the margin for benefits is rather small, and we have shown that it is open for discussion if broader labels are acceptable or not. This determination is rather subjective and might also only be possible within specific use cases but not as a general measure.

The approach of Hulpuş et al. has the clear benefit of being usable without preprocessing and completely unsupervised. The existing Kanopy web demo is a valuable asset for testing both time performance and result quality with custom text examples. The Kanopy framework

²¹ One text from every year was used to test the labels before the actual coding started.

is not meant to be “ready for fully automated labelling” yet [13]. There is no indication of further updates at the moment. To find the most promising approach it would have to be evaluated against more recent research in regards of topic labelling. Especially OntoLDA ([5, 6]) seems to be an interesting and comparable approach with active development. One of the most interesting distinctions is that OntoLDA transfers the probability distribution in the underlying topic model into weights in the extracted graphs while Kanopy treats topic clusters as bags of topic words only. Lately, [10] have raised criticism that the separation of the topic modelling and the graph labelling steps could “result in the loss of rich semantics”. Moreover they pointed out that, while being unsupervised with DBpedia, the approach is based on a very specific subset of relations and would have to be adapted accordingly for every change of the used knowledge base.

In summary, frameworks for automatic topic labelling could be beneficial to support researchers with their tasks in humanities. We have presented an example for such a use case regarding a quantitative analysis of the *Festival of German-Language Literature*. However, for the usage of an encyclopaedic knowledge base its limitations have to be evaluated beforehand, to assess the benefits for a specific task. Since the method is based on Semantic Web data structure, it should further be examined if there are more domain specific datasets in the LOD cloud that could be exploited for scientific use cases. Especially with historic documents, a text-based approach might be superior, if it does not inflict modern world knowledge and semantics, but mainly uses the connections within the text.

References

- 1 Ahmet Aker, Emina Kurtic, A. R. Balamurali, Monica Paramita, Emma Barker, Mark Hepple, and Rob Gaizauskas. A Graph-Based Approach to Topic Clustering for Online Comments to News. In Nicola Ferro et al., editors, *Advances in Information Retrieval*, pages 15–29. Springer International Publishing, Cham, 2016.
- 2 Nikolaos Aletras, Timothy Baldwin, Jey Han Lau, and Mark Stevenson. Evaluating topic representations for exploring document collections. *Journal of the Association for Information Science and Technology*, 2015.
- 3 Nikolaos Aletras and Mark Stevenson. Representing topics using images. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 158–167, Atlanta, GA, 2013.
- 4 Nikolaos Aletras and Mark Stevenson. Labelling Topics using Unsupervised Graph-based Methods. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 631–636, 2014.
- 5 Mehdi Allahyari and Krys Kochut. Automatic Topic Labeling Using Ontology-Based Topic Models. In *IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 259–264, 2015.
- 6 Mehdi Allahyari and Krys Kochut. OntoLDA: An Ontology-based Topic Model for Automatic Topic Labeling. <http://www.semantic-web-journal.net/content/ontolda-ontology-based-topic-model-automatic-topic-labeling>, 2016.
- 7 Amparo Elizabeth Cano Basave, Yulan He, and Ruifeng Xu. Automatic Labelling of Topic Models Learned from Twitter by Summarisation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 2: Short Papers*, pages 618–624, 2014.
- 8 Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009.
- 9 David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *The Journal of Machine Learning Research*, 3:993–1022, March 2003.

- 10 Long Chen, Joemon M. Jose, Haitao Yu, Fajie Yuan, and Dell Zhang. A Semantic Graph based Topic Model for Question Retrieval in Community Question Answering. In *WSDM '16 Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 287–296, 2016.
- 11 Besnik Fetahu, Stefan Dietze, Bernardo Pereira Nunes, Marco Antonio Casanova, Davide Taibi, and Wolfgang Nejdl. A Scalable Approach for Efficiently Generating Structured Dataset Topic Profiles. In David Hutchison et al., editors, *The Semantic Web: Trends and Challenges*, pages 519–534. Springer International Publishing, Cham, 2014.
- 12 Ioana Hulpuş. *Semantic Network Analysis for Unsupervised Topic Linking and Labelling*. PhD thesis, National University of Ireland, Galway, 2014.
- 13 Ioana Hulpuş, Conor Hayes, Marcel Karnstedt, and Derek Greene. Unsupervised Graph-based Topic Labelling using DBpedia. In *WSDM '13: Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, pages 465–474. ACM, 2013.
- 14 Ioana Hulpuş, Conor Hayes, Marcel Karnstedt, Derek Greene, and Marek Jozwicz. Kanopy: Analysing the Semantic Network around Document Topics. In David Hutchison et al., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 677–680. Springer, Berlin, Heidelberg, 2013.
- 15 Dirk Koschützki, Katharina Anna Lehmann, Leon Peeters, Stefan Richter, Dagmar Tenfelde-Podehl, and Oliver Zlotowski. Centrality Indices. In David Hutchison et al., editors, *Network Analysis*, pages 16–61. Springer, Berlin, Heidelberg, 2005.
- 16 Wanqiu Kou, Fang Li, and Timothy Baldwin. Automatic Labelling of Topic Models Using Word Vectors and Letter Trigram Vectors. In Guido Zuccon et al., editors, *Information Retrieval Technology*, pages 253–264. Springer International Publishing, Cham, 2015.
- 17 Jey Han Lau, Karl Grieser, David Newman, and Timothy Baldwin. Automatic Labelling of Topic Models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 1536–1545, Stroudsburg, Pa., 2011. Association for Computational Linguistics.
- 18 Qiaozhu Mei, Xuehua Shen, and Chengxiang Zhai. Automatic labeling of multinomial topic models. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 490–499, 2007.
- 19 M.E. J. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27(1):39 – 54, 2005.
- 20 Roman Prokofyev, Gianluca Demartini, and Philippe Cudré-Mauroux. Effective Named Entity Recognition for Idiosyncratic Web Collections. In *Proceedings of the 23rd international conference on World wide web*, pages 397–408, 2014.
- 21 Karin Röhrich. *Wettlesen um den Ingeborg-Bachmann-Preis: Korpusanalyse der Anthologie Klagenfurter Texte (1977-2011)*. Angewandte Literaturwissenschaft. StudienVerlag, Innsbruck, Wien, Bozen, 2016.
- 22 Michael Schuhmacher and Simone Paolo Ponzetto. Knowledge-based Graph Document Modeling. In Ben Carterette, Fernando Diaz, Carlos Castillo, and Donald Metzler, editors, *WSDM '14 Proceedings of the 7th ACM international conference on Web search and data mining*, pages 543–552, 2014.