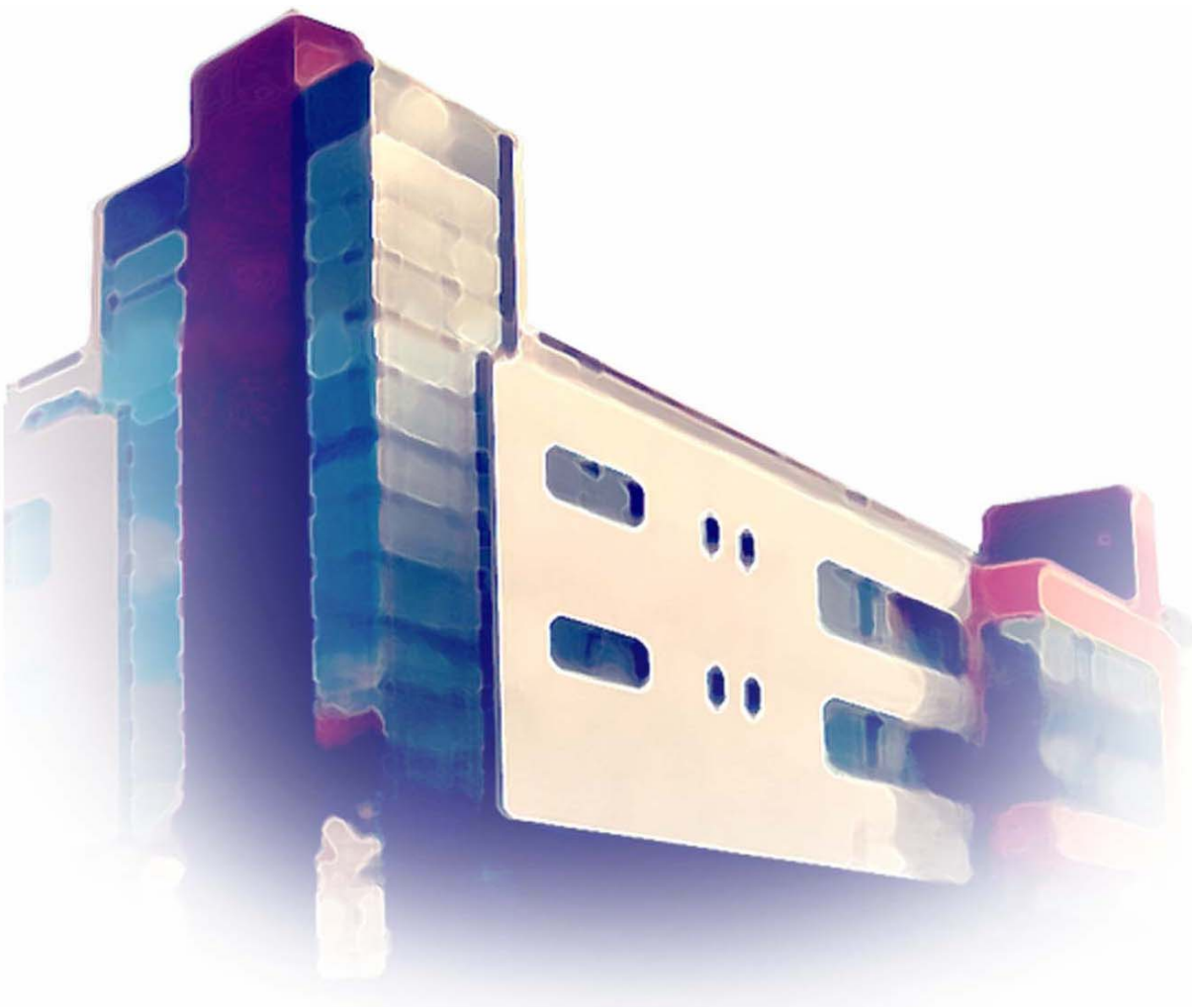


Peter Geibel & Brijnesh J. Jain (Eds.)

LNVD 2007
Learning from Non-Vectorial Data

Proceedings of the KI-2007 Workshop
held in Osnabrück, Germany, September 10th, 2007



PICS

Publications of the Institute of Cognitive Science

Volume 6-2007

ISSN: 1610-5389

Series title: PICS
Publications of the Institute of Cognitive Science

Volume: 6-2007

Place of publication: Osnabrück, Germany

Date: December 2007

Editors: Kai-Uwe Kühnberger
Peter König
Petra Ludewig

Guest editors: Peter Geibel, Brijnesh J. Jain

Cover design: Thorsten Hinrichs

Preface

In recent years, machine learning approaches that operate on non-vectorial data like sequences, trees, graphs, and logical descriptions have gained increasing importance. This is primarily due to new exciting application areas like bioinformatics, text/web mining, and computer linguistics. In these fields, data is often represented by sequences, trees, and graphs of varying length. Examples are DNA molecules and proteins in bioinformatics, or words and documents in text mining.

Recently, new powerful tools for solving classification tasks and other learning problems have been developed. Examples include string, tree, and graph kernels, learning in non-metric distance spaces, conditional random fields, learning in I/O space, spectral methods for graphs, probabilistic relational models, recurrent neural networks for structured data, inductive logic and functional programming, multi-relational data mining, analogical learning and ontology rewriting.

The workshop *Learning from Non-Vectorial Data* (LNVD07) was held at the 30th Annual German Conference on Artificial Intelligence in Osnabrück, Germany. The aim of the workshop was to bring together researchers from different application fields and disciplines like bioinformatics, text mining, and computer linguistics to discuss the cross-fertilization of the different disciplines and the integration of symbolic/qualitative and sub-symbolic/quantitative approaches, which is one of the central questions in cognitive science.

This proceedings volume contains seven contributions to the workshop, ranging from position papers to high quality full length articles. A number of contributions discuss kernel and distance methods, but there are also three articles focusing on more symbolic/deductive forms of learning like program synthesis, analogical reasoning, and learning in ontologies. The articles address a large variety of applications, with a focus on applications in the areas of computational linguistics and text mining.

The workshop as well as this volume would not have been possible without the help of numerous people. We would like to thank them all for their help and support, in particular all reviewers, the conference organizers, and Edwin Hancock (University of York, UK) and Stefan Wrobel (University of Bonn/FhG IAIS) for their presentations at the workshop. We would like to thank the Institute of Cognitive Science of the University of Osnabrück and the Universitätsgesellschaft Osnabrück for their financial support.

December 2007

Peter Geibel & Brijnesh J. Jain
Organizers

Table of Contents

Unsupervised Learning of Graph Languages using Kernels: New Results <i>Christophe Costa Florêncio</i>	p. 1
Theory of the Sample Mean of Structures <i>Brijnesh J. Jain, Klaus Obermayer</i>	p. 9
A Formal Text Representation Model Based on Lexical Chaining <i>Alexander Mehler, Ulli Waltinger, and Armin Wegner</i>	p. 17
Cost-based Ranking in Input Output Spaces <i>Ulf Brefeld</i>	p. 27
Information Extraction From Annotation Graphs Serving for the Adaptation of Ontologies <i>Uwe Mönnich, Jens Michaelis, Ekaterina Ovchinnikova, Tonio Wandmacher, and Kai-Uwe Kühnberger</i>	p. 31
Data-Driven Learning of Functions over Algebraic Datatypes from Input/Output- Examples <i>Emanuel Kitzelmann</i>	p. 36
Integrating Analogical and Inductive Learning at different Levels of Generaliza- tion <i>Helmar Gust, Ulf Krumnack, Kai-Uwe Kühnberger, Angela Schwering</i>	p. 46

Unsupervised learning of graph languages using kernels: New results

Christophe Costa Florêncio

Department of Computer Science, K.U. Leuven,
Celestijnenlaan 200A,
B-3001 Leuven, Belgium
Chris.CostaFlorencio@cs.kuleuven.be
<http://www.cs.kuleuven.be/>

Abstract. We propose to apply concepts and techniques from the field of Grammar Induction (GI) to graph mining and classification. From the perspective of GI, sets of graphs are just formal languages, and learning to classify graphs is equivalent to learning grammars for such languages. Our aim here is to formulate kernels that are close to being injective. Full injectivity is unlikely to be achieved or even approximated by an efficient graph kernel, since this problem is known to be as hard as deciding isomorphism. However, non-injective kernels can still be useful. One way of viewing the planar languages corresponding to these kernels is as graph languages defined by the closure under some graph transformations of a core language. The question then is, given a kernel, exactly what do these transformations look like?

We give some examples of such transformations for a few different graph kernels. These kernels are progressively ‘more injective’, but for each one an explicit counterexample to injectivity is given.

Key words: graph mining, grammar induction, graph kernels, planar languages, identifiability in the limit

1 Graph mining

Graphs offer a natural way of representing complex data objects. Thus, knowledge discovery with complex data requires learning algorithms that can deal with graphs. Up until now, most work on learning graphs consisted of extending well-known approaches from the field of machine learning to deal with more complex, structured data.

For both technical and conceptual reasons, borrowing techniques from Grammar Induction (GI) seems an obvious step. In GI, learning a classification task is approached as learning a representation for a language, this representation can then be used to decide whether or not previously unseen data is in the target language. Such representations usually take the form of automata, rule-based grammars or rewriting systems, and these can be used to represent graph languages as well as string or tree languages. Some attempts in this direction have been made ([LS98,DHO02], for example), but much work still needs to be done.

A novel approach to GI, called planar languages, makes use of linear equalities over a kernel feature space as representation. These inequalities define a hyperplane within the total feature space, and such a structure can easily be learned. This way, inherently learnable language classes can easily be defined. Given the amount of existing literature on graph kernels, one would expect some expressive, well-understood kernels to be already available that, with minor adjustments, can be used as the basis for the definition of rich learnable classes of graph languages.

2 Kernels and planar languages

The notion of *planar languages* was first defined in [CCFW06,CCFWS06]. Planar languages consist of sets of strings that correspond to data points that form a hyperplane in a feature space defined by a string kernel. Kernels are normally used in conjunction with an SVM or other classifier to perform supervised learning tasks. In the context of planar languages, the learning algorithm finds the lowest dimensional space spanned by the images of (exclusively) positive examples in feature space, so planar languages can be learned in an unsupervised manner.

It has been shown that all such language classes are learnable in the sense of being identifiable in the limit, with polynomial update time and with just a very small dataset. Furthermore, the learner can work under rationality constraints on its behaviour such as strong monotonicity.

3 Graphs

We define graphs as follows:

Definition 31 A graph $G = \langle V, E \rangle$ consists of a set of n vertices $V = \{v_1, \dots, v_n\}$ and a set of edges $E \subseteq V \times V$.

The in-degree and out-degree of a vertex are the number of edges leading away from it, and the number of edges incident on it, respectively.

We define the product graph, a useful concept for dealing with synchronous walks on two graphs:

Definition 32 Let graphs $G = \langle V, E \rangle$ and $G' = \langle V', E' \rangle$ be given. A (direct) product graph $G_{\times} = \langle V_{\times}, E_{\times} \rangle$ is a graph with $|V| \cdot |V'|$ vertices, where each such vertex represents a pair of vertices from G and G' , respectively. An edge $e \in E_{\times}$ connects two vertices in G_{\times} just if the corresponding vertices in G and G' are adjacent in both these graphs.

4 Graph kernels

In the context of planar languages, injectivity is a desirable property for kernels. In the case of string languages this is feasible, the gap-weighted string kernel for

example has been shown to be injective, for certain values of the decay value ([CWew]).

Unfortunately, in the case of graphs injectivity is not feasible. In [GFW03] it was shown that computing an injective graph kernel is at least as hard as deciding whether two graphs are isomorphic, a problem for which no polynomial time algorithm is known.¹ This result has been strengthened in [RG03], where it was shown that even approximating such a kernel is hard.

However, non-injective kernels are still useful. In the context of learning planar languages, such a kernel can be regarded as taking care of part of the generalization task. Another way of viewing this is that graph planar languages are defined as the closure of a core language under certain graph transformations. The question then is, given a kernel, exactly what generalizations are made by the kernel, i.e., what do these transformations look like, exactly? We will give some examples of such transformations in the next section.

4.1 The random walk graph kernel

In [BK07] a graph kernel is defined that is based on counting the number of matching random walks in graphs.² Since this method does not limit itself to comparing local features, one might expect the bad performance that other graph kernels suffer from, typically $O(n^6)$ or worse. However, efficient and theoretically sound algorithms do exist. In [VBS06], a reduction to Sylvester equations yields algorithms with worst-case time-complexity of $O(n^3)$.

Given the weight matrix of the product graph W_\times , initial and stopping probability distributions p_\times and q_\times , and some discrete measure μ , this kernel is defined as

$$\kappa(G, G') = \sum_{k=0}^{\infty} \mu(k) q_\times^\top W_\times^k p_\times. \quad (1)$$

The function $\mu(k)$ is commonly defined as λ^k , with $\lambda > 0$. Given an appropriate value for λ , Equation 1 is well-defined.

Note that, even though the random walk graph kernel is associated with an infinite-dimensional feature space, this does not imply that hyperplanes defining planar languages have to have infinite rank. Their dimensionality is determined solely by the size of the basis underlying the affine equations, which is finite by definition.

¹ Deciding isomorphism between two graphs is not known to be in P, and the lowest known bound is moderately exponential. However, in practice this problem is often tractable, see for example [BBLT97] for an algorithm that is polynomial if one of the graphs is weakly compact.

² It would obviously be possible to define a kernel based on the number of possible paths. Such kernels would in some ways be easier to deal with, but unfortunately they are known to be at least NP-hard to compute.

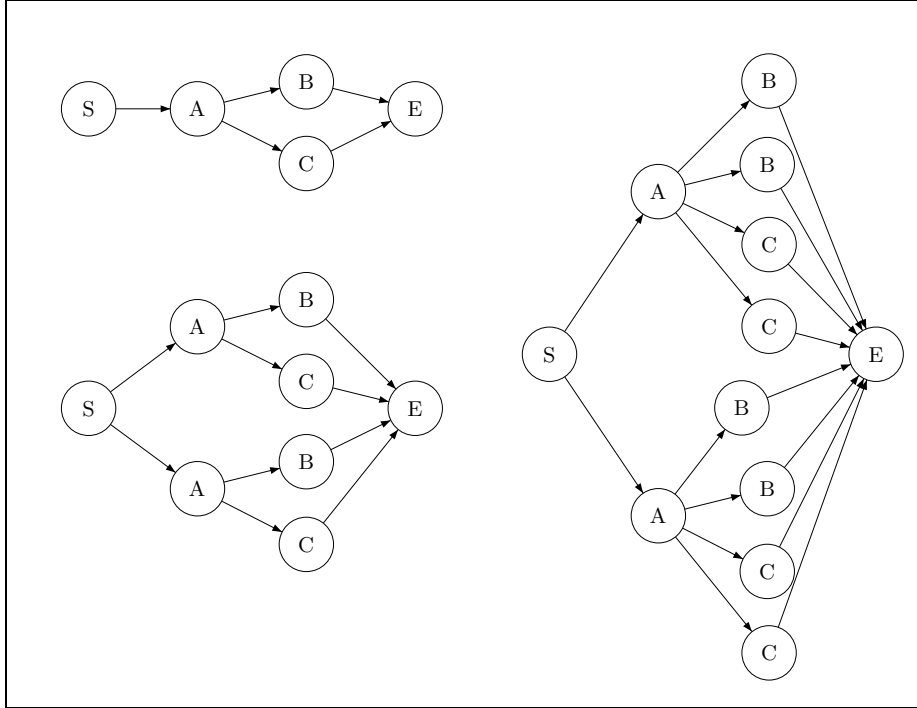


Fig. 1. Graphs that map to the same point in walks feature space.

4.2 The composite graph kernel

There is an asymmetry that is inherent in the concept of a product graph; absence of an edge between given vertices in both graphs G and G' is treated in the same way as absence of that edge in either G or G' . In some sense, potentially interesting information is ignored.

To counter this, [BK07] introduces a composite graph kernel based on the notion of *complement graph*. The complement graph $\overline{G} = \langle V, \overline{E} \rangle$ of a graph $G = \langle V, E \rangle$ has the same vertices as G , but just all the edges missing from G . The composite kernel is then defined as:

$$\kappa_{comp}(G, G') = \kappa(G, G') + \kappa(\overline{G}, \overline{G'}). \quad (2)$$

The practical usefulness of this kernel has been established in [BK07], it outperforms the random walk graph kernel on real-world data (PPI and gene expression data).

5 Graph planar languages

The walk kernel can be taken as the basis of a planar language. Since this kernel is known not to be injective, it is necessary to investigate to what set of graphs

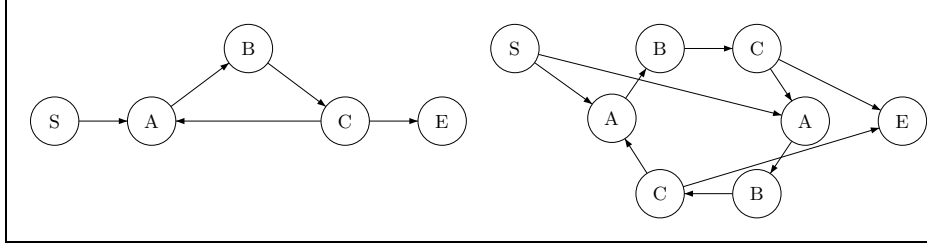


Fig. 2. Cyclic graphs that map to the same point in walks feature space.

a given point in feature space can map. We call any pair of data points mapping to the same point in feature space a *congruent pair*.

Consider the graphs in Figure 1, any two of these form a congruent pair for the walk kernel.³ Let G be a directed, connected graph with some node n_1 with in-degree 0 and some node n_2 with out-degree 0. Let G_n be the graph consisting of G and n copies of the (acyclic) subgraph between n_1 and n_2 , which are all connected to these nodes in the same way as in G . Then G and G_n map to the same point in the feature space spanned by the embedding underlying the walk kernel, for any n . It is easy to see that the exact same random walks exist for these graphs, and that they have the same likelihood. Figure 2 shows congruent pairs for the walk kernel where the graphs contain cycles. Any cyclic path in G can have a copy of itself inserted and the edges into and out of the cycle cloned, all at the appropriate places, the resulting graph maps to the same point in feature space as G . Note that the same examples hold for the composite kernel.

However, the composite kernel has greater expressive power; one can stipulate the non-existence of edges between nodes with certain labels by setting the values for all walks in which these labels appear next to each other to zero. This is not possible for the walk kernel, since the product graph does not contain information about absence of edges. Thus if graph $G = \langle V, E \rangle$ is in a walk-planar language, then so is any graph $G' = \langle V, E \cup E' \rangle$, $E' \neq \emptyset$ (note that from this it immediately follows that for the walk kernel the edges into a cycle need not be cloned to obtain a congruent pair). This is clearly an undesirable property for a graph kernel in almost any context.

It is possible to combine the walks kernel with the Parikh (count) kernel over vertex labels. This kernel cannot distinguish between the graphs in Figure 1, provided that vertices are added to the first two graphs so that all three have the same vertex label count.⁴

One obvious way to deal with this would be to use count kernels over both vertices and edges. The graphs in Figure 3 form a congruent pair for this kernel.

³ This example is a generalization of an example from [RG03] based on graphs made up of just 4 nodes.

⁴ Note that the first two graphs would then not be connected. It is presently unclear whether this kernel is injective for graph languages restricted to connected graphs.

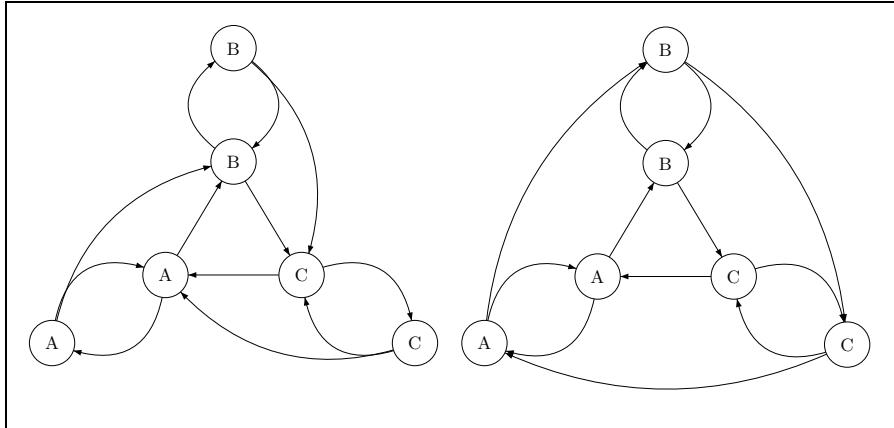


Fig. 3. Graphs that map to the same point in walks + vertex count + edge count feature space.

Intuitively, a kernel based on the neighbourhoods of vertices would be able to distinguish between these graphs. We propose to combine the walk kernel with a count kernel over triples of vertex label, in-degree and out-degree.

Given the mathematical properties of kernels, the sum of two distinct kernels always yields a kernel as well. This property can be very useful. In [CCFWS06], the gap-weighted+ kernel, a combination of the Parikh kernel (which simply counts occurrences of symbols) and gap-weighted kernel, was defined in order to obtain a kernel which is defined for strings of length 1 as well as for strings of length > 1 , yielding an injective kernel.

In the present context, we exploit this property with the purpose of making graph kernels ‘more injective’. That is, we present a series of kernels, each of these defined in such a way as to be able to distinguish between two graphs that for the previous kernel formed a congruent pair.

The resulting kernel is again not injective; Figure 4 shows a congruent pair for this kernel. However, in some intuitive, pre-theoretic sense, these graphs are ‘more similar’ than those in the previous figures. Therefore this kernel is expected to be more useful than the previous ones, although this obviously also depends on the application domain.

6 Conclusion

Planar languages offer a promising approach to learning rich classes of graph languages, even though the graph kernels they are based on are in all likelihood non-injective. For most kernels it should generally not be too hard to formulate graph transformations that exactly describe in what way the kernel generalizes, giving an explicit description of the structure of the graph planar languages it defines. Combining different graph kernels yields kernels that are closer to

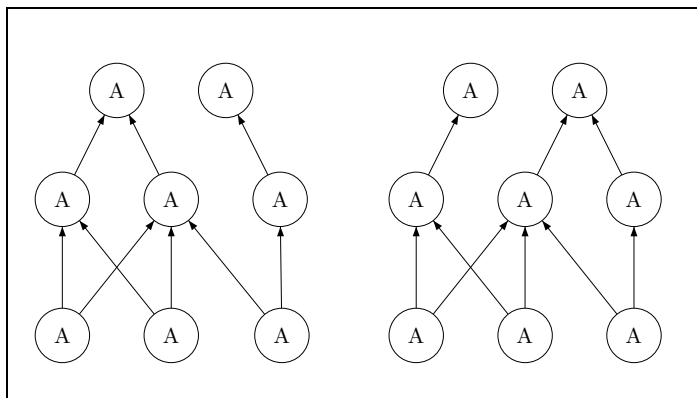


Fig. 4. Graphs that map to the same point in walks + vertex signature count feature space.

being injective, although full injectivity does not seem feasible. Depending on the particular application domain this non-injectivity, which implies generalization, may even be desirable (for example, two distinct graphs may represent isomers of one particular molecule, a non-injective kernel may map these to the same point in feature space).

It is expected that implementing efficient learning algorithms will pose few problems. However, earlier experiments with a (string) planar language learner [CCFW06] indicate this approach is noise-sensitive, so it would be interesting to see how a graph planar language learner performs on real-world data.

Although we only considered walk-based kernels here, there are others that are also good candidates for serving as a basis for graph planar languages, the tree-structured pattern kernel [RG03] or the cyclic pattern kernels from [HGW04], for example.

References

- [BBLT97] L. Babel, S. Baumann, M. Ludecke, and G. Tinhofer. STABCOL: Graph isomorphism testing based on the Weisfeiler-Leman algorithm. Preprint TUM-M9702, Institut für Mathematik, Technische Universität München, 1997. 33 pp.
- [BK07] Karsten M. Borgwardt and Hans-Peter Kriegel. Graph kernels for disease outcome prediction from protein-protein interaction networks. In *Proc. of Pacific Symposium on Biocomputing (PSB 2007), Maui, USA, 2007*.
- [CCFW06] Alexander Clark, Christophe Costa Florêncio, and Chris Watkins. Languages as hyperplanes: grammatical inference with string kernels. In *ECML, 17th European Conference on Machine Learning*. Springer-Verlag, 2006.
- [CCFWS06] Alexander Clark, Christophe Costa Florêncio, Chris Watkins, and Mariette Serayet. Planar languages and learnability. In *International Colloquium on Grammatical Inference (ICGI)*, Tokyo, 2006.

- [CWew] Alexander Clark and Chris Watkins. Some alternatives to Parikh matrices using string kernels. Under review.
- [DHO02] Shailesh P. Doshi, Fang Huang, and Tim Oates. Inferring the structure of graph grammars from data. In *International Conference on Knowledge Based Computer Systems*, 2002.
- [GFW03] Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop*, pages 129–143. Springer-Verlag, August 2003.
- [HGW04] Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. Cyclic pattern kernels for predictive graph mining. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 158–167, New York, NY, USA, 2004. ACM Press.
- [LS98] Damián López and José M. Sempere. Handwritten digit recognition through inferring graph grammars. A first approach. In *SSPR '98/SPR '98: Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition*, pages 483–491, London, UK, 1998. Springer-Verlag.
- [RG03] Jan Ramon and Thomas Gärtner. Expressivity versus efficiency of graph kernels. In T. Washio and L. De Raedt, editors, *Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences*, pages 65–74, 2003.
- [VBS06] S. V. N. Vishwanathan, K. Borgwardt, and N. N. Schraudolph. Fast computation of graph kernels. In *Advances in Neural Information Processing Systems 19 (NIPS 2006)*. MIT Press, Cambridge, MA, 2006.

Theory of the Sample Mean of Structures

Brijnesh J. Jain and Klaus Obermayer

Berlin University of Technology, Faculty IV, Germany, bj@cs.tu-berlin.de

Abstract. Graphs often occur as "natural" representations of structured objects in different application areas of machine learning. To adopt methods like central clustering or principal component analysis for graphs, an understanding of the structural version of the sample mean is imperative. We present an analytic and geometric view of the structural mean.

1 Introduction

In many applications in pattern recognition, it is common practice to represent data by feature vectors living in an Euclidean space, because the Euclidean space provides powerful mathematical tools for data analysis, which are usually not available for other representations. One standard tool is the sample mean, which is the most important and widely used measure of central tendency in statistics. In pattern recognition, this measure is an integral part of different data analysis techniques such as central clustering algorithms, mixture models, and dimension reduction methods.

But often, the data points of a sample we want to summarize by a measure of central tendency have no natural representation as feature vectors and are more naturally represented in terms of graphs. In contrast to feature vectors, the concept of a sample mean of graphs is hardly investigated, although a number of central clustering algorithms for graphs have been devised [1]-[3].

The focus of this paper is on extending the concept of sample mean to graphs. The proposed formulation is based on the median graph [4]. A median graph is a graph with minimal sum of distances from the given sample graphs. This formulation corresponds to the formulation of the standard sample mean as an optimization problem. The median graph is a general and widely applicable measure of central tendency, because it makes no assumptions on the vertex and edge attributes and the underlying graph-edit distance measure. But this generality makes a theoretical analysis and a deeper insight into the nature of the concept more difficult. What we want is a measure of central tendency that summarizes a sample of graphs by recording the relative frequencies of common structural overlaps.

This paper aims at establishing a theoretical basis of the sample mean of graphs. In doing so we try to span a bridge from structural to statistical pattern recognition. The chosen approach is geometrically inspired and yields properties and characterizations that find — to a certain extent — their counterpart in the standard sample mean. The proposed framework yields a conceptually simple

plug-in mechanism to extend central clustering algorithms like k-means or competitive learning to graphs. Although the proposed approach is general enough to apply to finite structures other than graphs, we will for sake of concreteness restrict our attention exclusively to this domain. Due to lack of space, proofs have been omitted.

2 The Basic Approach

The purpose of this section is to give a nontechnical overview and a motivation of the proposed approach.

A weighted graph is a triple $X = (V, E, w)$ consisting of a finite nonempty set V of *vertices*, a set $E \subseteq V \times V$ of *edges*, and a *weight function* $w : V \times V \rightarrow \mathbb{R}$ such that each edge has nonzero weight and each non-edge has weight zero. The weight $w(i, i)$ of vertex i can be any value from \mathbb{R} . A weighted graph X of order $|V| = n$ is completely specified by its *weight matrix* $\mathbf{X} = (x_{ij})$ with elements $x_{ij} = w(i, j)$ for all $1 \leq i, j \leq n$. By \mathcal{G} we denote the set of weighted graphs.

Suppose that $\mathcal{D}_{\mathcal{T}} = (X_1, \dots, X_k)$ is a sample of k not necessarily distinct graphs from \mathcal{G} . The standard method $M = (X_1 + \dots + X_k) / k$ for determine the sample mean fails, because a well-defined addition of graphs that meets our requirements is unknown. Following [4], we adopt the optimization formulation of the standard sample mean. For vectors, the sample mean minimizes the sum of squared Euclidean distances from the data points. In line with this formulation, we define a sample mean of $\mathcal{D}_{\mathcal{T}}$ as a global minimum of the cost function

$$F : \mathcal{G} \rightarrow \mathbb{R}, \quad F(X) = \sum_{i=1}^k D(X, X_i)^2, \quad (1)$$

where D is a distance function on \mathcal{G} .

In principle, we can use any distance function D . Here, we focus on geometric distance functions that are related to the Euclidean metric, because the Euclidean metric is the underlying metric of the vectorial mean. The vectorial mean in turn provides a link to deep results in probability theory and is the foundation for a rich repository of analytical tools in pattern recognition. To access at least parts of these results, it seems to be reasonable to relate the distance function D in (1) to the Euclidean metric. This restriction is acceptable from an application point of view, because geometric distance functions on graphs and their related similarity functions are a common choice of proximity measure in a number of different applications [5-6].

Geometric distance functions D are typically defined as the maximum of a set of Euclidean distances. This definition implies that (1) the cost function F is neither differentiable nor convex; (2) the sample mean of graphs is not unique; and (3) determining a sample mean of graphs is NP-complete, because evaluation of D is NP-complete. Thus, we are faced with an intractable combinatorial optimization problem, where, at a first glance, a solution has to be found from an uncountable infinite set. In addition, multiple global minima of the cost function F complicates a characterization of a structural mean.

To cope with these difficulties, we view graphs as equivalence classes of vectors via their weight matrices, where the elements of the same equivalence class are different vector representations of the same graph. The resulting quotient set (the set of equivalence classes) leads to the more abstract notion of \mathcal{T} -space. Formally, a \mathcal{T} -space $\mathcal{X}_{\mathcal{T}}$ over a vector space \mathcal{X} is a quotient set of \mathcal{X} , where the equivalence classes are the orbits of the group action of a transformation group \mathcal{T} on \mathcal{X} . The theory of \mathcal{T} -spaces generalizes the vector space concept to cope with combinatorial structures and aims at retaining the geometrical and algebraic properties of a vector space to a certain extent.

The \mathcal{T} -space concept clears the way to approach the structural version of the sample mean in a principled way. We present a geometric characterization of a structural mean that is closely related to the standard formulation of the vectorial mean. This characterization has important implications: (i) it shows that the solutions can come from a finite discrete set; and (ii) it shows that any sample mean is a well-defined weighted graph. The second important result we show is that the cost function F is locally Lipschitz and therefore differentiable almost everywhere. Hence, we can exploit generalized gradient information to minimize F .

3 \mathcal{T} -Spaces

In this section, we develop the theory of \mathcal{T} -spaces that allows us to formally adopt geometrical and analytical concepts for graphs. This framework provides the theoretical foundation for deriving results on the sample mean of graphs.

The first step to relate graphs to the Euclidean space is to align them to a fixed dimension. Let $\mathcal{G}[n]$ be the set of weighted graphs of order n . We can regard any weighted graph X of order $m < n$ as a graph of order n by adding $p = n - m$ isolated vertices. The weighted adjacency matrix of the aligned graph X' is then of the form

$$\mathbf{X}' = \begin{pmatrix} \mathbf{X} & \mathbf{0}_{m,p} \\ \mathbf{0}_{p,m} & \mathbf{0}_{p,p} \end{pmatrix},$$

where \mathbf{X} is the weight matrix of X , and $\mathbf{0}_{m,p}$, $\mathbf{0}_{p,m}$, $\mathbf{0}_{p,p}$ are padding zero matrices. Using alignment, we can regard $\mathcal{G}[n]$ as the set of weighted graphs of bounded order n . Note that specifying an order n and aligning smaller graphs to graphs of order n are purely technical assumptions to simplify mathematics, which can be safely ignored in a practical setting. We only demand that the graphs are bounded.

The positions of the diagonal elements of \mathbf{X} determine an ordering of the vertices. Conversely, different orderings of the vertices may result in different matrices. Since we are interested in the structure of a graph, the ordering of its vertices does not really matter. Therefore, we consider two matrices \mathbf{X} and \mathbf{X}' as being equivalent, denoted by $\mathbf{X} \sim \mathbf{X}'$, if they can be obtained from one another by reordering the vertices. Mathematically, the equivalence relation can be written as

$$\mathbf{X} \sim \mathbf{X}' \Leftrightarrow \exists \mathbf{P} \in \mathcal{P} : \mathbf{P}^{\top} \mathbf{X} \mathbf{P} = \mathbf{X}', \quad (2)$$

where \mathcal{P} denotes the set of all $(n \times n)$ -permutation matrices. The equivalence class $[\mathbf{X}]$ of \mathbf{X} describes the *abstract weighted graph* that is independent of the particular numbering of the vertices. By $\mathcal{G}[n]/\mathcal{P}$ we denote the set of all abstract weighted graphs of bounded order n . The equivalence relation \sim gives rise to a more abstract and convenient approach of $\mathcal{G}[n]$ and $\mathcal{G}[n]/\mathcal{P}$ that allows to formally adopt statistical, analytical and geometrical concepts. The suggested approach are \mathcal{T} -spaces that will be introduced in the sequel.

Let $\mathcal{X} = \mathbb{R}^N$ be the N -dimensional Euclidean vector space, and let \mathcal{T} be a subgroup of the group \mathcal{P} of all $(N \times N)$ -permutation matrices. Then the binary operation

$$\cdot : \mathcal{T} \times \mathcal{X} \rightarrow \mathcal{X}, \quad (T, \mathbf{x}) \mapsto T\mathbf{x}$$

is a group action of \mathcal{T} on \mathcal{X} . For $\mathbf{x} \in \mathcal{X}$, the *orbit* of \mathbf{x} is denoted by $[\mathbf{x}]_{\mathcal{T}} = \{T\mathbf{x} : T \in \mathcal{T}\}$. If no misunderstanding can occur, we write $[\mathbf{x}]$ instead of $[\mathbf{x}]_{\mathcal{T}}$. A \mathcal{T} -space over \mathcal{X} is the orbit space $\mathcal{X}_{\mathcal{T}} = \mathcal{X}/\mathcal{T}$ of all orbits of $\mathbf{x} \in \mathcal{X}$ under the action of \mathcal{T} . We call \mathcal{X} the *representation space* of $\mathcal{X}_{\mathcal{T}}$. By $\mu : \mathcal{X} \rightarrow \mathcal{X}_{\mathcal{T}}$ we denote the *membership function* that sends vector representations to the structure they describe. We use capital letters X, Y, Z, \dots to denote the elements of $\mathcal{X}_{\mathcal{T}}$. Suppose that $X = \mu(\mathbf{x})$ for some $\mathbf{x} \in \mathcal{X}$. By abuse of notation, we sometimes identify X with $[\mathbf{x}]$ and write $\mathbf{x} \in X$ instead of $\mathbf{x} \in [\mathbf{x}]$.

Suppose that $N = n^2$. Then any matrix \mathbf{X} representing a weighted graph $X \in \mathcal{G}[n]$ can be regarded as a vector \mathbf{x} by stacking the columns of \mathbf{X} . Let \mathcal{T} be the subgroup of all $(N \times N)$ -permutation matrices that corresponds to the set of all $(n \times n)$ -permutations matrices for renumbering the vertices of a graph of order n . Obviously, we have a relaxation in the sense that $\mathcal{G}[n] \subseteq \mathcal{X}$ and $\mathcal{G}[n]/\mathcal{P} \subseteq \mathcal{X}_{\mathcal{T}}$ such that μ restricted on $\mathcal{G}[n]$ sends vector representations to the graphs they represent. Note that there are structures in $\mathcal{X}_{\mathcal{T}}$ that are not well-defined graphs. To transfer geometrical properties of the Euclidean space \mathcal{X} to the \mathcal{T} -space $\mathcal{X}_{\mathcal{T}}$, we consider similarity and distance functions on $\mathcal{X}_{\mathcal{T}}$ of the following type. Let $f : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ be a symmetric function satisfying $f(\mathbf{x}, \mathbf{y}) = f(\mathbf{y}, \mathbf{x})$ for all $\mathbf{x}, \mathbf{y} \in \mathcal{X}$. Then f induces symmetric functions

$$\begin{aligned} F^* : \mathcal{X}_{\mathcal{T}} \times \mathcal{X}_{\mathcal{T}} &\rightarrow \mathbb{R}, & (X, Y) &\mapsto \max \{f(\mathbf{x}, \mathbf{y}) : \mathbf{x} \in X, \mathbf{y} \in Y\}, \\ F_* : \mathcal{X}_{\mathcal{T}} \times \mathcal{X}_{\mathcal{T}} &\rightarrow \mathbb{R}, & (X, Y) &\mapsto \min \{f(\mathbf{x}, \mathbf{y}) : \mathbf{x} \in X, \mathbf{y} \in Y\}. \end{aligned}$$

Since \mathcal{T} is finite, the orbits $[\mathbf{x}]$ of \mathbf{x} are finite. Hence, F^* and F_* assume an extremal value. We call F^* *maximizer* and F_* *minimizer* of f on $\mathcal{X}_{\mathcal{T}} \times \mathcal{X}_{\mathcal{T}}$. Let F be either a maximizer or minimizer of f . The *support* of F at $(X, Y) \in \mathcal{X}_{\mathcal{T}} \times \mathcal{X}_{\mathcal{T}}$ is the set defined by

$$\text{supp } F(X, Y) = \{(\mathbf{x}, \mathbf{y}) \in X \times Y : F(X, Y) = f(\mathbf{x}, \mathbf{y})\}.$$

The class of similarity functions on $\mathcal{X}_{\mathcal{T}}$ we consider are maximizers of inner products $\langle \cdot, \cdot \rangle$ on \mathcal{X} . The *inner \mathcal{T} -product* induced by $\langle \cdot, \cdot \rangle$ is defined by

$$\langle \cdot, \cdot \rangle^* : \mathcal{X}_{\mathcal{T}} \times \mathcal{X}_{\mathcal{T}} \rightarrow \mathbb{R}, \quad (X, Y) \mapsto \max \{\langle \mathbf{x}, \mathbf{y} \rangle : \mathbf{x} \in X, \mathbf{y} \in Y\}.$$

The inner \mathcal{T} -product is *not* an inner product, because the maximum-operator in the definition of $\langle \cdot, \cdot \rangle^*$ does not preserve the bilinearity property of an inner product. But as we will see shortly, the inner \mathcal{T} -product has the same geometrical properties as the standard inner product.

Any inner product space \mathcal{X} is a normed space with norm $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$ and a metric space with metric $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$. The norm $\|\cdot\|$ and the metric d on \mathcal{X} give rise to minimizers $\|\cdot\|_*$ of $\|\cdot\|$ and D_* of d on $\mathcal{X}_{\mathcal{T}}$. Since elements from \mathcal{T} preserve lengths and angles, we have $\|T\mathbf{x}\| = \|\mathbf{x}\|$ for all $T \in \mathcal{T}$. Hence, a \mathcal{T} -norm $\|X\|_*$ is independent from the choice of vector representation. We show that a \mathcal{T} -norm is related to an inner \mathcal{T} -product in the same way as a norm to an inner product.

Proposition 1. *Let $\mathcal{X}_{\mathcal{T}}$ be a \mathcal{T} -space over $(\mathcal{X}, \langle \cdot, \cdot \rangle)$, and let $X \in \mathcal{X}_{\mathcal{T}}$. Then*

1. $\langle X, X \rangle^* = \langle \mathbf{x}, \mathbf{x} \rangle$ for all $\mathbf{x} \in X$.
2. $\|X\|_* = \sqrt{\langle X, X \rangle^*}$.

The inner \mathcal{T} -product together with its \mathcal{T} -norm satisfies a structural version of the Cauchy-Schwarz inequality.

Theorem 1. *Let $\mathcal{X}_{\mathcal{T}}$ be a \mathcal{T} -space over an Euclidean space $(\mathcal{X}, \langle \cdot, \cdot \rangle)$, and let $X, Y \in \mathcal{X}_{\mathcal{T}}$. Then*

$$|\langle X, Y \rangle^*| \leq \|X\|_* \|Y\|_*.$$

Using Theorem 1, we can introduce the notion of angle between structures in the usual way. This shows that the inner \mathcal{T} -product has the same geometrical properties as the standard inner product.

The next result states that the minimizer D_* of the Euclidean metric $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$ is also a metric.

Theorem 2. *Let \mathcal{X} be an Euclidean space with Euclidean metric d . Then the minimizer D_* of d is a metric on the \mathcal{T} -space $\mathcal{X}_{\mathcal{T}}$.*

We call the minimizer D_* of a Euclidean metric d on \mathcal{X} the \mathcal{T} -metric induced by d . The \mathcal{T} -metric D_* can also be expressed in terms of $\langle \cdot, \cdot \rangle^*$.

Proposition 2. *We have $D_*(X, Y)^2 = \|X\|_*^2 - 2\langle X, Y \rangle^* + \|Y\|_*^2$.*

A \mathcal{T} -function is a function of the form $F : \mathcal{X}_{\mathcal{T}} \rightarrow \mathbb{R}$, where $\mathcal{X}_{\mathcal{T}}$ is a \mathcal{T} -space over \mathcal{X} . Instead of considering the \mathcal{T} -function F , it is often more convenient to consider its *representation function*

$$f : \mathcal{X} \rightarrow \mathbb{R}, \quad \mathbf{x} \mapsto F \circ \mu(\mathbf{x}),$$

which is invariant under transformations from elements of \mathcal{T} .

4 The sample mean of graphs

Throughout this section we assume that \mathcal{X} is an Euclidean space with inner product $\langle \cdot, \cdot \rangle$, and $\mathcal{X}_{\mathcal{T}}$ is a \mathcal{T} -space over \mathcal{X} . Let $\mathcal{D}_{\mathcal{T}} = (X_1, \dots, X_k)$ be a data sample of k (not necessarily distinct) elements from $\mathcal{X}_{\mathcal{T}}$. A *sample mean* of $\mathcal{D}_{\mathcal{T}}$ is any solution of the following optimization problem

$$(P_{\mathcal{T}}) \quad \begin{array}{ll} \text{minimize} & F(X) = \sum_{i=1}^k D_*(X, X_i)^2, \\ \text{subject to} & X \in \mathcal{X}_{\mathcal{T}} \end{array}$$

where D_* is the \mathcal{T} -metric induced by the Euclidean metric d on \mathcal{X} . The *representation sample* of the sample $\mathcal{D}_{\mathcal{T}}$ is defined by the set $\mathcal{D} = X_1 \times \dots \times X_k = \mu^{-1}(X_1) \times \dots \times \mu^{-1}(X_k)$. Each element of \mathcal{D} is a k -tuple $(\mathbf{x}_1, \dots, \mathbf{x}_k)$, where the components \mathbf{x}_i are vector representations of X_i . We identify the elements of the representation sample \mathcal{D} with matrices. A *matrix representation* of $\mathcal{D}_{\mathcal{T}}$ is a $(n \times k)$ -matrix $\mathbf{X} = (\mathbf{x}_1 \cdots \mathbf{x}_k)$, where the columns of \mathbf{X} form a k -tuple from \mathcal{D} . Suppose that $\mathbf{X} = (\mathbf{x}_1 \cdots \mathbf{x}_k) \in \mathcal{D}$ is an arbitrary matrix representation of $\mathcal{D}_{\mathcal{T}}$. We can rewrite $(P_{\mathcal{T}})$ to the equivalent form

$$(P) \quad \begin{array}{ll} \text{minimize} & f(\mathbf{x}) = \sum_{i=1}^k \min_{\mathbf{x}_i \in X_i} \|\mathbf{x} - \mathbf{x}_i\|^2 \\ \text{subject to} & \mathbf{x} \in \mathcal{X} \end{array}$$

where $f = F \circ \mu$ is a representation function of F . Note that the solutions of problem (P) are independent from the particular choice of $\mathbf{x} \in X$. The following result shows that problem $(P_{\mathcal{T}})$ has a solution, which is in general not unique.

Proposition 3. *Problem $(P_{\mathcal{T}})$ has a solution.*

4.1 Characterization of the sample mean

Theorem 3 shows that a sample mean of $\mathcal{D}_{\mathcal{T}}$ is of similar form as the sample mean of a set of vectors.

Theorem 3. *Let $\mathcal{D}_{\mathcal{T}} = (X_1, \dots, X_k)$ be a sample of k structures from $\mathcal{X}_{\mathcal{T}}$. Then any vector representation \mathbf{m} of a sample mean $M \in \mathcal{X}_{\mathcal{T}}$ of $\mathcal{D}_{\mathcal{T}}$ is of the form*

$$\mathbf{m} = \frac{1}{k} \sum_{i=1}^k \mathbf{x}_i,$$

where $\mathbf{X} = (\mathbf{x}_1 \cdots \mathbf{x}_k)$ is a matrix representation of $\mathcal{D}_{\mathcal{T}}$ satisfying $(\mathbf{m}, \mathbf{x}_i) \in \text{supp } D_*(M, X_i)$ for all $i \in \{1, \dots, k\}$.

We call a matrix representation \mathbf{X} of $\mathcal{D}_{\mathcal{T}}$ *conform* if the sample mean of the columns of \mathbf{X} is a solution to problem (P) , i.e. a vector representation of a sample mean of $\mathcal{D}_{\mathcal{T}}$.

Theorem 3 has three implications. First, any sample mean of well-defined graphs from the subset $\mathcal{G}[n]/\mathcal{P}$ of $\mathcal{X}_{\mathcal{T}}$ is a well-defined weighted graph from $\mathcal{G}[n]/\mathcal{P}$.

Second, problem $(P_{\mathcal{T}})$ is a discrete combinatorial optimization problem. Any solution \mathbf{m} of the equivalent problem (P) is a sample mean of the columns of a conform matrix representation $\mathbf{X} \in \mathcal{D} = X_1 \times \cdots \times X_k$. Since the group \mathcal{T} is finite, the set \mathcal{D} is also finite. Hence, solving problem $(P_{\mathcal{T}})$ reduces to finding a conform matrix representation \mathbf{X} from a finite set \mathcal{D} . Third, Theorem 3 includes equivalent geometric characterizations of a sample mean. The remainder of this subsection is devoted to this issue.

The common tool to derive geometric characterizations of the sample mean is the Gram matrix. Let $\mathbf{X} = (\mathbf{x}_1 \cdots \mathbf{x}_k)$ be a matrix representation of a sample $\mathcal{D}_{\mathcal{T}} = (X_1, \dots, X_k)$ of structures. The *Gram matrix* of \mathbf{X} is defined by

$$\mathbf{G} = \mathbf{G}_{\mathbf{X}} = \mathbf{X}^T \mathbf{A} \mathbf{X},$$

where \mathbf{A} denotes the symmetric matrix representing the inner product $\langle \cdot, \cdot \rangle$ of \mathcal{X} . Thus, the elements of $\mathbf{G} = (g_{ij})$ are of the form $g_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$. The *Gram sum* of \mathbf{X} is defined by

$$\Gamma(\mathbf{X}) = \sum_{i=1}^n \sum_{j=1}^n g_{ij}.$$

Theorem 4. *Let $\mathcal{D}_{\mathcal{T}} = (X_1, \dots, X_k)$ be a sample of structures from $\mathcal{X}_{\mathcal{T}}$. Then \mathbf{X} is a conform matrix representation of $\mathcal{D}_{\mathcal{T}}$ if, and only if, $\Gamma(\mathbf{X}) \geq \Gamma(\mathbf{X}')$ for all $\mathbf{X}' \in \mathcal{D}$.*

Theorem 4 states that a structure M is a sample mean of $\mathcal{D}_{\mathcal{T}} = (X_1, \dots, X_k)$ if, and only if, it is represented by the sample mean of vector representations $(\mathbf{x}_1, \dots, \mathbf{x}_k) \in \mathcal{D}$ with maximal Gram sum. Since the Gram sum contains information about pairwise similarities, distances, correlations, and angles, we can derive the following equivalent geometrical statements: A sample mean of graphs X_1, \dots, X_k is represented by the sample mean of those vector representations of X_i that have (1) maximal average pairwise Gram similarity, (2) minimal average pairwise Euclidean distance, (3) maximal average correlation coefficient, and (4) minimal average pairwise angle.

4.2 Subgradient methods for computing a sample mean

The cost function f of problem (P) is non-differentiable. But it can be shown that the function f is locally Lipschitz and therefore non-differentiable on a set of Lebesgue measure zero by Rademacher's Theorem. In addition, the gradient at differentiable points is invariant under permutation. Hence, the gradient of a smooth function on structures is a well-defined structure pointing in direction of steepest ascent. To minimize locally Lipschitz functions, the field of nonsmooth optimization offers a number of techniques [7]. The simplest and probably also the most used method for non-differentiable optimization are subgradient methods. Their basic idea is to replace gradients by subgradients at non-differentiable points.

4.3 Sufficient Uniqueness Condition

Theorem 5. *The sample mean of a finite sample of weighted graphs is well-defined with probability one.*

The proof of this result is based on sufficient conditions derived from the geometry of polyhedral cones. Unfortunately, the sufficient conditions are rarely satisfied in a practical setting. The benefit, however, is that the techniques to prove Theorem 5 set the scene to derive sharper sufficient conditions more appropriate for application problems. A probabilistic statement is still an open problem.

5 Conclusion

Based on the theory of \mathcal{T} -spaces, we have presented a principled approach to characterize the sample mean of graphs and derived a subgradient method for its approximation. In the workshop, we present first experiment to show how the framework can be plugged in into k-means and competitive learning to cluster protein structures.

References

- [1] S. Gold, A. Rangarajan, and S. Mjolsness. Learning with preknowledge: Clustering with point and graph matching distance measures. *Neural Computation*, 8(4):787–804, 1996.
- [2] S. Günter and H. Bunke. Self-organizing map for clustering in the graph domain. *Pattern Recognition Letters*, 23:401–417, 2002.
- [3] M.A. Lozano and F. Escolano. Protein classification by matching and clustering surface graphs. *Pattern Recognition*, 39(4):539–551, 2006.
- [4] X. Jiang, X., A. Munger, and H. Bunke. On Median Graphs: Properties, Algorithms, and Applications. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 23(10):1144–1151, 2001.
- [5] S. Gold and A. Rangarajan. Graduated Assignment Algorithm for Graph Matching. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 18:377–388, 1996.
- [6] L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology*, 233(1):123–38, 1993.
- [7] M. Mäkelä and P. Neittaanmäki. Nonsmooth Optimization: Analysis and Algorithms with Applications to Optimal Control. *World Scientific*, 1992.

A Formal Text Representation Model Based on Lexical Chaining

Alexander Mehler, Ulli Waltinger, and Armin Wegner

Bielefeld University

{Alexander.Mehler,Ulli.Waltinger,Armin.Wegner}@uni-bielefeld.de

Abstract. This paper presents a formal text representation model as an alternative to the vector space model. It combines a tree-like model with graph-inducing lexical relations. The paper aims at formalizing two yet unrelated approaches, i.e. lexical chaining [3] and quantitative structure analysis [9], in order to combine content *and* structure modeling.

1 Introduction

The *bag of features* approach has been seminal for measuring the similarities of natural language texts. Its most prominent exponent is the vector space model [14]. It has been used in various fields of computational linguistics such as, e.g., disambiguation [16], text categorization [6] and topic tracking [1]. Further, it has been extended in terms of latent semantic analysis [7] and related descendants in order to measure *indirect* relations of texts which are no longer judged to be similar by sharing a subset of relevant words, but by containing similarly used ones.

These and related models all assume that texts are adequately represented as weighted vectors of mainly lexical features. Obviously, this model disregards text structure though there are extensions which additionally take non-lexical features into account [17]. Further, the bag-of-features approach primarily explores lexical knowledge from input texts or some reference corpora, but disregards terminological ontologies such as social tagging systems [19] which – due to the fact that they result from “crowdsourcing” – provide large scale access to various topic areas which are continuously enriched by their users [11].

This paper presents an alternative model which does both mapping some part of text structure and utilizing a terminological ontology as a source of measuring direct and indirect similarities of signs. Its starting point is the independently confirmed success of a purely structure-oriented approach to text categorization, called *quantitative structure analysis* (QSA) [9], and its content-related counterpart in the form of *lexical chaining* [3]: Whereas QSA shows that a large number of genre-related text categories can be learnt by solely exploring the structure of input texts, lexical chaining proofs its potential in content-related categorization [10]. In this paper, we present a formal model of the textual input of these two approaches. Our starting point is the lack of an integrated graph-theoretical text model as input to QSA *and* lexical chaining. The paper aims at filling this gap by contributing an integrated structure- *and* content-oriented text model.

2 A Formal Model of Lexical Chains

This section presents a formal model of lexical chaining which extends the standard approach by integrating a graph-theoretical text model. Our starting point is the notion of the *Logical Document Structure* (LDS) [12] and an *ordered hierarchy of content objects* [13] as exemplified by the inclusion hierarchy of a text based on its nested section, paragraph and sentence structure. As this structure is extended by graph inducing links as, e.g., anaphoric relations between different text spans, we extend the notion of LDS in terms of a generalized tree [8]. Note that we aim at a formal text representation model as an alternative to the feature vector model. Thus, we start with an abstract definition of textual units which is later on instantiated by natural language texts.

Definition 1. (Preliminaries) Let $A = \{a_1, \dots, a_n\}$, $B = \{b_1, \dots, b_m\}$ be two sets and $t: B \rightarrow A$ a total function where $t(b) = a \in A$ is called the *type of form* $b \in B$. We define a relation $\models \subseteq B \times A$ with $(b, a) \in \models$, i.e. $b \models a$, iff $t(b) = a$. Next, let $\langle B^*, \circ, \epsilon \rangle$ be the free monoid over B based on the concatenation function \circ . Any $x = b_{i_1} \circ \dots \circ b_{i_l} \in B^*$, $b_{i_j} \neq \epsilon, j \in \{1, \dots, l\}$, is called a *text string* of length l . In the present paper we identify tokens by their text position such that b_{i_j} is the j th token in x . This is done by the function $x: \{1, \dots, l\} \rightarrow \{b_{i_1}, \dots, b_{i_l}\}$ with $x(j) = b_{i_j}$. Note that for two tokens $n, m \in \{1, \dots, l\}$, $n \neq m$, it may hold that $x(n) = x(m)$. A set $C = \{x_1, \dots, x_k\}$ of text strings is called a *text string corpus*. For any $1 \leq m \leq n \leq l$, $b_{i_m} \circ \dots \circ b_{i_n}$ is called a *text substring* of $x = b_{i_1} \circ \dots \circ b_{i_l} \in B^*$. The *set of all text substrings* of x is denoted by $[x]$.

A text string is the formal counterpart of a tokenized text in the stage before the segmentation of its LDS. Analogously, a text string corpus corresponds to a text corpus. This level of abstraction is indispensable in order to formalize lexical chains in terms of graph theory.

Definition 2. (Generalized Trees) Let $T(r) = (V, E, r, \mathcal{O})$ be a directed ordered tree with vertex set V , edge set E , root r and order relation $\mathcal{O} \subset V^2$ which for each vertex $v \in V$ linearly orders the set $\{w \mid (v, w) \in E\}$ of vertices to which v is adjacent. Let further $P_{rv} = (v_{i_0}, e_{j_1}, v_{i_1}, \dots, v_{i_{n-1}}, e_{j_n}, v_{i_n})$, $v_{i_0} = r, v_{i_n} = v$, $e_{j_k} = (v_{i_{k-1}}, v_{i_k}) \in E$, $k \in \{1, \dots, n\}$, be the unique path in T from r to $v \in V$ and $V(P_{rv})$ be the set of all vertices of P_{rv} . A *generalized tree*

$$GT(r) = (V, E_{[1]}, E_{[2]}, E_{[3]}, E_{[4]}, E_{[5]}, E_{[6]}, E_{[7]}, r)$$

induced by $T(r)$ is a graph whose partitioned edge set is incrementally defined:

$$\begin{aligned} \text{kernel edges: } E_{[1]} &= E \\ \text{up edges: } E_{[2]} &\subseteq E_u = \{(v, w) \mid v \in V \wedge w \in V(P_{rv}) \setminus \{v\}\} \\ \text{down edges: } E_{[3]} &\subseteq E_d = \{(v, w) \mid w \in V \wedge v \in V(P_{rw}) \setminus \{w\}\} \\ \text{reflexive edges: } E_{[4]} &\subseteq E_r = \{(v, v) \mid v \in V\} \\ \text{across edges: } E_{[5]} &\subseteq V^2 \setminus (E \cup E_u \cup E_d \cup E_r) \\ \text{sequential edges: } E_{[6]} &= \mathcal{O} \\ \text{external edges: } E_{[7]} &= \emptyset \end{aligned}$$

We abbreviate $GT(r) = (V, E_{[1..7]}, r)$ where $e \in E_{[1..7]}$ iff $e \in \cup_{i=1}^7 E_{[i]}$; the sets $E_{[1..7]}$ do not need to be pairwise disjoint. A generalized tree $GT(r) = (V, E_{[1..7]}, r)$ induces a directed tree $\text{Tree}(GT(r)) = T(r)$ called *kernel hierarchy*.

Generalized trees have been introduced by [4, 8] to model graph-like structures of websites which span a graph over a kernel tree without relying on a notion of levels of structuring. That is, due to the degree of structural freedom induced by hyperlinks, constituents of different web pages of the same site do not need to belong to the same level. For more details on this model see [8]. The next definition extends generalized trees in terms of inclusion hierarchies which depart from web structures by integrating such a notion of structure levels (see Fig. 1):

Definition 3. (Generalized Inclusion Hierarchies) Let $x \in B^*$ be a text string and $GT(r) = (V, E_{[1..7]}, r)$ a generalized tree. $GT(r)$ is called a *generalized inclusion hierarchy* $GH(x) = (V, E_{[1..7]}, r, \mathcal{L}, \lambda, \omega)$ induced by x if its kernel hierarchy $\text{Tree}(GT(r)) = (V, E, r, \mathcal{O})$, $E = E_{[1]}$, $\mathcal{O} = E_{[6]}$, the labeling function $\mathcal{L}: V \rightarrow [x]$ and the level mapping λ have the following properties:

1. Ordering:

- $\lambda: V \rightarrow \{0, \dots, \text{hgt}(\text{Tree}(GT(r))) - 1\}$ is a function mapping each vertex $v \in V$ onto a level $\lambda(v)$ such that¹
 - $\lambda(r) = 0$,
 - $\forall v \in V: \text{leaf}(v) = 1 \Rightarrow \lambda(v) = \text{hgt}(\text{Tree}(GT(r))) - 1$,
 - $\forall (v, w) \in E: \lambda(v) < \lambda(w)$.
- Let $V_l = \{v \in V \mid \lambda(v) = l\}$ be the set of all vertices of level l . For each $l \in \{0, \dots, \text{hgt}(\text{Tree}(GT(r))) - 1\}$ \mathcal{O} is a linear order over V_l .

2. Labeling:

- $\mathcal{L}(r) = x$.
- For any $v \in V$, $[v] = \{w \mid (v, w) \in E\}$ is the set of all vertices of $\text{Tree}(GT(x))$ adjacent to v . We distinguish two cases:
 - If $|[v]| > 1$: If $\mathcal{O}(w_{j_1}, w_{j_2}), \dots, \mathcal{O}(w_{j_{|[v]|-1}}, w_{j_{|[v]|}})$ is the linear order of $[v]$ according to \mathcal{O} , then $\mathcal{L}(v) = \mathcal{L}(w_{j_1}) \circ \dots \circ \mathcal{L}(w_{j_{|[v]|}})$ is a text substring of $\mathcal{L}(r)$.
 - If $[v] = \{w\}$: $\mathcal{L}(v) = \mathcal{L}(w)$.
- For any pair of leaves $v, w \in V$ of $\text{Tree}(GT(r))$ with $\mathcal{O}(v, w)$, there exist two tokens $m < n$ such that $\mathcal{L}(v) = x(m)$ and $\mathcal{L}(w) = x(n)$.

- 3. Weighing:** $\omega: V^2 \rightarrow \mathbb{R}$ is a function of the geodesic distance of vertices $v, w \in V$: the shorter this distance, the higher the weight $\omega(v, w)$.

¹ $\text{hgt}(T)$ is the height, i.e. the length of the longest path starting from the root of tree $T = (V, E)$ to one of its leaves. For any vertex $v \in V$: $\text{leaf}(v) = 1$ iff v is a leaf of T .

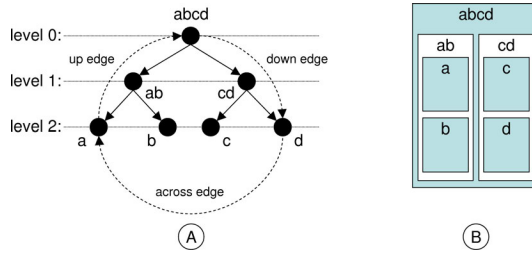


Fig. 1. A generalized inclusion hierarchy mapped onto three levels.

A generalized inclusion hierarchy is a model of the tree-like LDS [12] of a natural language text x as exemplified in Figure 1: (A) exemplifies the stratification of x 's LDS; (B) demonstrates x 's inclusion hierarchy. Note that since generalized inclusion hierarchies are generalized trees, they span graphs by including up, down and across links as exemplified by anaphoric and rhetorical relations. Below, we introduce lexical chains by means of across edges.

Proposition 1. *A generalized inclusion hierarchy induces a one to one mapping between its leafs and the tokens of its root label x such that any dominating vertex is labeled by a substring of x by concatenating the labels of its immediate descendants.*

Because of space limits we skip the proof of this proposition. The kernel hierarchy of a generalized inclusion hierarchy is a formal model of an ordered hierarchy of content objects in the sense of [13]. Note that leafs of the kernel hierarchy T of a generalized inclusion hierarchy are labeled by forms (see Definition 1) in a way that equally labeled leafs relate to different tokens of x . Thus, the order of leafs maps the order of tokens in x as the label of the root of T . Generalized inclusion hierarchies are inclusion hierarchies as their kernel hierarchy maps constituent structure in terms of inclusion. They are generalized as they may extend this kernel by up, down and across edges. Next, we define lexical chains and the graphs they span over inclusion hierarchies. This is done by means of type networks as a model of the terminological ontologies [18] used as a lexical resource for chaining:

Definition 4. (Lexical Type Network) Let A be the set of types according to Definition 1. A lexical type network is a graph $LN = (A, B, \mathcal{L}_A, \mathcal{L}_B, \omega)$ where $B \subseteq A^2$ and $\mathcal{L}_A : A \rightarrow T_A$ is a vertex labeling function for a set T_A of vertex labels (or vertex types, respectively), $\mathcal{L}_B : B \rightarrow T_B$ is an edge labeling function for a set T_B of edge labels and $\omega : B \rightarrow \mathbb{R}$ is an edge weighing function.

As an example of a lexical type network think of a specific release of the Wikipedia in terms of a terminological ontology. In this case, wiki articles, categories, portals etc. induce vertices, whereas hyperlinks induce edges. Thus, vertices are typed, e.g., as *articles*, *portals* or *categories* and edges are labeled as, e.g., *hyperonym of* (in the case of a link from a superordinate to a subordinate category),

article of (in the case of a link from an article to a portal) or as an *association* (in the case of a link between two articles). Edge weighing may be done by measuring the lexical similarity of interlinked pages in terms of the vector space model [14] or latent semantic analysis [7]. Note that in the case of WordNet [5], words and synsets are equally represented as vertices though of different type.

Lexical networks span the reference plane of lexical edges as building blocks of lexical chains. These edges span a sort of commutative diagram in terms of category theory [2]. This is specified in Definition 6 which utilizes the following definition of linkage constraints:

Definition 5. (Linkage Constraints) Let $G = (V, E, \mathcal{L}_V, \mathcal{L}_E, \omega)$ be a graph with a vertex and edge labeling function \mathcal{L}_V and \mathcal{L}_E , respectively. Let further $\omega : E \rightarrow \mathbb{R}$ be an edge weighing function. A *linkage constraint* $P[x, y]$ is a predicate logic statement which solely quantifies over and predicates of elements of G including its labeling and weighing functions. Two vertices $v, w \in V$ are said to obey the constraint $P[x, y]$ if $P[v, w]$ is fulfilled by G when any occurrences of x and y are substituted by v and w , respectively.

In the case of a lexical network a linkage constraint may state, for example, that the shortest path P_{vw} between two vertices v, w is required to be no longer than a certain threshold, that the product of the weights of all edges of P_{vw} is smaller than a certain threshold, that edges of certain types are not allowed to be traversed when building P_{vw} etc. See [3] who describe a system of path and distance related constraints of linkage in terminological ontologies. Now, we can define lexical chains between tokens of a text whose linkage obeys certain constraints: on the token level and on the level of a corresponding type network:

Definition 6. (Lexical Chains) Let $GH(x) = (V, E_{[1..7]}, r, \mathcal{L}, \lambda, \omega)$ be a generalized inclusion hierarchy induced by the text string x , $LN = (A, B, \mathcal{L}_A, \mathcal{L}_B, \omega)$ a lexical type network and $P_{\top}[y_1, y_2], P_{\perp}[z_1, z_2]$ be two linkage constraints over LN and $GH(x)$, respectively.²

- A **lexical edge** is an across edge $(v, w) \in E_{[5]}$ between two leafs v, w of the kernel hierarchy $\text{Tree}(GT(r))$ such that there exist two types $a, b \in A$ where
 - $\mathcal{L}(v) \models a$ and $\mathcal{L}(w) \models b$,
 - a, b are connected by a path P_{ab} in LN ,
 - a, b obey $P_{\top}[y_1, y_2]$ and v, w obey $P_{\perp}[z_1, z_2]$.

We denote the set of all lexical edges in $GH(x)$ by $E_{[5]}|_1 \subseteq E_{[5]}$. Note that $E_{[5]}|_1$ is the set of across edges according to Definition 2.

- A **lexical token chain** K in $GH(x)$ is a subgraph $K = (W, L)$ of $(V, E_{[5]}|_1)$ such that there exists a connected orientation of K , but no lexical edge $e \in E_{[5]}|_1 \setminus L$ which starts from or ends at a vertex $w \in W$.
- For a given $E_{[5]}|_1$, the **lexical partition** of $GH(x)$ is the set $lp(GH(x))$ of all lexical chains induced by $E_{[5]}|_1$.

²Note that in generalized inclusion hierarchies, edge labeling is due to the partitioning of $E_{[1..7]}$ into seven different subsets.

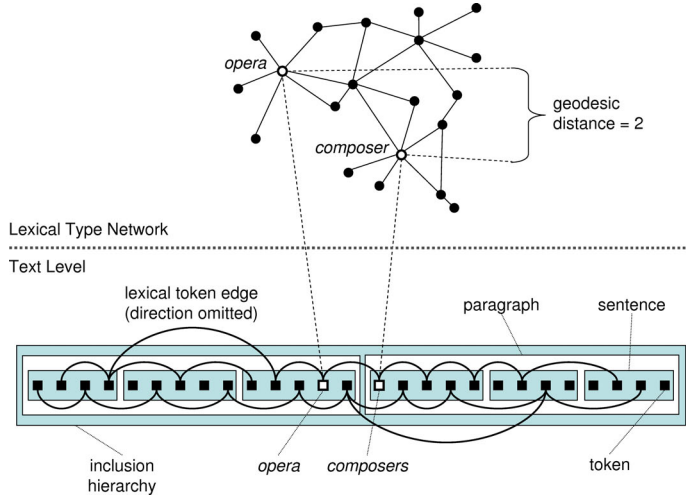


Fig. 2. A lexical partitioning of a generalized inclusion hierarchy.

For a lexical chain $K = (W, L)$, we call $v = \arg \min_{u \in W: \mathcal{L}(u) = n_x(n)} n$ and $w = \arg \max_{u \in W: \mathcal{L}(u) = n_x(n)} n$ the **start** and **end** of K in $GH(x)$, respectively, and denote this by K_{vw} . All other vertices $u \in W \setminus \{v, w\}$ are called **inner** to K_{vw} .

A simple consequence of this definition – which clarifies a topological regularity of lexical chaining – is stated by the following proposition:

Proposition 2. *Lexical chains of the same generalized inclusion hierarchy are pairwise disjoint.*

A lexical partition of a generalized inclusion hierarchy is exemplified in Figure 2. We are finally in a position to define lexical type chains whose vertices do no longer represent tokens, but span subgraphs of type networks:

Definition 7. (Lexical Type Chains) Let $lp(GH(x))$ be the lexical partition of a generalized inclusion hierarchy $GH(x) = (V, E_{[1..7]}, r, \mathcal{L}, \lambda, \omega)$ induced by a text string x , a type network $LN = (A, B, \mathcal{L}_A, \mathcal{L}_B, \omega)$ and two linkage constraints $P_{\top}[y_1, y_2], P_{\perp}[z_1, z_2]$ over LN and $GH(x)$, respectively. For any lexical token chain $(W, L) \in lp(GH(x))$ we define a corresponding lexical type chain (W', L') as follows: $W' = \{a \in A \mid \exists w \in W: \mathcal{L}(w) \models a\}$ and $L' = \{(a, b) \mid \exists e = (v, w) \in L: \mathcal{L}(v) \models a \wedge \mathcal{L}(w) \models b\}$. We write $(W, L) \models (W', L')$ in order to denote that the type chain (W', L') corresponds to the token chain (W, L) . Finally, we define $LP(GH(x))$ as the set of all lexical type chains induced by $lp(GH(x))$.

Figure 3 exemplifies an outline of the token and type partition of a press text where the German release of the Wikipedia has been utilized to span a lexical type network. The underlying chaining algorithm is described in [10]. Intuitively spoken, a token partition of a text is a set of chains of tokens where each of the

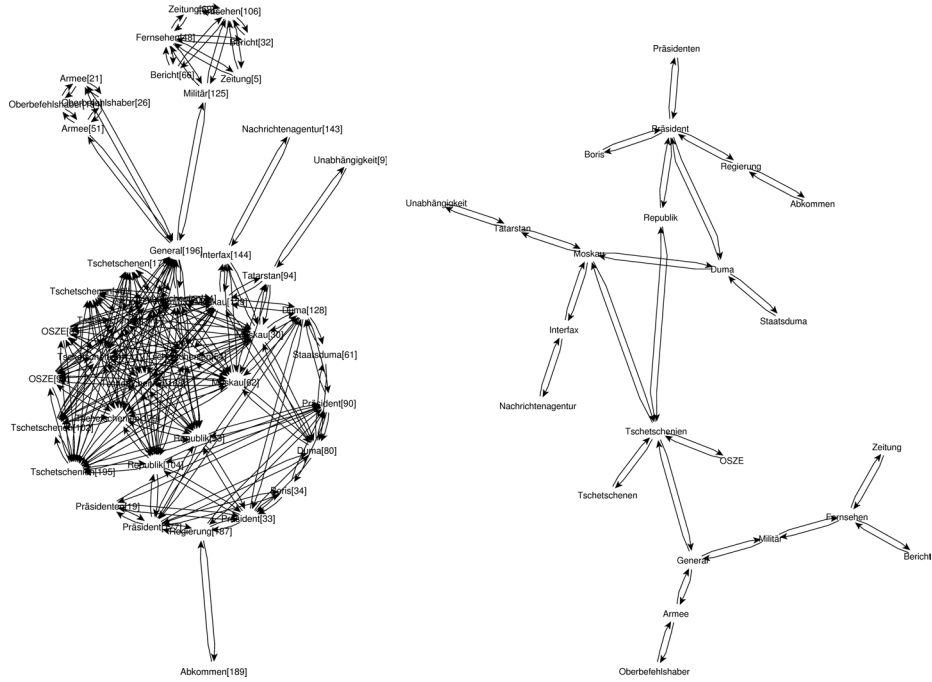


Fig. 3. The largest lexical chain of the token (left) and the type partition (right) of a newspaper article.

chains consists of content-related tokens whose relatedness is computed in terms of the underlying lexical type network (see Definition 4). Note that tokens are numbered in order to distinguish formally identical tokens. In a type partition of a text formally identical tokens are mapped onto the same type where two types are linked if at least two of their token instances are linked in the underlying token partition (see Definition 7). That is, type partitions manifest the linkage structure of lexical text constituents, but abstract away their recurrence. The next section describes how to utilize these graphs in order to quantify text similarities.

3 Quantifying Text Similarities in a Unified Model

Now we introduce a text similarity model based on lexically partitioned generalized inclusion hierarchies. This is done by a linear model which combines the similarity of two texts $x, y \in C$ according to QSA and lexical chaining:

- For two generalized inclusion hierarchies $GH(r), GH(s), \mathcal{L}(r) = x, \mathcal{L}(s) = y,$

$$\sigma_1(x, y) = \text{sim}_1(GH(r), GH(s)) \in [0, 1] \quad (1)$$

is called the *structure-related similarity* of x and y where sim_1 is computed according to QSA as described in [9]. [9] show that the structural similarity of texts is reliably and efficiently computed in terms of feature vectors whose cells solely denote structural features of the input texts.

- For two lexical type partitions $LP(GH(r)), LP(GH(s)), \mathcal{L}(r) = x, \mathcal{L}(s) = y$,

$$\sigma_2(x, y) = \text{sim}_2(LP(GH(r)), LP(GH(s))) \in [0, 1] \quad (2)$$

is the *content-related similarity* of x and y where sim_2 is computed according to the maximum common subgraph approach of [15] as utilized in [10].

- Finally, for some $\alpha \in [0, 1]$, the *text similarity* of x and y is the weighted sum of their structure and content-related similarity

$$\sigma_3(x, y) = \alpha \cdot \sigma_1(x, y) + (1 - \alpha) \cdot \sigma_2(x, y) \quad (3)$$

Equation 3 is a generalized approach to measuring the similarity of texts by exploring their lexical content *and* their logical document structure. It can serve as a starting point of computational linguistic experiments in various fields of application as mentioned in the introductory section. As QSA already proves a high potential in measuring structural similarities, this is a promising way to overcome the limits of the vector space model and to include the contribution by rising Web 2.0-based social ontologies.

4 Conclusion

This paper introduced a formal text representation model which integrates structure- and content-related features of textual units. Its aim is to provide a formal framework for elaborating text similarity models which go beyond the limits of the classical vector space approach. Future work will focus on systematically evaluating this model.

Bibliography

- [1] J. Allan, editor. *Topic Detection and Tracking. Event-based Information Organization*. Kluwer, Boston/Dordrecht/London, 2002.
- [2] M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice Hall, New York/London/Toronto, 1990.
- [3] A. Budanitsky and G. Hirst. Evaluating WordNet-based measures of semantic distance. *Computational Linguistics*, 32(1):13–47, 2006.
- [4] M. Dehmer, A. Mehler, and F. Emmert-Streib. Generalized trees. In *Proceedings of the 2007 International Conference on Machine Learning: Models, Technologies & Applications (MLMTA'07), June 25-28, 2007, Las Vegas, 2007*.
- [5] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, 1998.
- [6] T. Joachims. *Learning to classify text using support vector machines*. Kluwer, Boston, 2002.
- [7] T. K. Landauer and S. T. Dumais. A solution to plato's problem. *Psychological Review*, 104(2):211–240, 1997.
- [8] A. Mehler. Structure formation in the web. In A. Witt and D. Metzger, editors, *Linguistic Modeling of Information and Markup Languages*. Springer, Dordrecht, 2007.
- [9] A. Mehler, P. Geibel, and O. Pustynnikov. Structural classifiers of text types. *Appears in: LDV Forum – Zeitschrift für Computerlinguistik und Sprachtechnologie*, 23(2), 2007.
- [10] A. Mehler and A. Storrer. What are ontologies good for? Evaluating terminological ontologies in the framework of text graph classification. In K.-U. Kühnberger and U. Mönnich, editors, *Proc. of OTT '06*, pages 11–18, 2006.
- [11] S. P. Ponzetto and M. Strube. Deriving a large scale taxonomy from wikipedia. In *Proceedings of AAAI '07*. 2007.
- [12] R. Power, D. Scott, and N. Bouayad-Agha. Document structure. *Computational Linguistics*, 29(2):211–260, 2003.
- [13] A. Renear, E. Mylonas, and D. Durand. Refining our notion of what text really is: The problem of overlapping hierarchies. In N. Ide and S. Hockey, editors, *Research in Humanities Computing*, pages 263–280. Oxford University Press, Oxford, 1996.
- [14] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison Wesley, Reading, Massachusetts, 1989.
- [15] A. Schenker, H. Bunke, M. Last, and A. Kandel. *Graph-Theoretic Techniques for Web Content Mining*. World Scientific, New Jersey, 2005.
- [16] H. Schütze. Automatic word sense discrimination. *Computational Linguistics*, 24(1):97–123, 1998.

- [17] R. Serafin and B. D. Eugenio. FLSA: Extending latent semantic analysis with features for dialogue act classification. In *Proc. of ACL '04*, pages 692–699, 2004.
- [18] J. F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks/Cole, Pacific Grove, 2000.
- [19] L. Steels. Collaborative tagging as distributed cognition. *Pragmatics & Cognition*, 14(2):287–292, 2006.

Cost-based Ranking in Input Output Spaces

Ulf Brefeld

Max Planck Institute for Computer Science
Stuhlsatzenhausweg, 66123 Saarbrücken, Germany
brefeld@mpi-inf.mpg.de

Abstract. We study the problem of finding the most relevant candidates within a finite set of items under budget constraints. The choice of whether to bag an item not only depends on the actual sample but also on the associated costs and the remaining budget. We cast the problem of adapting a ranking function into the structural learning framework to capture the involved multiple-way dependencies. Key to our approach is the linearity of the rephrased task that can be solved optimally by the knapsack algorithm. Since inference is not tractable in general settings, we provide an ϵ -approximation that can be computed in polynomial time.

1 Introduction

Ranking approaches gain a lot of attention in recent years due to an omnipresence of search engines and recommender systems. In many applications, however, the goal is not to output a perfect ranking of items but to identify the maximal subset of items that fulfills certain additional constraints.

As an example consider software companies using the expiring time to the next deadline for final debugging. Since it is generally impossible to revise the whole program in only a few days, the companies face a combinatorial problem of deciding on which parts of their code they should have a second look at *and* on which parts they realistically can look at within the given time frame. Pure ranking approaches ignore the time constraint and only aim at finding the most buggy routines. However, this solution is inappropriate for the exemplary application since the time needed to debug the top-scoring routines might overrun the time left.

We propose a novel approach to the constraint-based identification of relevant items. To learn the ranking function, we cast the constraint-based task into the structured learning framework that allows for capturing the involved multiple-way dependencies. We devise a generalized linear model in joint input output space that can be optimized by structural support vector machines [3]. The derived model implements the knapsack criterion and is not tractable for large data sets. As a remedy, we propose a polynomial time approximation to the exact solution.

The remainder is organized as follows. The problem setting is introduced in Section 2. We then derive the constrained-based ranking method in Section 3 and Section 4 sketches the ϵ -approximation. Section 5 concludes.

2 Problem Setting

In this Section we abstract the task of finding the most relevant items under budget constraints. We decompose the task into a *decoding* and a parameter estimation step and present an appropriate loss function.

Given a training set $\{(\mathbf{x}^i, \mathbf{y}^i, \mathbf{c}^i, b^i)\}_{i=1}^n$, where \mathbf{x}^i denotes a collection of items $\mathbf{x} = \{x_1^i, \dots, x_{m_i}^i\}$ with associated costs $\mathbf{c}^i = (c_1^i, \dots, c_{m_i}^i)^\top \in \mathbb{R}_+^{m_i}$, and b^i denotes the budget to spend. The corresponding output \mathbf{y}^i is a subset of items contained in \mathbf{x}^i . We treat the output $\mathbf{y}^i = (y_1^i, \dots, y_{m_i}^i)^\top \in \{0, 1\}^{m_i}$ as a binary vector, indicating whether items are contained in the solution, that is $y_j^i = 1$ if x_j^i is relevant and 0 otherwise. The true labeling is required to achieve the highest profit according to an (unknown) profit function p , that is,

$$\sum_{j=1}^{m_i} y_j^i p(x_j^i) > \max_{\substack{\bar{\mathbf{y}} \neq \mathbf{y}^i \\ \sum_j \bar{y}_j c_j^i < b^i}} \sum_{j=1}^{m_i} \bar{y}_j p(x_j^i),$$

with $p(x) \geq 0$ for all x . Of course, having the true profit function renders the parameter estimation unnecessary. However, recall that we focus here on problems in which the true profit function is not accessible or unknown. We thus aim at finding a ranking function f that assigns higher decision scores to the true labeling than to all valid alternative labelings, that is,

$$\forall_{i=1}^n \forall_{\bar{\mathbf{y}} \neq \mathbf{y}^i \wedge \sum_j \bar{y}_j c_j^i < b^i} f(\mathbf{x}^i, \mathbf{y}^i, \mathbf{c}^i, b^i) > f(\mathbf{x}^i, \bar{\mathbf{y}}, \mathbf{c}^i, b^i).$$

Similarly, at prediction time we are seeking the top-scoring hypothesis given by

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\bar{\mathbf{y}}: \sum_j \bar{y}_j c_j < b} f(\mathbf{x}, \bar{\mathbf{y}}, \mathbf{c}, b). \quad (1)$$

In the remainder we will refer to the computation of the argmax as *decoding* step which we will address in the next section.

We measure the quality of a hypothesis by a loss function Δ that details the differences between the true labeling and the prediction. For instance, an appropriate loss function in our discourse area is a Hamming-like loss that simply counts the number of errors in the prediction, that is, $\Delta_H(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{j=1}^m 1_{[y_j \neq \hat{y}_j]}$, where $1_{[z]}$ equals 1 if z is true and 0 otherwise. In order to find a hypothesis that generalizes well on new and unseen data, we seek the minimizer of the regularized empirical risk

$$\hat{R}(f) = \sum_{i=1}^n \Delta_H(\mathbf{y}^i, \operatorname{argmax}_{\bar{\mathbf{y}}: \sum_j \bar{y}_j c_j^i < b^i} f(\mathbf{x}^i, \bar{\mathbf{y}}, \mathbf{c}^i, b^i)) + \eta \|f\|^2.$$

3 Learning Knapsack

Given an input triple $(\mathbf{x}, \mathbf{c}, b)$, the goal is to maximize the profit without exceeding the budget. This can be expressed as the following constrained optimization

problem, also known as the *knapsack* problem [4],

$$\max_{\mathbf{y}} \sum_{i=1}^m y_i p(x_i) \quad \text{s.t.} \quad \sum_{j=1}^n y_j c_j \leq b. \quad (2)$$

Since the true profits are unknown, we employ a $\boldsymbol{\lambda}$ -parameterized profit function by linearly combining features $\psi(x)$ drawn from objects,

$$\hat{p}(x) = \langle \boldsymbol{\lambda}, \psi(x) \rangle. \quad (3)$$

Depending on the problem at hand, features may capture the size, weight, or color of item x . Substituting Equation (3) into (2) allows to rewrite the objective as a generalized linear model in joint input output space. We have

$$\begin{aligned} \sum_j y_j \hat{p}(x_j) &= \sum_j y_j \langle \boldsymbol{\lambda}, \psi(x_j) \rangle \\ &= \langle \boldsymbol{\lambda}, \underbrace{\sum_j y_j \psi(x_j)}_{=: \Phi(\mathbf{x}, \mathbf{y})} \rangle, \end{aligned}$$

where $\Phi(\mathbf{x}, \mathbf{y})$ is sometimes called the joint feature mapping of inputs and outputs. Thus, the decision value of the model f is determined by

$$f(\mathbf{x}, \mathbf{y}, \mathbf{c}, b) = \begin{cases} \langle \boldsymbol{\lambda}, \Phi(\mathbf{x}, \mathbf{y}) \rangle & : \sum y_j c_j \leq b \\ \text{not defined} & : \text{otherwise.} \end{cases}$$

The distinction of valid and illegal assignments can be augmented with the computation of the argmax at prediction time, leading to an elegant model that can be optimized by structural SVMs [3]. Let $\mathcal{Y}^i = \{\mathbf{y} : \sum y_j c_j^i \leq b^i\}$, we obtain,

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\bar{\mathbf{y}} \in \mathcal{Y}^i} f(\mathbf{x}, \bar{\mathbf{y}}, \mathbf{c}, b) = \operatorname{argmax}_{\bar{\mathbf{y}} \in \mathcal{Y}^i} \langle \boldsymbol{\lambda}, \Phi(\mathbf{x}, \bar{\mathbf{y}}) \rangle. \quad (4)$$

If the decision values were discrete, Equation (4) could be computed by the knapsack algorithm in time $\mathcal{O}(m^2 \max_j \langle \boldsymbol{\lambda}, \psi(x_j) \rangle)$ [4]. The next Section proposes an approximation by inducing equivalence classes on the decision values.

Notice, that the definition of $\Phi(\mathbf{x}, \mathbf{y})$ allows to rewrite the inner product in joint input output space in terms of a kernel function $k(x, x') = \langle \psi(x), \psi(x') \rangle$, defined solely on pairs of input items; we have $\langle \Phi(\mathbf{x}, \mathbf{y}), \Phi(\mathbf{x}', \mathbf{y}') \rangle = \sum_j y_j y'_j k(x_j, x'_j)$.

4 ϵ -approximate Inference

Due to non-discrete profit estimates $\hat{p}(x) = \langle \boldsymbol{\lambda}, \psi(x) \rangle$, the knapsack algorithm needs to enumerate all possible candidate sets which is prohibitive for real world applications and renders the optimization problem intractable. As a remedy, we induce equivalence classes by some rounding operations on the estimates \hat{p}

as follows. Let $\epsilon > 0$, we define a constant $\kappa = \frac{\epsilon \cdot \max_j \langle \lambda, \psi(x_j) \rangle}{m}$ that induces equivalence classes by normalizing and rounding the decision values; we have,

$$\hat{p}(x_j) \leftarrow \left\lfloor \frac{\langle \lambda, \psi(x_j) \rangle}{\kappa} \right\rfloor.$$

Utilizing these rounded profits instead of the decision values leads to a fully polynomial time approximation scheme for the knapsack problem that can be solved in $\mathcal{O}(m^2 \lfloor \frac{m}{\epsilon} \rfloor)$ where the error is bounded by the factor ϵ [4].

Due to the approximate inference, convergence of the SVM to the global optimum for the parameter estimation step cannot be guaranteed. However, empirical studies on the benefits of approximate over exact inference techniques show that the former frequently leads to more accurate prediction models in many domains [1, 2].

5 Conclusion

We devised a novel large margin approach in joint input output space for relevance ranking under budget constraints. The proposed solution allows to capture multiple-way dependencies within the data by adapting a parameterized profit function to a labeled training sample. Since the exact solution is intractable for large data sets we propose an approximation based on equivalence classes. Initial empirical results are promising and will be presented at the workshop. The proposed algorithm can be easily generalized to maintain several budgets and/or bags.

Acknowledgments

This work has been funded by the German Science Foundation DFG under grant SCHE540/10-2.

References

1. U. Brefeld, T. Klein, and T. Scheffer. Support vector machines for collective inference. In *Proceedings of the International Workshop on Mining and Learning with Graphs*, 2007.
2. T. Finley and T. Joachims. Parameter learning for loopy markov random fields with structural support vector machines. In *Proceedings of the ICML Workshop on Constraint Optimization and Structured Output Spaces*, 2007.
3. I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.
4. V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

Information Extraction From Annotation Graphs Serving for the Adapation of Ontologies – Position Paper –

Uwe Mönnich¹, Jens Michaelis², Ekaterina Ovchinnikova², Tonio
Wandmacher², and Kai-Uwe Kühnberger²

¹Seminar für Sprachwissenschaft, University of Tübingen

²Institut für Kognitionswissenschaft, Universität Osnabrück

{um@sfs.uni-tuebingen.de, jmichael@uos.de, e.ovchinnikova@gmail.com,
tonio.wandmacher@uos.de, kkuehnbe@uos.de}

Abstract. The importance of annotation graphs increases rapidly due to new developments in multi-media, text technology, and semantic web applications. The paper proposes an architecture designed for extracting semantic knowledge from information coded in annotation graphs. More precisely, we suggest a logical specification of annotation graphs commonly used to code documents that are inherently structured on various levels. The corresponding tree-like graph representations can in turn be used to build an ontology of the domain of interest. Because inconsistencies can occur in this building process, rewriting algorithms are applied to resolve these inconsistencies automatically. Learning is therefore based on the rewriting of the logical descriptions.

The Logical Representation of Annotation Graphs

It is a matter of fact that a long history in artificial intelligence and computational linguistics tries to develop tools to extract semantic knowledge from syntactic information. In particular, from a text technological point of view the general research perspective is to extract (semantic) information from annotated documents. Regarding this aim, some of the relevant annotation models used are multilayer annotations, hyperlinks, discourse structure, or (classical) linguistic description levels. It is not only rather unclear, how the extraction of semantic knowledge is possible from the assembly of often heterogeneous formal specifications in an appropriate unified way. It can also be asked, if this is possible at all, whether a corresponding extraction can be performed in such a way that integration, or even adaptation, into an (existing) ontological representation becomes more or less directly possible.

An important aspect concerning annotation models is the development of a suitable logic system in order to specify the syntactic and semantic structures of such models. The challenges of such architectures are twofold: first, the dynamic interaction between syntactic and semantic information must be represented and second, the efficiency of algorithms must be guaranteed.

Fortunately, in the case of *annotation graphs (AGs)* in the sense of [1], techniques from parameterized complexity theory can be exploited. This theory provides powerful tools for a detailed investigation of algorithmic problems. As it turns out, the concept of *treewidth* in the sense of [11], indicating the similarity of a graph or a relational structure with a tree, is a parameter which helps to show that many otherwise intractable problems become computable in linear time when restricted to tree-like inputs (cf. [6]).

The key feature of AGs is their abstraction from the diversity of concrete formats used for the transcription of text and speech. This feature makes them an ideal candidate for the comparison of different annotation systems, e.g., those currently developed by several linguistic collaborative research centers in Germany. Translating these different representation schemes into the framework of AGs is a necessary prerequisite for the transfer of the pleasant computational properties of AGs to the original systems. This is particularly important in those circumstances where the natural data structure of the concrete markup system cannot be readily understood as describing trees.

A classical domain for multilayered annotations is linguistics. Utterances of speakers can be considered from different perspectives: examples are syntactic, semantic, discourse and intonation aspects, just to mention some of them. Representing these types of data in one representation format yields overlapping hierarchies. Moreover, the resulting structures are naturally considered as graphs rather than trees. These challenges can be met by representing AGs in logical form resulting in a representation of low descriptive complexity, the representability as open diagrams, and an abstract format for the interaction of different linguistic levels. Furthermore logical representations are amenable to techniques from logical graph theory, especially in terms of *Monadic Second Order (MSO)* logic based on the work of Courcelle starting from [2].

While the logical approach towards annotation models provides a unified format for the syntactic level, it still has to be complemented with a component that serves to integrate syntactic with semantic structures. Clearly an MSO representation of AGs is not in accordance to standards generally accepted for representing semantic knowledge. These standards usually require a reduced representation of the available knowledge to a form of a two-variable logic. A procedure of reducing the rich MSO representations to a two-variable representation where semantically irrelevant information is discarded and complex representations are simplified yields a core of conceptual information that can be used for a learning procedure on ontologies.

Learning Ontologies as an Adaptation Process

Provided a reduced representation of the information from AGs is given, which is suitable to represent ontological knowledge covered in the original annotations, there cannot be a guarantee that this knowledge remains stable during potential updates. During the last years, a significant endeavor has been undertaken in order to develop frameworks for automatically learning and expanding ontolog-

ical knowledge. Due to the fact that the hand-coded development of ontological resources is tedious and time-consuming, a natural idea for an alternative is to apply automatic update devices. Because it is uncertain whether such updates can be performed in a consistent way, rewriting algorithms are needed to consistently modify and expand ontologies in an adaptation process.¹ Various forms of interactions can make modifications or adaptations of ontological knowledge necessary. Besides the incremental expansion of the ontology by further concepts and relations, we just mention two examples of potential problems that may occur (in [8] more types of problems are described and assessed):

- Polysemy: If the concept *tree* is declared to be a subconcept of both *plant* and of *data structure*, where *plant* and *data structure* are disjoint concepts. Although both interpretations of *tree* are correct, it is still necessary to describe two different concepts with different identifiers in the ontology (e.g. *TreePlant*, *TreeStructure*).
- Overgeneralization: The well-known Tweedy example from non-monotonic reasoning describes a situation in which a concept is too general (*Bird* is subsumed by *CanFly*), in order to incorporate the new information about a *Penguin Tweedy* (hence a *Bird*) which cannot fly.

In [7], [8], [9], and [10] various algorithms are proposed in order to resolve ontologies coded in description logics of variable strengths.² Although these approaches provide a first step for a general framework of resolving occurring inconsistencies in ontology design, there is the need to develop rewriting algorithms for stronger logics, because the reduced representation of AGs are (in general) coded in relatively unrestricted two-variable fragments of first-order logic. Contrary to these representations, standardly used description logics are rather restricted concerning their applicable quantifications and set of constructors.

It should be noticed that the mentioned rewriting mechanisms of ontologies are rather different from classical machine learning techniques. Whereas many state-of-the-art learning methods are based on neural or probabilistic techniques, the present approach focuses on logic-based rewriting algorithms.³ As a consequence of this logical basis, several classical concepts from machine learning are only weakly related. Just to mention one of these concepts, convergence properties, for example in the sense of PAC learning [13], have no clearly identifiable counterpart in our approach. Nevertheless certain clear limitations of logic can be specified: automatic rewriting techniques can only be successfully applied to certain types of occurring inconsistencies (in our case we mentioned overgeneralization and polysemy, in [8] a further type is introduced), whereas other inconsistencies, for example, inconsistencies of the general form A and $\neg A$, cannot be resolved on a purely logical basis.

¹ Just to mention some examples of recent work dealing with debugging or resolving inconsistencies, the reader is referred to [12], [4], [5]

² For example, solutions for rewriting ontologies in the description logics $\mathcal{AL}\mathcal{E}$, $\mathcal{AL}\mathcal{C}$, and $\mathcal{AL}\mathcal{C}\mathcal{N}$ are provided in these papers.

³ Perhaps research that is closest in spirit to the present approach is work applying ILP techniques to ontology design [3].

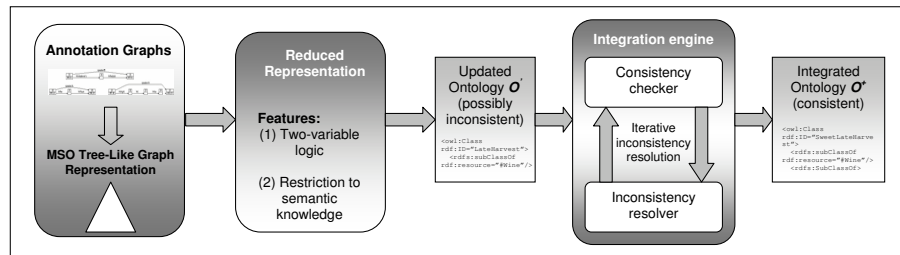


Fig. 1. The overall architecture for (a) the extraction of information coded in annotation graphs, (b) the coding of a reduced representation in a two-variable logic, and (c) the adaptation of the knowledge base by learning.

The Overall Architecture

Figure 1 depicts a proposal for an overall architecture. Annotation graphs are used as rich information sources. A tree-like graph representation of annotation graphs in terms of MSO provides a precise logical description of these graphs. Based on these logical descriptions the reduction of the available information with respect to ontological knowledge yields the basis for the usage of an appropriate two-variable logic. Because the update of a knowledge base O by new axioms can result in a inconsistent knowledge base O' , an integration engine resolves occurring inconsistencies and integrates the newly provided knowledge to the extended ontology yielding O^+ .

Several problems potentially occurring in the proposed architecture are not solved yet: For example, the transition from rich MSO representations to rather weak semantic representations and the extension of rewriting systems with respect to stronger two-variable logics are waiting for solutions. Nevertheless a convincing treatment of these problems would provide a uniform framework for extracting semantic knowledge from structured data for text technological applications. Due to the fact that automatic learning procedures for ontology learning are currently not sufficiently developed yet, the usage of structured data for the extraction of semantic information could be beneficial for a wide range of applications.

Acknowledgements

This research was partially supported by the grant MO 386/3-4 of the German Research Foundation (DFG).

References

1. Bird, S. and Liberman, M. A formal framework for linguistic annotation. *Speech Communication*, 33:23–60, 2001.

2. Courcelle, B. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.
3. Fanizzi, N., Ferilli, S., Iannone, L., Palmisano, I. and Semeraro, G. Downward Refinement in the ALN Description Logic. *Proc. of the Fourth International Conference on Hybrid Intelligent Systems (HIS'04)*, 68–73, 2005.
4. Haase, P., van Harmelen, F., Huang, Z., Stuckenschmidt, H., Sure, Y. A framework for handling inconsistency in changing ontologies. *Proc. of the Fourth International Semantic Web Conference*, LNCS, Springer, 2005.
5. Kalyanpur, A. *Debugging and Repair of OWL Ontologies*. Ph.D. Dissertation, University of Maryland College Park, 2006.
6. Michaelis, J. and Mönnich, U. Towards a Logical Description of Trees in Annotation Graphs. Under review.
7. Ovchinnikova, E. and Kühnberger, K.-U. Adaptive $\mathcal{AL}\mathcal{E}$ -TBox for Extending Terminological Knowledge. In A. Sattar, B. H. Kang (eds.): *Proceedings of the 19th ACS Australian Joint Conference on Artificial Intelligence*, LNAI 4304, Springer, pp. 1111-1115, 2006.
8. Ovchinnikova, E. and Kühnberger, K.-U. Aspects of Automatic Ontology Extension: Adapting and Regeneralizing Dynamic Updates. In M. Orgun & T. Meyer (eds.): *Advances in Ontologies 2006, Conferences in Research and Practice in Information Technology*, 72:52-60, 2006.
9. Ovchinnikova, E., Wandmacher, T. and Kühnberger, K.-U. Solving Terminological Inconsistency Problems in Ontology Design. *International Journal of Interoperability in Business Information Systems*, 4:65–80, 2007.
10. Ovchinnikova, E., Kühnberger, K.-U. Debugging Overgeneralized Concepts in \mathcal{ALCN} Terminologies. In preparation.
11. Robertson, N. and Seymour, P.-D. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.
12. Schlobach, S., Cornet, R. Non-standard reasoning services for the debugging of description logic terminologies. *Proc. of IJCAI'03*, Morgan Kaufmann, 2003
13. Valiant, L. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Data-Driven Learning of Functions over Algebraic Datatypes from Input/Output-Examples

Emanuel Kitzelmann

Faculty of Information Systems and Applied Computer Sciences,
University of Bamberg, 96045 Bamberg, Germany
emanuel.kitzelmann@wiai.uni-bamberg.de
<http://www.cogsys.wiai.uni-bamberg.de/kitzelmann/>

Abstract. We describe a technique for inducing recursive functional programs over algebraic datatypes from few non-recursive and only positive ground example-equations. Induction is data-driven and based on structural regularities between example terms. In our approach, functional programs are represented as constructor term rewriting systems containing recursive rewrite rules. In addition to the examples for the target functions, background knowledge functions that may be called by the induced functions can be given in form of ground equations. Our algorithm induces several dependent recursive target functions over arbitrary user-defined algebraic datatypes in one step and automatically introduces auxiliary subfunctions if needed. We have implemented a prototype of the described method and applied it to a number of problems.

1 Introduction

Automatic inductive synthesis of recursive functional or logic programs from input/output-examples (I/O-examples) is an active area of research since the sixties (see [1] for classical methods, [2] for systems in the field of inductive logic programming, and [3] for recent research).

There exist two general approaches to tackle inductive program synthesis: (i) In the generate-and-test approach (e.g., the ADATE system [4]), programs of a defined class are enumerated heuristically and then tested against given examples. (ii) In the analytical approach, programs of a defined class are derived by detecting recurrences in given examples which are then generalized to recursively defined functions; i.e., hypotheses are (more or less) computed instead of searched. Generate-and-test methods are applicable for very general program classes since there are no principal difficulties in enumerating programs. They naturally facilitate usage of predefined functions (background knowledge, BK) in induced programs. On the other side, generate-and-test methods are search intensive and therefore time consuming. Analytical approaches have more restricted program classes and generally do not facilitate the usage of background knowledge (BK). On the other side, analysis minimizes search. The goal of the

approach described in this paper was to relax the analytical approach by applying a search for hypotheses but by keeping analytical concepts within the search. The result is a functional program induction system that data-driven searches a comparatively less restricted hypothesis space and allows the use of BK.

One classical and influential analytical approach is from Summers [5], who put inductive synthesis on a firm theoretical foundation. His system induces functional Lisp programs containing one function definition whose body consists of a conditional for an arbitrary number of base cases and one recursive case containing one recursive call. Parameters are restricted to be S-expressions (the general datatype in Lisp) and I/O-examples have to be linearly ordered. A more recent analytical system inspired by this approach is described in [6].

Another line of research is the field of inductive logic programming (ILP). Though ILP has a focus on learning non-recursive concept descriptions, there has also been research in inducing recursive programs (see [2]). One relatively recent analytical ILP program synthesis method is DIALOGS [7]. In order to induce more complex functions, e.g., the quicksort algorithm and to automatically invent needed subfunctions, e.g., the partitioning function for quicksort, it makes use of some general schemas, e.g., divide-and-conquer, which the user must choose and requires further information from the user. The schemas strongly restrict the hypothesis space. Moreover, DIALOGS is restricted to some predefined datatypes like lists and numbers.

The new approach described in this paper induces multiple dependent target functions over arbitrary user defined algebraic datatypes in one step, facilitates the use of BK and allows complex recursion patterns (nested calls of induced recursive functions, mutual recursion, tree recursion, and arbitrary numbers of base cases and recursive cases). Additionally needed subfunctions can be introduced automatically if the calling relation fulfills some conditions. Its integrated analytical concepts lead to induction times which are very small compared to powerful generate-and-test systems like ADATE. E.g., to induce the *Reverse*-function, our system needs milliseconds whereas ADATE needs more than a minute on the same computer. Particularly the capability of inducing multiple related target functions as for example mutual recursive definitions for *Even* and *Odd* on natural numbers is a feature not provided by most program induction systems. A recent ILP system also capable of learning multiple related recursively defined target concepts is ATRE [8].

2 Preliminaries

We represent functional programs as sets of equations (pairs of terms) over a many-sorted first order signature Σ . I.e., we abstract from any concrete functional programming language and do not consider higher order functions. Each equation specifying a function F has a left hand side (lhs) of the form $F(t_1, \dots, t_n)$ where neither F nor the name of any other of the defined functions occur in the t_i . Thus, the symbols in the signature Σ are divided into two disjoint subsets \mathcal{F} of *defined function symbols*, e.g., F , and \mathcal{C} of *constructors*. Terms without de-

defined function symbols are called constructor terms. Ground constructor terms denote values. The constructor terms t_i in the lhs of the equations for a defined function F may contain variables and are called *pattern*. This corresponds to the concept of pattern matching in functional programming languages and is the only form of case distinction. Each variable in the rhs of an equation must occur in the lhs, i.e., in the pattern. To evaluate a function defined by equations we read the equations as simplification rules from left to right. A set of simplification (or rewrite) rules is called *term rewriting system (TRS)*. TRSs whose lhs have defined function symbols as roots and constructor terms as arguments, i.e., whose lhs have the described pattern-matching form, are called *constructor term rewriting systems (CSs)*.

In order to formalize the simplification process we first introduce some standard concepts on terms: One can denote each subterm of a term t by its unique *position* within t , a sequence of positive integers. The term t itself stands at position ϵ —the empty sequence—called *root position*. If $t = f(t_1, \dots, t_n)$ then each t_i stands at position i . If a subterm s of t_i stands at position u within t_i then it stands at position $i.u$ within t . The subterm at position u is written $t|_u$. Consider the term $t = f(a, g(x, y))$. Then, e.g., holds $t|_2 = g(x, y)$ and $t|_{2.1} = x$.

A *substitution* σ is a mapping from variables to terms and is extended to a mapping from terms to terms which is also denoted by σ and written in postfix notation; $t\sigma$ is the result of applying σ to all variables in term t . If $s = t\sigma$, then t is called *generalization* of s and we say that t *subsumes* s and that s *matches* t by σ . Given two terms t_1, t_2 and a substitution σ such that $t_1\sigma = t_2\sigma$, then we say that t_1, t_2 *unify* and call σ *unifier* of t_1 and t_2 . Given a set of terms, $S = \{s, s', s'', \dots\}$, then there exists a term t which subsumes all terms in S and which is itself subsumed by each other term subsuming all terms in S . The term t is called *least general generalization (lgg)* of the terms in S [9].

A *context* is a term that contains a distinguished symbol \square , denoting *holes*, at at least one position. If C is a context containing n holes then $C[t_1, \dots, t_n]$ denotes the term resulting from replacing the n holes in C by the t_i from left to right. The *rewrite relation* \rightarrow_R established by a CS R is defined as follows: A term t rewrites to s according to R , written $t \rightarrow_R s$ iff there exists a rule $l \rightarrow r$ in R , a substitution σ , and a context C such that $t = C[l\sigma]$ and $s = C[r\sigma]$. Evaluating an n -ary function F for an input i_1, \dots, i_n consists of repeatedly rewriting the term $F(i_1, \dots, i_n)$ w.r.t. the rewrite relation until the term is in *normal form*, i.e., cannot be further rewritten. A sequence of (in)finately many rewrite steps $t_0 \rightarrow_R t_1 \rightarrow_R \dots$ is called *derivation*. If a derivation starts with term t and results in a normal form s , then s is called normal form of t , written $t \xrightarrow{!} s$. We say that t normalizes to s . In order to define a *function* on a domain (a set of ground terms) by a CS, no two derivations starting with the same ground term may lead to different normal forms, i.e., normal forms must be unique. A sufficient condition for this is that no two lhs of a CS unify; then the CS is *confluent*. A CS is *terminating* if each possible derivation terminates. A sufficient condition for termination is that the arguments/inputs of recursive calls strictly decrease within each derivation and w.r.t. a well founded order.

3 Function Induction by Pattern Refinement, Matching, and Ubiquitous Subprogram Introduction

3.1 Analytical Induction of Recursive Functions

We write t^n for a sequence of terms t_1, \dots, t_n . If i^n is an input to a recursively defined function F with a corresponding output term o and $i^{n'}$ is the input to F resulting from a recursive call of F within computing i , then o contains the output term o' for $i^{n'}$ as subterm. Using this structural regularity between computations of recursively defined functions in order to infer the recursive definition from I/O-examples is the core of analytical function induction as proposed by Summers [5]. Examples for a target function F are equations of the form $F(i^n) = o$ where the i^n and o are ground constructor terms and are called example inputs and outputs respectively. A necessary condition for applying the described principle is that for each example input, all inputs resulting from recursive calls are also included in the example set. The following definition states this condition formally and extended to more than one target function.

Definition 1. *Let R be a CS which correctly computes a set of example equations E . The example equations E are called recursively subsumed w.r.t. R if for all example inputs i^n hold: Let $F(p^n) \rightarrow t$ be a rule in R such that i^n matches p^n by substitution σ . Then for each (recursive) call $F'(r^m)$ of a defined function F' of R in $t \in E$ contains the instantiation $r^m \sigma$ as an example input.*

3.2 Overview over the Induction Algorithm

We require that induced CSs are terminating and that they represent functions, i.e., unique normal forms. We require that a hypothesis is *correct*:

Definition 2. *A hypothesis, i.e., a CS R , is consistent or complete w.r.t. a set of example equations if for each example equation $F(i^n) = o$ holds: $F(i^n) \xrightarrow{!}_R o$ or $F(i^n) \xrightarrow{!}_R s$ for a non-constructor term s (consistent) or $F(i^n) \xrightarrow{!}_R s$ for a constructor term s (complete) respectively. A hypothesis is correct if it is both consistent and complete.*

The consistency condition assures that if the induced function is defined for an input then the function value is the specified output. The completeness condition assures that the induced function is total on the example inputs.

Fig. 1 shows example equations for the *Reverse*-function and the equations induced by our system. Only the example equations and the corresponding datatype definitions were provided.¹ Two subfunctions have been introduced automatically, *Last* and *Init*, which resp. compute the last element of a list and the list without the last element.²

¹ Note that the example equations contain variables and hence are a generalized form of I/O-examples. This allows fewer examples and makes induction faster but requires an extended theory not described in this paper.

² Automatically invented subfunctions are named “Sub1”, “Sub2” etc. by the system.

Example equations:

1. $Reverse([]) = []$
2. $Reverse([X]) = [X]$
3. $Reverse([X, Y]) = [Y, X]$
4. $Reverse([X, Y, Z]) = [Z, Y, X]$
5. $Reverse([X, Y, Z, V]) = [V, Z, Y, X]$

Induced CS:

$$\begin{array}{ll}
 Reverse([]) & \rightarrow [] \\
 Reverse([X|Xs]) & \rightarrow [Last([X|Xs])|Reverse(Init([X|Xs]))] \\
 Last([X]) & \rightarrow X \\
 Last([X_1, X_2|Xs]) & \rightarrow Last([X_2|Xs]) \\
 Init([X]) & \rightarrow [] \\
 Init([X_1, X_2|Xs]) & \rightarrow [X_1|Init([X_2|Xs])]
 \end{array}$$

Fig. 1. Example equations and the induced solution for the *Reverse*-function

The induction of a terminating, confluent, correct CS is organized as a kind of best first search. During search, a hypothesis is a set of equations entailing the example equations and constituting a terminating and confluent CS *but potentially with variables in the rhss not occurring in the lhss*. I.e., hypotheses during search do not necessarily represent *functions*. We call such equations and hypotheses *unfinished*. A goal state is reached, if at least one of the best—according to a criterion explained below—hypotheses is finished.

Our induction bias is to prefer CSs whose patterns partition the example inputs into fewer subsets. This corresponds to preferring fewer case distinctions and leads, in some sense, to most general hypotheses. Regarding *one* function, this prefers CSs with fewer rules, since each determines one subset. But consider the solution for *Reverse* as shown in Fig. 1. The CS contains six rules but the number of induced subsets is only three, because *Last* and *Init* induce the same subsets (pattern $[X]$ subsumes example 2, pattern $[X_1, X_2|Xs]$ examples 3 - 5) which are again partitions of the subset induced by pattern $[X|Xs]$ of *Reverse* such that pattern $[]$ from *Reverse* remains and induces the subset containing example 1. If we would have chosen fewer rules as preference bias then obviously the five example equations themselves would have been favored over the solution with *Init* and *Last* such that no generalization would have taken place.

W.r.t. this bias and in order to get a complete CS, the initial CS has one rule per target function such that its pattern subsumes all example inputs. In most cases (e.g., for recursive functions) one rule is not enough and thus unfinished. Then for one unfinished rule successors are computed leading to new hypotheses. Now repeatedly unfinished rules of best hypotheses are replaced. Since different hypotheses may contain the same rule, successor rules originate successors of *all* hypotheses containing this rule, i.e., hypotheses are processed in parallel.

3.3 Initial Rules

Given a set of example equations for one target function, the initial rule is constructed by first antiunifying [9] all example inputs.³ This leads to the lgg of the example inputs, i.e., to the most specific pattern subsuming all example inputs. Second, the example outputs are antiunified w.r.t. the substitutions resulting from antiunification of the inputs. This gives the lgg of all outputs were variables from the pattern are used if possible. Considering only lgg of example inputs as patterns narrows the search space. It does not constrain completeness of hypotheses regarding the example equations as shown by the following lemma.

Lemma 1. *Let R be a CS with non-unifying patterns and which is correct regarding a set of recursively subsumed example equations. Then there exists a CS R' such that R' contains exactly one pattern p' for each pattern p in R , each p' is the lgg of all example inputs matching the corresponding pattern p , and R and R' compute the same normal form for each example input.*

Proof. Sketch: It has to be shown that if the patterns p are replaced by lggs p' then also the rhss can be replaced such that the rewrite relation remains the same for the example inputs. This can be done by assuming a position in p and p' where p and p' are different and then constructing a suitable rhs.

3.4 Processing Unfinished Rules

This section describes the three methods for computing successor rules. All three methods are applied to a chosen unfinished rule. The first method, *splitting rules by pattern refinement*, replaces an unfinished rule with pattern p^n by at least two new rules with more specific patterns in order to establish a case distinction. The second method, *introducing function calls*, implements the principle described in Sec. 3.1 in order to introduce recursive calls or calls to other defined functions which can be further target functions or BK functions. The third method, *introducing subfunctions*, generates new induction problems, i.e., new example equations, for the unfinished subterms of an rhs. These new problems are treated the same way as the “original” problems, i.e., this method implements the capability to automatically find auxiliary subfunctions.

Splitting Rules by Pattern Refinement The first method for generating successors of a rule is to replace its pattern p^n by a set of more specific patterns, such that the new patterns induce a partition of the example inputs matching p^n . This results in a *set* of new rules replacing the original rule and—from a programming point of view—establishes a case distinction.

Suppose a rule with pattern p^n . Then the examples whose inputs match p^n have to be partitioned into a minimum number of at least two subsets and p^n

³ Note that for functions with arity > 1 inputs are sequences $i^n, n > 1$ of terms. We may consider such a sequence as *one* term by assuming a distinguished constructor symbol as root and the i^n as direct subterms.

has to be replaced by the lggs of the inputs of the respective subsets. It has to be assured that no two of the new lggs unify. This is done as follows: First a position u is chosen at which a variable stands in p^n . Since p^n is the lgg of all inputs matching it it holds that at least two inputs have different constructor symbols at position u . Then resp. all example inputs with the *same* constructor at position u are taken into the same subset. This leads to a partition of the example inputs. Finally, for each subset the lgg is computed. The new lggs do not unify, since they have different constructors at at least one position.

Possibly different positions of variables in pattern p^n lead to different partitions. Then all partitions and the corresponding sets of specialized patterns are generated. Each new pattern determines the lhs of a new rule. The corresponding initial rhss are computed as lggs of the respective outputs as described in Sect. 3.3. Since the refined patterns subsume fewer examples, the number of variables in the initial rhss which are not contained in the corresponding lhs decreases with each refinement step.

For example, let

1. $Reverse([]) = []$, 2. $Reverse([a]) = [a]$, 3. $Reverse([b]) = [b]$,
4. $Reverse([a, b]) = [b, a]$, 5. $Reverse([b, a]) = [a, b]$

be some examples for the *Reverse*-function. The pattern of the initial rule is simply a variable X , since the example input terms have no common root symbol. Hence, the unique position at which the pattern contains a variable and the example inputs different constructors is the root position. The first example input consists of only the constant $[]$ at the root position. All remaining example inputs have the constructor *cons* as root. I.e., two subsets are induced by the root position, one containing the first example, the other containing all remaining examples. The lggs of the example inputs of these two subsets are $[]$ and $[X|Xs]$ resp. which are the patterns of the two successor rules.

Introducing Function Calls The second method to generate successor sets for an unfinished rule with pattern p^n for a target function F is to replace its rhs by a call to a defined function F' , i.e. by a term $F'(R_1(p^n), \dots, R_m(p^n))$. Each R_i denotes a new introduced (sub)function. This finishes the rule, since now the rhs does not longer contain variables not contained in the lhs. In order to get a rule leading to a *correct* hypothesis, for each example equation $F(i^n) = o$ of function F whose input i^n matches p^n with substitution σ must hold: $F'(R_1(p^n), \dots, R_m(p^n))\sigma \xrightarrow{!} o$. This holds if for each output o an example equation $F'(i'_1, \dots, i'_m) = o$ of function F' exists such that $R_i(p^n)\sigma \xrightarrow{!} i'_i$ for each R_i and i'_i . Thus, if we find example equations of F' with outputs o , then we abduce example equations $R_i(i^n) = i'_i$ for the new subfunctions R_i and induce them from these examples. Provided, the final hypothesis is correct for F' and all R_i then it is also correct for F . To assure termination it must hold $i^{m'} < i^n$ according to any reduction order $<$ if the function call is recursive.⁴

⁴ Assuring decreasing arguments is more complex if mutual recursion is allowed.

Introducing Subfunctions The last method to generate successor equations can be applied, if all outputs o of the inputs matching the pattern of the considered unfinished rule have the same constructor c as roots. Let c be of arity m then the rhs of the rule is replaced by the term $c(Sub_1(p^n), \dots, Sub_m(p^n))$ where each Sub_i denotes a new introduced defined (sub)function. This finishes the rule since all variables from the new rhs are contained in the lhs. The examples for the new subfunctions are abduced from the examples of the current function as follows: If $o|_i$ are the i th subterms of the outputs o , then the equations $Sub_i(i^n) = o|_i$ are the example equations of the new subfunction Sub_i . Thus, correct rules for Sub_i compute the i th subterm of the outputs o such that the term $c(Sub_1(p^n), \dots, Sub_m(p^n))$ normalizes to the outputs o .

Remark As described in Sect. 3.4, (recursive) calls to defined functions specified by examples are only introduced at the root of a rhs (since such calls are introduced by replacing an unfinished rhs). Of course, generally, function calls can occur at any position in a rhs, compare for example the recursive definition for *Init* in Fig. 1. The reason why e.g. *Init* can be induced by our approach though function calls are only introduced at root positions is that deeper positions are (indirectly) considered as consequence of subprogram introduction as described in Sect. 3.4. Rhs root positions of such subprograms correspond to deeper positions of the rhs of the rule calling these subprograms.

4 Experimental Results and Evaluation

We have implemented a prototype of the described algorithm in the reflective programming language Maude [10]. The implementation includes two extensions: First, example equations may contain variables such that the amount of example equations needed to specify a target function decreases. This is advantageous for the specifier as well as it leads to smaller induction times. Second, different variables within a pattern can be tested for equality. This establishes—besides pattern refinement—a second form of case distinction.

Tab. 1 lists experimental results. The first column lists the names of the induced target functions, the second the names of additionally specified BK functions, the third the number of given examples (for target functions), the fourth the number of automatically introduced recursive subfunctions, the fifth the maximal number of calls to defined functions within one rule, the sixth the times in seconds consumed by the induction. Note that the example equations contain variables if possible (except for the *Add*-function); cp. Fig. 1. The experiments were performed on a Pentium 4 with Linux and the Maude 2.3 interpreter.

All induced programs compute the intended functions. *Length*, *Last*, *Reverse*, and *Member* are the well known list functions. *Reverse* has been specified twice, first with *Snoc* as BK which inserts an element at the end of a list and second without BK. The second case (see Fig. 1 for given data and computed solution), is an example for the capability of automatic subfunction introduction and nested calls of defined functions. *Odd* is a predicate and true for odd natural numbers,

target functions	bk funs	#expl	#subfuns	#funcalls	times
<i>Length</i>	/	3	0	1	.012
<i>Last</i>	/	3	0	1	.012
<i>Odd</i>	/	4	0	1	.012
<i>ShiftL</i>	/	4	0	1	.024
<i>Reverse</i>	<i>Snoc</i>	4	0	2	.024
<i>Even, Odd</i>	/	3, 3	0	1	.028
<i>Mirror</i>	/	4	0	2	.036
<i>Take</i>	/	6	0	1	.076
<i>Insertion Sort</i>	<i>Insert</i>	5	0	2	.160
<i>PlayTennis</i>	/	14	0	0	.260
<i>Add</i>	/	9	0	1	.264
<i>Member</i>	/	13	0	1	.523
<i>Reverse</i>	/	4	2	3	.790
<i>Quick Sort</i>	<i>Append, P₁, P₂</i>	6	0	5	63.271

Table 1. Some inferred functions

false otherwise. The solution contains two base cases (one for 0, one for 1) and in the recursive case, the number is reduced by 2. In the case where both *Even* and *Odd* are specified as target functions, both functions of the solution contain one base case for 0 and a call to the other function reduced by 1 as the recursive case. I.e. the solution contains a mutual recursion. *ShiftL* shifts a list one position to the left and the first element becomes the last element of the result list. The induced solution simply “bubbles” the first element position by position through the list. *Mirror* mirrors a binary tree. *Take* keeps the first n elements of a list and “deletes” the remaining elements. This is an example for a function with two parameters where both parameters are reduced within the recursive call. *Insertion Sort* and *Quick Sort* resp. are the well known sort algorithms. The five resp. six well chosen examples as well as the additional examples to specify the BK functions are the absolute minimum to generate correct solutions. The solution for *Insertion Sort* has been generated within a time that is not (much) higher as for the other problems, but when we gave a few more examples, the time to generate a solution explodes. The reason is, that all outputs of lists of the same length are equal such that many possibilities of matching the outputs in order to find recursive calls exist. The number of possible matches increases exponentially with more examples. The comparatively very high induction time for *Quick Sort* results from the many examples needed to specify the BK functions and from the complex calling relation between the target function and the BK functions. P_1 and P_2 are the functions computing the lists of smaller numbers and greater numbers resp. compared to the first element in the input list. For *Add* we have a similar problem. First of all, we have specified *Add* by ground equations such that more examples were needed as for a non-ground specification. Also for *Add* holds, that there are systematically equal outputs, since, e.g., $Add(2, 2)$, $Add(1, 3)$ etc. are equal and due to commutativity. Finally, *PlayTennis* is an attribute vector concept learning example from Mitchell’s machine learning text book [11]. The 14

training instances consist of four attributes. The five non-recursive rules learned by our approach are equivalent with the decision tree learned by ID3 which is shown on page 53 in the book. This is an example for the fact, that consistent learning of decision trees is a subproblem of inducing functional programs.

5 Conclusions and Further Research

We described a method to induce functional programs represented as confluent and terminating CSs. The method is inspired by classical and recent analytical approaches to the fast induction of functional programs. One goal was to overcome the drawback that “pure” analytical approaches do not facilitate the use of BK and generally have relatively restricted hypothesis languages and on the other side to keep the analytical approach as far as possible in order to be able to induce more complex functions in a reasonable time. This has been done by applying a search in a more comprehensive hypothesis space but where the successor functions are data-driven and not generate-and-test based, such that the number of successors is more restricted and the hypothesis space is searched in a controlled manner. Though the successor functions are data-driven, the search is complete and only favors hypotheses inducing fewer partitions but applies no further heuristics to estimate, how many partitions the final hypothesis will have. Developing such heuristics will be one of the further research topics.

References

1. Biermann, A.W., Guiho, G., Kodratoff, Y., eds.: Automatic Program Construction Techniques. Collier Macmillan (1984)
2. Flener, P., Yilmaz, S.: Inductive synthesis of recursive logic programs: Achievements and prospects. *Journal of Logic Programming* **41**(2-3) (1999) 141–195
3. Kitzelmann, E., Olsson, R., Schmid, U., eds.: Proceedings of the ICML 2005 Workshop Approaches and Applications of Inductive Programming. (2005)
4. Olsson, R.: Inductive functional programming using incremental program transformation. *Artificial Intelligence* **74**(1) (1995) 55–83
5. Summers, P.D.: A methodology for LISP program construction from examples. *Journal ACM* **24**(1) (1977) 162–175
6. Kitzelmann, E., Schmid, U.: Inductive synthesis of functional programs: An explanation based generalization approach. *Journal of Machine Learning Research* **7** (2006) 429–454 Special topic on Approaches and Applications of Inductive Programming.
7. Flener, P.: Inductive logic program synthesis with DIALOGS. In Muggleton, S., ed.: Proceedings of ILP’96, Springer (1997) 175–198
8. Malerba, D.: Learning recursive theories in the normal ILP setting. *Fundamenta Informaticae* **57**(1) (2003) 39–77
9. Plotkin, G.D.: A note on inductive generalization. In: Machine Intelligence. Volume 5. Edinburgh University Press (1969) 153–163
10. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: The maude 2.0 system. In Nieuwenhuis, R., ed.: Rewriting Techniques and Applications (RTA 2003). Number 2706 in LNCS, Springer (2003) 76–87
11. Mitchell, T.M.: Machine Learning. McGraw-Hill Higher Education (1997)

Integrating Analogical and Inductive Learning at Different Levels of Generalization

Helmar Gust, Ulf Krumnack, Kai-Uwe Kühnberger, Angela Schwering
{hgust | krumnack | kkuehnbe | aschweri}@uos.de

Institute of Cognitive Science, University of Osnabrück, Germany

Abstract. Several inductive and analogical learning methods have been developed in the past and applied to tasks such as concept formation and classification. However, it seems that the learning of general principles and abstract coherences can hardly be achieved by these methods when taken separately. In this paper, we argue that inductive and analogical learning address complementary tasks and therefore can be combined in a prolific manner. We present a three-level stage model in which analogical and inductive elements are used to generalize and integrate knowledge from different domains. Via analogy, commonalities of two different domains are identified, and thereby hints for abstract concepts are provided. Inductive methods are used to refine these concepts - and by transfer into new domains they are further elaborated. Our approach is applied to demonstrate how general physical principles can be learned from a series of examples from astronomy, nuclear physics, classical mechanics, and electrodynamics.

1 Introduction

In psychology, cognitive science, and artificial intelligence, learning has been discussed in many different facets and diverse learning strategies have been proposed. However, none of these approaches can explain how humans extract abstract principles across various domains. For example, general physical principles such as the “equilibrium of forces” or complex systems such as a “central force system” can be discovered in many different domains, but cannot be generalized via simple induction, because the concrete examples in which these general principles occur differ due to context and domain differences (e.g. gravitational force systems in astronomy or electromagnetic force systems have not much in common except the basic principle of a general force system).

This paper proposes an approach explaining how general physical principles can be learned and understood. We propose a learning-by-levels strategy that combines two learning mechanisms: One is analogical learning which is used to compare the (sometimes rather) different domains. The following example illustrates how different such domains can be: Consider the gravity force which originates from the sun due to its mass, the tractive force that a chain conveys to the seats of a merry-go-round or the coulomb force between nucleus and electron. These are all attracting forces, but have completely different reasons. The other learning mechanism is inductive learning which can generalize over a large amount of data samples and propose abstract generalizations such as general principles.

The remainder of this the paper is structured as follows: The second section shortly explains the differences between analogical and inductive learning (from a cognitive science perspective). Afterwards we introduce heuristic-driven theory projection (HDTP), our model for computing analogies. Section four describes the learning-by-levels strategy and explains how analogical and inductive learning interact. In section 5 we outline the learning process step-by-step with several examples.

2 Induction and Analogies as Different Approaches to Learning

Inductive learning and analogical learning are the constituents on which our learning-by-levels approach is built.

Inductive Learning creates knowledge via generalization over a large set of data samples which share common properties, facts or laws and deduces some general principles. The likeliness of inductive inferences increases with the number of valid cases. Our inductive learning strategy differs in some respect from the classical term “induction” as used in AI and machine learning: While induction requires a (rather) large set of data samples to draw sound conclusions, humans apply the inductive strategy already with a limited number of examples (e.g. four or five cases). Inductive learning is typically applied in one of the following three situations. (1) *Quantitative refinement*: induction refines the conceptualization of a domain. From the observation that water freezes in an environment of 0°C , -10°C and -20°C , we conclude that it freezes for all temperatures less than 0°C . Based on the refinement we make a general statement for a set of cases or intervals. (2) *Recognition of qualitative regularities*: Induction is used to find regularities across a set of data samples and to identify a general law which holds across all data samples. (3) *Category learning and concept formation*: Induction is the main driver for category learning or concept formation. Once a common pattern is recognized across a number of samples, these samples are grouped together and constitute examples for one category [1].

Analogical Learning differs from inductive learning: It typically requires only two domains - a source and a target domain. Analogies aim to identify structural commonalities, analogous relations or objects playing the same roles [2]. These commonalities are captured in an abstract generalization (section 3), however, this generalization is not assumed to be generally true, but only for the specific source and target domain. Analogical learning is typically applied across different domains. The purpose of analogies is to adapt knowledge available about the source such that it can be applied to the target in a way that new analogous inferences can be drawn. Learning based on analogy have been discussed extensively in literature. For instance, Klenk and Forbus apply Gentner’s Structure Mapping Engine [3, 4] to solve qualitative reasoning problems in physics by analogy [5]. This approach is weakly related to our approach, however, Klenk and Forbus only focus on the first level of analogical learning (compare figure 2 for the different stages). In this paper, we will restrict our considerations to the analogy model HDTP [6], which is a formal-logic approach for analogy-making. Although other approaches in machine learning such as transfer learning or multi-level learning [7, 8] have similarities in spirit with our approach to analogical learning, they use fundamentally different techniques (e.g. probabilistic or neural) and are incompatible with our logical approach.

Although analogical learning and inductive learning differ, they do not contradict to each other and can be applied within one overall learning strategy. Inductive learning can generalize commonalities, but it cannot detect commonalities across different domains which are not directly comparable. This is one of the strengths of analogical reasoning. On the other hand, analogical reasoning compares only two domains and therefore cannot propose general laws across many domains that might lead to general physical principles. Inductive learning is required for this.

3 Analogical Learning with Heuristic-Driven Theory Projection

Heuristic-Driven Theory Projection (HDTP) is a formally sound theory for computing analogical relations between a source and a target domain. HDTP computes analogies not only by associating concepts, relations, and objects, but also complex rules and facts between target and source domain. We refer to [6] for the specification of the syntactic, semantic, and algorithmic properties of HDTP.

Syntactically, HDTP is defined on the basis of a many-sorted first-order language. First-order logic is used in order to guarantee the necessary expressive power of the theory. An important assumption is that analogical reasoning crucially contains a generalization (or abstraction) process. In other words, the identification of common properties or relations is represented by a generalization of the input of source and target. Mathematically this can be modeled by an extension of the theory of anti-unification [9, 10], a mathematically sound account describing the possibility of generalizing terms of a given language using substitutions. More precisely, an anti-unification of two terms t_1 and t_2 can be interpreted as finding a generalized term t (or structural description t) of t_1 and t_2 which may contain variables, together with two substitutions θ_1 and θ_2 of variables, such that $t\theta_1 = t_1$ and $t\theta_2 = t_2$. Because there are usually many possible generalizations, anti-unification tries to find the most specific one (cp. also [11]).

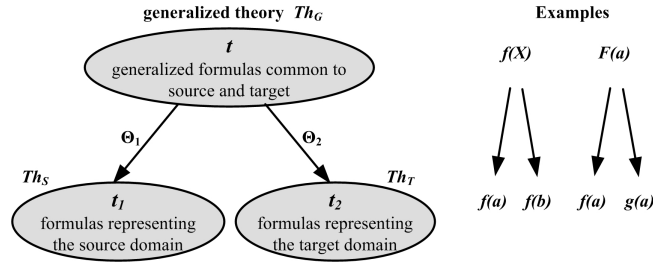


Fig. 1. Establishing the analogical relation between the source theory Th_S and the target theory Th_T and constructing the general theory Th_G .

Given two theories Th_S and Th_T modeling source and target domain as input, the HDTP algorithm computes the analogy. Table 1 shows the algorithm, which contains the following steps:

Input: A theory Th_S of the source domain and a theory Th_T of the target domain represented in a many-sorted predicate logic language.

Output: A generalized theory Th_G such that the input theories Th_S and Th_T can be reestablished by substitutions.

Algorithm: Selection and generalization of facts and rules. Select an axiom from Th_T according to a heuristics h . In HDTP, this heuristics could select formulas according to their complexity, i.e. prefer less complex literals before complex rules. Afterwards, select an axiom from Th_S and construct a generalization (together with corresponding substitutions).

Optimize the generalization w.r.t. a given heuristics and update Th_G w.r.t. the result of this process. The heuristics used by HDTP orders the anti-instances according to the complexity of their substitutions (e.g. length of substitutions).

Transfer (project) facts and laws of Th_S to Th_T provided they are not generalized yet. Test (using an oracle) whether the transfer is consistent with Th_T . This can be done via experiments or using world knowledge in a database.

```

T = axioms of the target domain sorted by a heuristics h
S = axioms of the source domain
G = empty list of axioms of generalized theory
 $\Theta_1 = \Theta_2 =$  empty substitution
 $Th_T^{Ah} = Th_T$ 
FOR  $\phi \in T$ 
  T = normal_form(T)
  SELECT  $\psi \in S$ 
     $\psi = normal\_form(\psi)$ 
    IF not same_structure( $\phi, \psi$ ) REJECT
    SELECT  $(\xi, \Theta_1, \Theta_2) \in anti\_instances(\phi, \psi, \Theta_1, \Theta_2)$ 
    WITH  $\xi$  best according to a heuristics  $h'$ 
    IF  $h'(\xi) >$  a given threshold
      ADD  $\xi$  to G
      ADD  $\xi \Theta_2$  to  $Th_T^{Ah}$ 
      REMOVE  $\psi$  from S
    ELSE FAIL
  END FOR
FOR  $\psi \in S$ 
   $\phi = transfer(\psi, \Theta_1, \Theta_2)$ 
  IF  $Th_T^{Ah} \vdash \neg \phi$  CONTINUE
  IF oracle( $\phi$ ) = FALSE CONTINUE
  ADD  $\phi$  to  $Th_T^{Ah}$ 
  ADD generalize( $\psi, \Theta_1$ ) to G
END FOR

```

Table 1. The algorithm HDTP-A generalizing two theories Th_S and Th_T .

For our purpose it is important to stress the two ways of analogical learning: (1) *Learning by analogical transfer* refers to new knowledge gained in the target domain. Usually, the source domain is the more familiar domain described by a richer theory. By drawing the analogy, HDTP transfers knowledge of the source domain to the target domain. (2) *Learning by generalization* is a by-product of the analogy-making process: Via the theory of anti-unification, HDTP computes a generalized theory which con-

tains the commonalities between Th_S to Th_T . The “learning by generalization” is the important step for our proposed stage model.

4 Learning by Levels

None of the above described learning theories can explain the whole process of learning and understanding general principles across different domains. Analogical reasoning can compare different domains by analyzing their common structure, but the analogical generalizations hold only for the specific source and target domain. On the other hand, inductive learning seems to be inadequate, because it can find regularities only from data samples containing the same (type of) information. This means, that induction cannot identify commonalities across different domains or contexts, because these often differ in their superficial properties and commonalities exist only with respect to an analogous structure. Inductive learning requires a large number of examples to supply enough evidence for the generalization. Unlike analogical reasoning, we consider inductive generalizations as universally quantified laws. Therefore we suggest to enhance analogical learning with inductive mechanisms and propose three levels of learning.

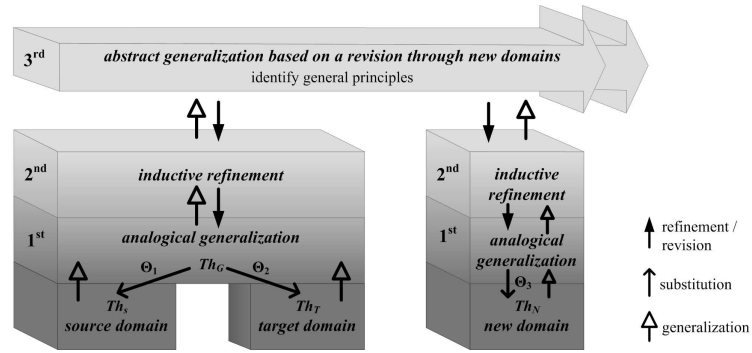


Fig. 2. A stage model for learning by levels: Integrating analogical and inductive learning.

Figure 2 illustrates the stage model of the learning process. The basis for the learning process are the domain theories. In the case of analogical learning these are a source domain Th_S and a target domain Th_T (left side of the figure). The three levels (see also [12]) contribute to the learning process as follows:

- 1. Level:** At the lowest level of learning the source and the target domain are compared via analogy to identify common structures between the different domains. The commonalities are generalized in a general theory Th_G . It is important to stress that the knowledge contained in Th_G is only general in so far as it applies to the source and target domain, but not generally across many or even all domains. Commonalities identified at this learning stage are existentially quantified within the two domains, i.e. there exists some parameter configuration for which the analogical generalization holds.

- 2. Level:** At this level of learning the analogy proposed at the 1. level has to be tested, because the analogical knowledge transfer might be true only for prototypical situations. Therefore the knowledge transfer must be tested for various cases in the target domain. The process runs through a number of different refinement steps:
- To investigate the range parameters for which the analogy holds, new knowledge inferred by the analogy (the analogical transfer) is tested with experiments for a large number of different parameters. Via induction we can infer certain intervals for which the analogy holds.
 - If the analogy was shown to be wrong in some cases, the parameter configuration is selected for which the analogy no longer holds.
 - The results are projected back to the source, to test experimentally whether the theory holds for this parameter configuration. If necessary, the source domain theory is adjusted.
 - New findings in the source domain are included in the analogy and projected back to the target domain.

In section 5, we explain the analogy between the relation of the moon to the earth and the solar system. We show that an abstract generalization of a central force system holds for the earth-moon relation, the solar system, and for new domains such as the rotating seats of a merry-go-round. In all three domains exists an orbiting element and an attracting force. The inductive refinement compares possible values for the intensity of the attracting forces and the velocity: if the velocity is too small or too fast, the rotating object leaves the orbit. However, the possible parameter settings might differ in the solar system and the merry-go-round.

The outcome is a refined analogy. This level is called inductive refinement, because it applies induction for the parameter check. While the generalized facts and laws on the first level were existentially quantified (a parameter-configuration exists for which the analogical generalization holds), the refinement at the second level can be considered to be universally quantified, but still restricted to Th_S and Th_T .

- 3. Level:** The aim of this level is the identification of general principles - in our example these are general principles in physics such as the equilibrium of forces or general systems such as the central force system. This learning step goes beyond the comparison of two domains and requires inductive learning (in the sense of section 2) on top of analogical comparison. At this level, the learning process starts with an intuitive hypothesis about a general principle and compares this iteratively with other domains to gain more confidence. This process leads to a revision of the general principle, which in the end contains only those elements important for the principle and no domain-specific attributes. Such general principles are considered to be universally quantified across different domains.

The three levels proposed above outline the different mechanisms applied (pure analogical learning or learning based on inductive strategies). However, we do not claim that humans learn sequentially in such levels. The identification of an abstract generalization is by no means a straightforward process. It is driven by intuitive hypotheses of possible generalizations and search for new domains supporting this hypothesis. If the new domain does not support the hypothesis, the abstract generalization either does not apply to this domain or it can be revised. A revision takes place, if a meaningful

generalization can be found, which applies to the old and the new domains. This iterative process leads to the elaboration of a general principle. Learning is a process in which different learning mechanisms interact and are iteratively repeated to refine and revise knowledge on the various abstraction levels. We present a model explaining how to learn general principles based on the analogical generalization produced by HDTP.

5 Illustration with Examples

This section explains the process of learning the general physical principle “central force system” using various examples. We use HDTP to compute analogical generalizations and explain how the abstract generalization can be reached.

5.1 Analogical Generalization

Our first example compares the relation between the earth and its moon with the solar system (i.e. the relation between sun and earth). The observation that the earth has a gravitational effect and attracts objects and that the moon revolves around the earth in a more or less circular motion leads to the following conceptualization (table 2).

Table 2. The formalization of the relation between earth and moon and of the solar system.

Formalization of the earth-moon relation (Th_S)	Formalization of the sun-earth relation (Th_T)
types $real, object, time$	types $real, object, time$
constants $earth : object, moon : object$	constants $sun : object, earth : object$
functions $mass : object \rightarrow real \times \{kg\}$ $dist : object \times object \times time \rightarrow real \times \{m\}$ $gravity : object \times object \times time \rightarrow real \times \{N\}$ $centrif : object \times object \times time \rightarrow real \times \{N\}$	functions $mass : object \rightarrow real \times \{kg\}$ $dist : object \times object \times time \rightarrow real \times \{m\}$ $gravity : object \times object \times time \rightarrow real \times \{N\}$ $centrif : object \times object \times time \rightarrow real \times \{N\}$
facts $mass(earth) > mass(moon) > 0$ $\forall t : time : gravity(moon, earth, t) > 0$ $\forall t : time : dist(moon, earth, t) > 0$	facts $mass(sun) > mass(earth) > 0$ $\forall t : time : gravity(earth, sun, t) > 0$ $\forall t : time : dist(earth, sun, t) > 0$
laws 1. $\forall t : time, o_1 : object, o_2 : object :$ $mass(o_1) > 0 \wedge mass(o_2) > 0$ $\rightarrow gravity(o_1, o_2, t) > 0$ 2. $\forall t : time, o_1 : object, o_2 : object :$ $dist(o_1, o_2, t) > 0 \wedge gravity(o_1, o_2, t) > 0$ $\rightarrow centrif(o_1, o_2, t) = -gravity(o_1, o_2, t)$ 3. $\forall t : time, o_1 : object, o_2 : object :$ $0 < mass(o_1) < mass(o_2) \wedge$ $dist(o_1, o_2, t) > 0 \wedge centrif(o_1, o_2, t) < 0$ $\rightarrow revolves_around(o_1, o_2)$	laws 1. $\forall t : time, o_1 : object, o_2 : object :$ $mass(o_1) > 0 \wedge mass(o_2) > 0$ $\rightarrow gravity(o_1, o_2, t) > 0$ 2. $\forall t : time, o_1 : object, o_2 : object :$ $dist(o_1, o_2, t) > 0 \wedge gravity(o_1, o_2, t) > 0$ $\rightarrow centrif(o_1, o_2, t) = -gravity(o_1, o_2, t)$ 3. $\forall t : time, o_1 : object, o_2 : object :$ $0 < mass(o_1) < mass(o_2) \wedge$ $dist(o_1, o_2, t) > 0 \wedge centrif(o_1, o_2, t) < 0$ $\rightarrow revolves_around(o_1, o_2)$

The gravitational effect of a heavenly body depends on its mass. Due to the earth’s mass there exists terrestrial gravity which attracts the moon. Moreover, we can observe that earth and moon have a positive (more or less constant) distance over time and the moon does not “fall down” to the earth. If the terrestrial gravity attracts the moon, but it stays distant from the earth, there must exist another force - a counter force to the

gravitational force - which is called centrifugal force (abbreviated by *centrif* in the formalization). From the facts that moon and earth have a positive distance, the earth's mass is greater than that of its moon and there exists some centrifugal force we conclude that the moon revolves around the earth.

The rotational relationship between the earth and its moon is actually analogous to the relationship between the sun and the earth. The analogy model HDTP computes the analogical relation via comparing the source theory describing earth/moon to the target theory describing sun/earth and computes a generalized theory (table 3).

Table 3. The analogical generalization of the earth/moon relation and the sun/earth relation.

```

types
  real, object, time
constants
  X : object, Y : object
functions
  mass : object → real × {kg}
  dist : object × object × time → real × {m}
  gravity : object × object × time → real × {N}
  centrif : object × object × time → real × {N}
facts
  mass(X) > mass(Y) > 0
  ∀t : time : gravity(X, Y, t) > 0
  ∀t : time : dist(X, Y, t) > 0
laws
  1. ∀t : time, o1 : object, o2 : object :
     mass(o1) > 0 ∧ mass(o2) > 0
     → gravity(o1, o2, t) > 0
  2. ∀t : time, o1 : object, o2 : object :
     dist(o1, o2, t) > 0 ∧ gravity(o1, o2, t) > 0
     → centrif(o1, o2, t) = -gravity(o1, o2, t)
  3. ∀t : time, o1 : object, o2 : object :
     0 < mass(o1) < mass(o2) ∧
     dist(o1, o2, t) > 0 ∧ centrif(o1, o2, t) < 0
     → revolves_around(o1, o2)

```

As analogies refer to structural commonalities, HDTP analyzes common relations and common roles and detects, that earth_{Th_S} and sun play the same role: they are the objects with the greater mass, they have a positive distance to the orbiting objects and have a gravitational effect on them. Moon and earth_{Th_T} play the same role as the orbiting objects. Also the laws are generalized. It is important to stress that analogy is required to find such role commonalities: a superficial search for commonalities would have aligned earth_{Th_S} with earth_{Th_T}.

5.2 Abstract Generalization

The relationship between earth/moon or sun/earth is not unique in our universe. Actually, this relation holds for all heavenly bodies. Based on our analogy (section 5.1) humans are likely to come up with a hypothesis about a universal law of gravitation or an even more abstract law about the relation of forces and a resulting rotation. At the level of abstract generalization, the learner with the above described hypothesis searches for

other domains to support or reject his hypothesis. We distinguish two effects: On the one hand, this is a refinement process of the abstract generalization. New domains either support the hypothesis and increase the likeliness of the abstract generalization or they do not support the hypothesis. If only parts of the hypothesis are rejected, it can be revised. If the hypothesis as a whole is rejected, this can mean that the new domain was no example for the abstract generalization. On the other hand, the hypothesis of an abstract generalization also determines the way the learner conceptualizes the new domain. The following description of the Rutherford Atom is an example for this case.

The central body system. The analogical generalization describes a gravitation system with one center of gravitation, one orbiting body and two forces: the gravitational force is an attracting force and the centrifugal force is its counter force. The conceptualization of the gravitation system influenced mainly Rutherford’s construction of the atom model. Based on the idea of a gravitation system Rutherford developed his atom model with electrons revolving around the nucleus (table 4, left side).

Table 4. The hypothesis, that the gravitation system is a general principle, guides Rutherford’s conceptualization of an atom (left side) and leads to a revised general principle: the “central body system” (right side).

Formalization of the Rutherford Atom	Abstract generalization: Central body system
<p>types <i>real, object, time</i></p> <p>constants <i>nucleus : object, electron : object</i></p> <p>functions <i>mass : object → real × {kg}</i> <i>dist : object × object × time → real × {m}</i> <i>electric_charge : object → real × {eV}</i> <i>coulomb : object × object × time → real × {N}</i> <i>centrif : object × object × time → real × {N}</i></p> <p>facts <i>mass(nucleus) > mass(electron) > 0</i> <i>electric_charge(electron) < 0</i> <i>electric_charge(nucleus) > 0</i> $\forall t : time : coulomb(electron, nucleus, t) > 0$ $\forall t : time : dist(electron, nucleus, t) > 0$</p> <p>laws 1. $\forall t : time, o_1 : object, o_2 : object :$ <i>electric_charge(o₁) > 0</i> $\wedge electric_charge(o_2) < 0$ $\rightarrow coulomb(o_1, o_2, t) > 0$ 2. $\forall t : time, o_1 : object, o_2 : object :$ <i>dist(o₁, o₂, t) > 0</i> \wedge <i>coulomb(o₁, o₂, t) > 0</i> $\rightarrow centrif(o_1, o_2, t) = -coulomb(o_1, o_2, t)$ 3. $\forall t : time, o_1 : object, o_2 : object :$ $0 < mass(o_1) < mass(o_2) \wedge dist(o_1, o_2, t) > 0 \wedge$ <i>centrif(o₁, o₂, t) < 0</i> $\rightarrow revolves_around(o_1, o_2)$</p>	<p>types <i>real, object, time</i></p> <p>constants <i>X : object, Y : object</i></p> <p>functions <i>mass : object → real × {kg}</i> <i>dist : object × object × time → real × {m}</i> <i>F : object × object × time → real × {N}</i> <i>centrif : object × object × time → real × {N}</i></p> <p>facts <i>mass(X) > mass(Y) > 0</i> $\forall t : time : F(X, Y, t) > 0$ $\forall t : time : dist(X, Y, t) > 0$</p> <p>laws 1. $\forall t : time, o_1 : object, o_2 : object :$ <i>Feature1(o₁)</i> \wedge <i>Feature2(o₂)</i> $\rightarrow F(o_1, o_2, t) > 0$ 2. $\forall t : time, o_1 : object, o_2 : object :$ <i>dist(o₁, o₂, t) > 0</i> \wedge <i>F(o₁, o₂, t) > 0</i> $\rightarrow centrif(o_1, o_2, t) = -F(o_1, o_2, t)$ 3. $\forall t : time, o_1 : object, o_2 : object :$ $0 < mass(o_1) < mass(o_2) \wedge$ <i>dist(o₁, o₂, t) > 0</i> \wedge <i>F(o₁, o₂, t) < 0</i> $\rightarrow revolves_around(o_1, o_2)$</p>

Many analogous commonalities exist here: There are two types of objects, a nucleus and electrons. The nucleus has a greater mass than the electrons. They attract each other, however the attracting coulomb force is caused by the opposite electric charges of electron and nucleus. The general principle of the “gravitation system” can be applied to the atom using the mechanism of analogy and leads to the revised general principle “central

Table 5. Formalization of the merry-go-round and the electron in a homogeneous magnetic field.

Formalization of the merry-go-round	Electron in a homogeneous magnetic field
types $real, object, location, time$	types $real, object, location, time$
constants $seat : object, center : location$	constants $electron : object, center : location$
functions $mass : object \rightarrow real \times \{kg\}$ $dist : object \times location \times time \rightarrow real \times \{m\}$ $tractivef : object \times object \times time \rightarrow real \times \{N\}$ $centrif : object \times location \times time \rightarrow real \times \{N\}$	functions $mass : object \rightarrow real \times \{kg\}$ $dist : object \times location \times time \rightarrow real \times \{m\}$ $lorentzf : object \times object \times time \rightarrow real \times \{N\}$ $centrif : object \times location \times time \rightarrow real \times \{N\}$
facts $mass(seat) > 0$ $\forall t : time : revolves_around(seat, center, t)$	facts $mass(electron) > 0$ $\forall t : time : revolves_around(electron, center, t)$
laws 1. $\forall t : time, o : object, l : location :$ $mass(o) > 0 \wedge dist(o, l, t) > 0 \wedge$ $revolves_around(o, l, t)$ $\rightarrow centrif(o, l, t) < 0$ 2. $\forall t : time, o : object, l : location :$ $dist(o, l, t) > 0 \wedge centrif(o, l, t) < 0$ $\rightarrow tractivef(o, l, t) = -centrif(o, l, t)$	laws 1. $\forall t : time, o : object, l : location :$ $mass(o) > 0 \wedge dist(o, l, t) > 0 \wedge$ $revolves_around(o, l, t)$ $\rightarrow centrif(o, l, t) < 0$ 2. $\forall t : time, o : object, l : location :$ $dist(o, l, t) > 0 \wedge centrif(o, l, t) < 0$ $\rightarrow lorentzf(o, l, t) = -centrif(o, l, t)$

body system” (table 4, right side). The atom domain supports the hypothesis that the gravitation system could be generalized across other domains and leads to a revised, more general concept which is not restricted to heavenly bodies and gravitational force.

The central force system. The revision process of the abstract generalization is continued by comparing the “central body system” with two other domains: a merry-go-round and an electron in a homogeneous magnetic field.

The merry-go-round (left of table 5) is a rotating system, but it differs from the previous examples with regard to the center: there is no *body* in the center of the circular motion. The seats are the revolving objects: the centrifugal force pushes them outside, away from the center. A tractive force (called *tractivef* in the formalization) is the attracting counter force and acts on the seats via the metal chains connecting the seats with the center of the merry-go-round.

The conceptualization of an electron in a homogeneous magnetic field (right of table 5) is also a rotational system: It can be observed that an electron in a magnetic field moves on a circular motion which has a center point. Therefore a centrifugal force acts on the electron. Since it moves continuously on the same orbit, a counter-force towards the center of the orbit must exist (called *lorentzf*). The examples merry-go-round and the electron in the magnetic field revise the abstract generalization of a “central force system” with regard to the central body. Now the learner recognizes that it is enough to have a central location of the orbit which does not need to be an object.

The General Principle. The general principle resulting from the revision process is a “central force system” explaining the relationship between an attracting force towards a center, a centrifugal force and an object revolving around the center.

Table 6 shows the formalization of the resulting abstract generalization. It contains only the information which is necessary for a general force system and reduces domain specific knowledge such as mass differences between bodies in the gravitation system or electric charges. Also it is irrelevant whether one observes the forces and the positive distance first or the revolving motion of the orbiting object. Figure 3 illustrates

Table 6. The abstract generalization of the general principle.

```

types
  real, object, location, time
constants
  O : object, L : location
functions
  mass : object → real × {kg}
  dist : object × location × time → real × {m}
  F : object × location × time → real × {N}
facts
  mass(O) > 0
  ∀t : time : F(O, L, t) ≠ 0
  ∀t : time : dist(O, L, t) > 0
laws
  1. ∀t : time, o : object, l : location :
     dist(o, l, t) > 0 ∧ F(o, l, t) ≠ 0
     → ∃counterforce : counterforce(o, l, t) = -F(o, l, t) ∧
  2. ∀t : time, o : object, l : location :
     dist(o, l, t) > 0 ∧ mass(o, t) > 0
     → (centrif(o, l, t) < 0 ↔ revolves_around(o, l))

```

the line of argument to understand the revision process of the abstract generalization. The earth/moon and the sun/earth relation as well as the Rutherford atom - all examples for the central body system - start from the discovery of the attracting force and conclude from the positive distance that there must exist a centrifugal force which induces a revolving motion. The merry-go-round and the electron in the homogeneous magnetic field start with the observation of a revolving object with a centrifugal force and conclude that there also must exist an attracting force towards the center of the orbit.

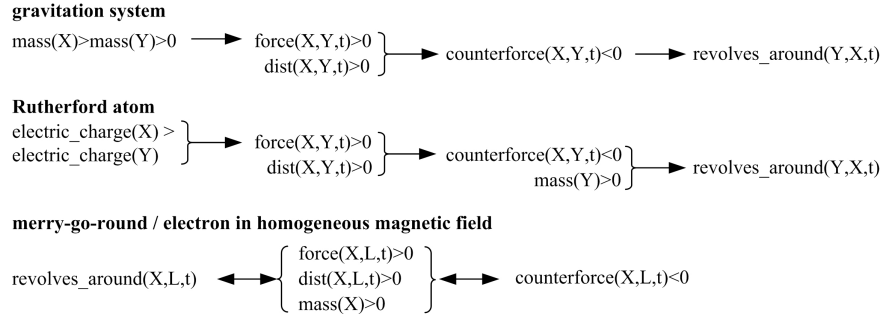


Fig. 3. The line of argumentation for the various examples leading to the abstract generalization

6 Conclusions and Future Work

Analogical reasoning is a fundamental construct of learning, because it is essential to compare commonalities across different domains. However, to reach abstract gener-

alizations such as general physical principles we need to enhance analogies with an inductive aspect. This leads to the establishment of generalizations which apply across various domains. This paper outlined three levels of learning and illustrated the step from the analogical generalization to the abstract generalization with a chain of example. The examples chosen in this paper lead to the general principle of a central force system and an abstract idea of force. The establishment of even more general principles of physics such as the equilibrium of forces or general laws of nature such as the conservation of energy follows the same learning strategy, but the generalization and revision process of the abstract generalization is driven further.

In a future application scenario, an e-learning tutor might use our stage model of learning to explain one domain via another domain which is easier to conceptualize and understand by the student. Analogous domains could be detected autonomously and related to a general principle. This teaching strategy is often used by human teachers. Future work will address the further development of a language for a formal specification of the learning stages and an algorithm for the abstract generalization and the revision process.

Acknowledgement

The work was supported by the German Research Foundation (DFG) through the project “Modeling of predictive analogies by Heuristic-Driven Theory Projection” (grant KU 1949/2-1).

References

1. Goldstone, R.L.: The role of similarity in categorization: Providing a groundwork. *Cognition* **52**(2) (1994) 125–157
2. Gentner, D.: The mechanism of analogical learning. In Vosniadou, S., Ortony, A., eds.: *Similarity and analogical reasoning*. Cambridge University Press, NY (1989) 199–241
3. Forbus, K.D., Hinrichs, T.: Companion cognitive systems: A step towards human-level AI. *AI Magazine* **27**(2) (2006) 83–95
4. Gentner, D.: Structure-mapping: A theoretical framework for analogy. *Cognitive Science* **7**(2) (1983) 155–170
5. Klenk, M., Forbus, K.D.: Cross domain analogies for learning domain theories. In Schwering, A., Krumnack, U., Kühnberger, K.U., Gust, H., eds.: *Analogies: Integrating Multiple Cognitive Abilities (AnICA07)*. Volume 5 of Publications of the Institute of Cognitive Science., Nashville, TN, USA, University of Osnabrück (2007) 3–8
6. Gust, H., Kühnberger, K.U., Schmid, U.: Metaphors and heuristic-driven theory projection (hdt). *Theoretical Computer Science* **354**(1) (2006) 98–117
7. Baxter, J.: A bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine Learning* **28**(7) (1997) 7–39
8. Caruana, R.: Multitask learning. *Machine Learning* **28**(41) (1997) 41–75
9. Plotkin, G.D.: A note on inductive generalization. *Machine Intelligence* **5** (1970) 153–163
10. Plotkin, G.D.: A further note on inductive generalization. *Machine Intelligence* **6** (1971)
11. Gust, H., Kühnberger, K.U., Schmid, U.: Anti-unification of axiomatic systems. Technical report, Institute for Cognitive Science, University of Osnabrück (2003)
12. Gust, H., Kühnberger, K.U.: Explaining effective learning by analogical reasoning. In Sun, R., Miyake, N., eds.: *28th Annual Conference of the Cognitive Science Society in cooperation with the 5th International Conference of Cognitive Science in the Asian-Pacific region (CogSci/ICCS)*, Lawrence Erlbaum (2006) 1417–1422