Creating a Cluster Hierarchy under Constraints of a Partially Known Hierarchy

Korinna Bade and Andreas Nürnberger
Otto-von-Guericke-University Magdeburg, D-39106 Magdeburg, Germany,
{korinna.bade,andreas.nuernberger}@ovgu.de

Abstract

Although clustering under constraints is a current research topic, a hierarchical setting, in which a hierarchy of clusters is the goal, is usually not considered. This paper tries to fill this gap by analyzing a scenario, where constraints are derived from a hierarchy that is partially known in advance. This scenario can be found, e.g., when structuring a collection of documents according to a user specific hierarchy. Major issues of current approaches to constraint based clustering are discussed, especially towards the hierarchical setting. We introduce the concept of hierarchical constraints and continue by presenting and evaluating two approaches using them. The approaches cover the two major fields of constraint based clustering, i.e. instance and metric based constraint integration. Our objects of interest are text documents. Therefore, the presented algorithms are especially fitted to work for these where necessary. Despite showing the properties and ideas of the algorithms in general, we evaluated the case of constraints that are unevenly scattered over the instance space, which is very common for real-world problems but not satisfyingly covered in other work so far.

1 Introduction

Lately, a lot of work on constraint based clustering has been published, e.g. [5, 7, 14, 15]. All these works aim at deriving a single flat cluster partition, even though they might use a hierarchical cluster algorithm. In contrast to them, we are interested in obtaining a hierarchical structure of nested clusters. This poses different requirements on the clustering algorithm.

There are many applications, in which a hierarchical cluster structure is more useful than a single flat partition. One such example is the clustering of text documents into a (personal) topic hierarchy. Such topics are naturally structured hierarchically. Furthermore, hierarchies can improve the access to the data for a user, if a large number of specific clusters is present, as the user can locate interesting topics step by step by several specializations.

After defining our hierarchical setting, we analyze how constraints can be used in the scope of hierarchical clustering (Sect. 2). In section 3, we review related work on constraint based clustering in more detail, trying to show different predominant concepts and their problems. We then present two different approaches for hierarchical constraint based clustering with special focus on text documents in section 4. These approaches are then evaluated in section 5 with different hierarchical datasets.

2 Hierarchical Constraint Based Clustering

To avoid confusion with other approaches of constraint based clustering as well as different opinions about the concept of hierarchical clustering, we use this section to define the current problem at hand from our perspective. Furthermore, we clarify the use of constraints in this setting.

As the algorithms presented here account to a semi-supervised learning problem, we first want to clarify the use of the terms *class* and *cluster*. Both terms have very similar meaning, describing a set of objects belonging together. However, *class* is usually used in supervised learning and *cluster* in unsupervised learning. As our setting lays in between, the usage of both terms seems appropriate. Here, we tend to use the term *class* to refer to the structure in the data that shall be uncovered, while *clusters* are the actual groupings of items derived by the algorithm. However, as a direct mapping from the derived *clusters* to the actual *classes* is sought, both terms are sometimes used interchangeable.

2.1 Problem Definition. We define our task at hand as a semi-supervised hierarchical learning problem. The goal of the clustering is to uncover a hierarchical class structure H that consists of a set of classes C, between which hierarchical relations $R_H = \{(c_1, c_2) \in C \times C | c_1 \geq_H c_2\}$ hold ($c_1 \geq_H c_2$ means that c_1 contains c_2 as a subclass). Thus, the combina-

tion between C and R_H represents the hierarchical tree structure of classes $H = (C, R_H)$. The data objects in O uniquely belong to one class in C. It is important to note that we specifically allow for the assignment of objects to intermediate levels of the hierarchy. Such an assignment is useful in many circumstances as a specific leaf class might not exist for certain instances. As an example consider a document giving an overview over a certain topic. As there might be several documents only describing a certain part of the topic and therefore forming several specific sub-classes, the document itself naturally fits into the broader class as the scope of its content is also broad. This makes the whole problem a true hierarchical problem. A straight-forward recursive application of flat partitioning algorithms is not sufficient. Therefore, we decided on using a bottom-up hierarchical agglomerative clustering as a basis. As an alternative, a top-down recursive partitioning approach could be used, if sophisticated techniques are integrated that estimate the number of clusters and are capable of leaving elements out of the partition (as done for noise detection).

The clustering algorithm will be constrained in deriving any cluster hierarchy H by making a part of the class hierarchy known to it, i.e. $H_k = (C_k, R_{Hk})$ with $C_k \subseteq C$ and $R_{Hk} \subseteq R_H$. Furthermore, some objects belonging to these classes are given, i.e. $O_k \subseteq O$. Please note that we assume for real world applications that C_k is smaller than C and O_k is smaller than O. Furthermore, we assume that we have at least one given object $o \in O_k$ for each given class $c \in C_k$. For higher level classes (i.e. classes that are not leaf nodes of the given class hierarchy), an object might not be assigned directly to the class but to at least one sub-class to fulfill this criterion. By H_k, the clustering algorithm is constrained to produce a hierarchical clustering that preserves the existing structure R_{Hk}, while discovering further clusters and extracting their relations to each other and to the classes in C_k, i.e. the constrained algorithm is supposed to refine the given structure by further data (see Fig. 1).

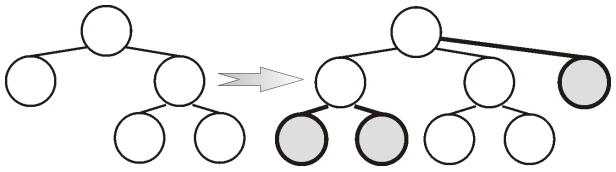

Figure 1: Hierarchy refinement/extension

2.2 Constraints. In constraint based clustering, the supervised knowledge is usually expressed by two sets of object pairs, the must-link constraints and the cannot-link constraints [14]. The pairs in the must-link-set describe objects that should be clustered in the same cluster, and the cannot-link set contains object pairs that should be separated. However, considering a hierarchical cluster structure, objects are linked over different hierarchy levels, which makes such an absolute constraint definition not appropriate. The drawbacks of this constraint definition for hierarchical clustering were also mentioned in [8]. The authors work with hierarchical agglomerative clustering and solve the problem by fixing the constraints to a certain dendrogram level. However, in most cases, it is not clear, on which dendrogram level the constraints should be applied.

We therefore suggested another definition of constraints that is suitable for hierarchical clustering tasks in [2], the must-link-before (MLB) constraints. Their goal is to stress the order, in which objects are linked. The MLB constraint set can be specified in two ways. In [2], we presented a hierarchy based formalization as

$$MLB = \{(o, \mathcal{S})\} = \{(o, (S_1, ..., S_m))\} \tag{2.1}$$

with $O_k = \bigcup_i S_i \cup o$ and $\forall i, j (i \neq j) : S_i \cap S_j = \emptyset$ and $\forall i : o \notin S_i$. Here, an element of the MLB constraint set is a pair of an object $o \in O_k$ with an ordered list $\mathcal{S}$ of m object sets. Each of these sets contains objects from O_k, to which o should be linked before all objects in the following sets. The order between the sets is important, while inside the sets it does not matter, which object is linked first. Each MLB constraint covers all objects in O_k. Therefore, application of the MLB constraints requires knowledge about all relations between the elements of O_k, which is true in our case of knowing a partial hierarchy H_k. In these cases, this formalization nicely condenses a large number of constraints.

For each $o \in O_k$ such a constraint can be extracted. An example is given in Fig. 2. Here, any object from class 4 should first be linked with all objects in 4. Then, it should be linked to objects from class 3 and 5, whereas the order does not matter. Finally, the object should be linked to all other objects, i.e objects from class 1 and 2. This leads to the MLB constraint shown on the right of Fig. 2.

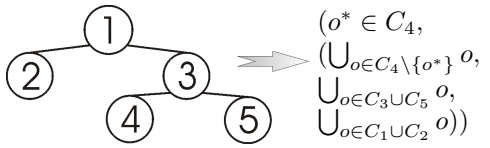

Figure 2: Example of a MLB Constraint Extraction

For a more general case, where hierarchical relations between some objects might be known, however,

without the knowledge of a complete hierarchy, this constraint formalization might not be applicable. For this case, an instance based formalization similar to the formalization in [11] is useful. Here, a MLB constraint set element expresses a relative comparison between three objects

$$MLB = \{(o_x, o_y, o_z)\}. \tag{2.2}$$

An element is interpreted as: o_x should be linked to o_y on a lower hierarchy level than o_x and o_z. This means that in any cluster in the cluster hierarchy that contains o_x and o_z o_y is also contained. However, there might be clusters, which only contain o_x and o_y but not o_z. An example constraint for the hierarchy in Fig. 2 is

$$(o_x \in C_4, o_y \in C_3, o_z \in C_1). \tag{2.3}$$

Considering single object triples instead of the complete hierarchical view on a known hierarchy is also closer to the initial idea of constraint based clustering, which assumed the unavailability of specific class labels. Every constraint formulation in the hierarchical view can be transformed into an equally expressive set of triple constraints.

3 Discussion of Related Work

Existing methods in constraint based clustering can be divided in two types of approaches, i.e. *instance based* and *metric based* approaches. In the *instance based* approaches, the constraints are used to influence the cluster algorithm directly based on the constraint pairs. This is done in different ways, e.g. by using the constraints for initialization (e.g. in [9]), by enforcing them during the clustering (e.g. in [14]), or by integrating them in the cluster objective function (e.g. in [4]).

The *metric based* approaches map the given constraints to a distance metric or similarity measure, which is then used during the clustering process (e.g. in [3]). The basic idea of most of these approaches is to weight features differently, depending on their importance for the distance computation. While the metric is usually learned in advance using only the given constraints, the approach in [5] adapts the distance metric during clustering. Such an approach allows for the integration of knowledge from unlabeled objects.

Instance based approaches usually have less impact on the clustering as the constraints influence the algorithm rather locally. The strength of their impact strongly depends on the clustering algorithm used and the method of their integration into the algorithm. However, in the worst case, directly enforcing the constraints might lead to no overall benefit or even decrease the clustering performance, especially if the constraints highly violate the underlying similarity space.

On the other hand, learning a distance metric aims on generalizing knowledge about the final clustering from the given constraints. More specific, it tries to identify what features were important for grouping objects or distinguishing objects. If the constraints were derived from labeled data, it might be sufficient to cluster only the unlabeled data with the learned metric and thus still deriving an improved cluster structure. As the structure of the labeled data is already known, ignoring it during the clustering process can save a large amount of runtime. This is in contrast to the instance based approaches that always require that the labeled data is clustered together with the unlabeled data.

Nevertheless, the global influence of the constraints can be problematic under two different aspects. First, different classes might contradict in terms of important features to stress. Features that improve the cluster formation for a certain class might split a different cluster. And second, if the constraints do not cover all interesting clusters, the metric learning might be strongly biased by the known classes. Unfortunately, this is not analyzed in current literature, although it is an important aspect. Considering our scenario, it is more likely that constraints are not evenly distributed over all classes as some are usually unknown in advance. Evaluations discussed in current literature are based on constraints that are about evenly distributed. They are usually drawn at random from the complete instance space, ignoring the probable fact of locally focused constraints. In this paper, we want to fill this gap by evaluating our approaches with scenarios of supervision yielding different local distributions of constraints.

From the used clustering algorithms, K-Means is the most predominant, e.g. in [14]. Hierarchical agglomerative clustering (HAC) can also be found several times, e.g. in [7]. The authors of [7] aimed at showing theoretical results on different constraint types using HAC. Although using HAC, their work only analyzed flat cluster structures. Furthermore, they were specifically interested, whether constraints could be always satisfied and what the result would look like in these cases. In contrast to that, our interest is on an improved clustering for unlabeled data objects. The strict enforcement of constraints is less important as long as clustering improves. For metric learning, optimization problems are often formulated and solved, e.g. in [15]. In [13], we experimented with SOM clustering and different metric learning strategies.

In our work presented here, we compare an instance based approach with a metric based approach as well as their combination. Gradient descent is used for metric learning and a HAC algorithm (with centroid linkage) for clustering. This choice was particularly motivated

by the fact that the number of clusters (on each hierarchy level) is unknown in advance. Furthermore, we are interested in hierarchical cluster structures, which also makes it difficult to compare our results to other approaches, as these algorithms were always evaluated on data with a flat cluster partitioning.

Another important factor is the data to be clustered. In general, all approaches in the literature only require a feature vector as representation of the data. However, if features are weighted for metric learning, special attention should be paid on how these vectors were created in the first place, i.e. what the vector representation means. Here, we focus on text data. Feature vectors of text documents are usually created by assigning each term to a single feature, weighting each feature with the $tf \times idf$ weighting scheme [10]. As the created vectors highly depend on the length of the document, a common procedure is the normalization of all vectors as content similarity should be independent of document length. However, this has the effect that dimensions are no longer independent from each other as the final value for a feature also depends on the values of the other features. This can cause problems for the metric learning approaches, because it makes finding the good features more difficult.

To illustrate this problem consider the following example. For two documents of the same class, there are some "good" features, which have about the same value in the initial vector representation. Furthermore, there are some "noise" features (i.e. features that do not contribute in identifying the class). Their values differ by a large value between the two documents. A good metric would ignore the noise features and give all weight to the good features. If the vectors will be normalized, the values of the good features might now also vary a lot, just due to the values in the noise features. The correct metric is no longer that obvious, and it gets questionable, whether the metric learning approach will still be capable to weight the correct features. Furthermore, a renormalization based on the learned metric is necessary to be able of getting the initially possible improvements. In contrast to related work, we therefore pay special attention to normalization effects in our suggested metric learning approach.

4 Constraint Based HAC

As mentioned earlier, we implemented an instance based approach as well as a metric based approach. Both approaches will be described in the following.

4.1 Instance Based HAC (iHAC). The instanced based approach (which is called iHAC from now on) is rather straight forward. Here, the hierarchical agglomerative clustering algorithm is modified in a way such that merges occur only in accordance with the given MLB constraints. This is achieved by replacing the step of merging the two closest clusters with merging the two closest clusters from the set of candidate merges that do not violate the given constraints. For all constraints (o_x, o_y, o_z), the cluster containing o_z can only be merged with a cluster that either contains both, o_x and o_y, or neither. If the constraints were created from a given hierarchy, they are always non-conflicting. In this case, iHAC always creates a complete dendrogram. In the other case of conflicting constraints, iHAC might stop in a dead end before reaching the root node. This means that any merge possible at this moment would violate a constraint.

In literature, it is sometimes proposed to modify the cluster initialization to group items known to belong to the same cluster [7]. Although this would be possible for the case of a given hierarchy, we decided against this for the following reasons. First, such groups could only be formed for leaf node classes in the given hierarchy. Furthermore, if constraints are only given by triplets (o_x, o_y, o_z), such groups are even not specifically known in advance. Moreover, we use centroid linkage for cluster similarity computation. Forming such a group of items, which might not be similar in the given similarity space, could lead to a creation of bad centroids, thus decreasing clustering performance. Additionally, it would prevent (or at least make more difficult) the detection of class refinements into sub-classes.

4.2 Metric Based HAC (mHAC). Our metric based approach mHAC uses the cosine similarity as basis, because it is often used in the domain of text mining. Thus, the similarity measure is defined as

$$\text{(4.4)}\quad \text{sim}(o_1, o_2) = \frac{o_1^T o_2}{|o_1| \cdot |o_2|} = \frac{\sum_i o_{1,i} o_{2,i}}{\sqrt{\sum_i o_{1,i}^2}\sqrt{\sum_i o_{2,i}^2}}$$

and computes the cosine of the angle between two vectors. Its most important property for this work is that it is by definition independent from vector length, which makes it especially valuable concerning the normalization issues stated in Section 3.

4.2.1 Similarity Measure. To adapt the similarity space, a weight vector w is defined that contains a weight for each dimension (i.e. for each feature/term). Its goal is to express that certain terms/features are more important for the similarity as others (of course according to the constraints). We require that each weight is greater than or equal to zero and that their sum is bound to the number of features to avoid extreme case

solutions:

$$\forall w_i : w_i \geq 0 \qquad \sum_i w_i = n \tag{4.5}$$

Setting all weights to 1 defines a weighting scheme for the standard case of no feature weighting. This weighting scheme is valid according to (4.5).

The weights are directly integrated in the vector representation. We use unnormalized $tf \times idf$ vectors for the representation of text documents, taking into account the effects from normalization as described in Section 3. Therefore, the document vectors are created as follows:

$$d = \begin{pmatrix} \sqrt{w_1} \cdot d_1 \\ \dots \\ \sqrt{w_n} \cdot d_n \end{pmatrix} = \begin{pmatrix} \sqrt{w_1} \cdot tf_{1,d} \log idf_1 \\ \dots \\ \sqrt{w_n} \cdot tf_{n,d} \log idf_n \end{pmatrix} \tag{4.6}$$

Instead of applying each weight linearly, we use its square root. This has computational reasons that do not change the effect of the weight: In all necessary computations, i.e. similarity or vector length computation, vector elements are always multiplied. Therefore, weights only occur in squared form, allowing for a substitution of $w_i' = w_i^2$ as a consequence. For the computation of the weight update rule in the weight learning process, this substitution is especially beneficial, as it leads to an update rule that can also modify weights that are currently 0 (see below). To sum up, similarity between documents is therefore computed by the cosine similarity as in (4.4) between the weighted document vectors as defined in (4.6).

4.2.2 Weight Adaptation. During learning of the weights, all constraint triplets (o_x, o_y, o_z) are presented to the algorithm several times. For each violated constraint, w is updated using a gradient descent approach. Each constraint thereby provides a relation between object similarities as defined in (4.7), i.e. o_x should be more similar to o_y than to o_z. A violated constraint is recognized by a violation of (4.7). However, we replaced 0 by a small ϵ (in our experiments 0.01) to obtain a more stable adaptation. The relation in (4.7) is also used to guide the gradient descent, trying to maximize (4.8) for each constraint. This leads to the weight update rule in (4.9), where η is the learning rate defining the step width of each adaptation step.

$$\text{sim}(o_x, o_y) - \text{sim}(o_x, o_z) > 0. \tag{4.7}$$

$$obj_{xyz} = \text{sim}(o_x, o_y) - \text{sim}(o_x, o_z) \tag{4.8}$$

$$w_i \leftarrow w_i + \eta \Delta w_i = w_i + \eta \frac{\partial obj_{xyz}}{\partial w_i} \tag{4.9}$$

The final computation of Δw_i after differentiation is:

$$\begin{aligned} \Delta w_i &= \overline{o}_{x,i}(\overline{o}_{y,i} - \overline{o}_{z,i}) - \frac{1}{2}\text{sim}(o_x, o_y)(\overline{o}_{x,i}^2 + \overline{o}_{y,i}^2) \\ &\quad + \frac{1}{2}\text{sim}(o_x, o_z)(\overline{o}_{x,i}^2 + \overline{o}_{z,i}^2) \\ &\quad \text{with } \overline{o}_{x,i} = o_{x,i}/|o_x|. \end{aligned} \tag{4.10}$$

Similarity and vector length are computed on the weighted vectors based on the current weights. As can be seen, this formula can also modify weights that are currently zero as it is not of the form $w_i \cdot term$. However, this would have been the case, if vectors were weighted by w_i instead of $\sqrt{w_i}$. After all weights have been updated for one violated constraint by (4.9), all weights are checked and modified, if necessary, to fit our conditions in (4.5). This means that all negative weights are set to 0. After that, the weights are normalized to sum up to n.

4.2.3 Discussion. Using this weight learning approach, several problems need to be addressed. First, we want to discuss the scenario of unevenly scattered constraints. It can emerge in two different ways. Primarily, terms can occur only in unlabeled documents but not in documents that are part of a constraint. These terms might be potentially important for unknown classes. However, the degree of their importance cannot be estimated. Therefore, their weights are not modified and remain to be set equal to 1. To achieve this, all terms that do not occur in a single constraint are omitted in the weight learning method. Once, the other weights were learned, weights for these terms are added and initialized with a value of 1.

Additionally, some of the terms already occurring in the known classes might be of increased importance for unknown classes. Updating these weights only based on the known classes might lead to a worse performance. Although this cannot be completely avoided, we try to minimize this effect by keeping the weight changes as small as possible to fulfill the constraints. In specific, this means changing the weights only in the case of a violated constraint.

A second problem is connected to run-time performance with increasing number of labeled data. The number of extracted constraints increases exponentially with increasing number of labeled data. This slows down the metric learning process significantly. At the same time, new labeled data does not contribute to the same extend in improving the quality of the metric. Often, similar knowledge is contributed by further data points. Therefore, we reduce the number of generated constraints by first clustering similar items. In our ex-

periments, we clustered all data points directly assigned to the same class with k-means using $k = 5$. The resulting centroids are then used to represent the class instead of the single items. This ensures an upper bound for the run-time. The choice of using five clusters had mainly two reasons. First, we wanted to use very few clusters to improve performance. Second, five clusters showed good results in our tests. However, so far, we did not try to find the optimal number of clusters. Determining the number of clusters based on the data might improve the efficiency of the clustering step. This, however, is left for future work.

Finally, we want to discuss the problem of contradicting constraints between classes. As we consider a hierarchy of classes, different granularity levels of the hierarchy contradict each other during weight learning. As an example, consider two classes that have a common parent class in the hierarchy. A few features are crucial to discriminate the two classes. On the specific hierarchy level of these two classes, these features are boosted to allow for distinguishing both classes. However, on the more general level of the parent class, these features get reduced in impact, because both classes are recognized as one that shall be distinguished from others on this level. By our process of constraint generation, there is an imbalance that results in the creation of many more constraints for higher hierarchy levels. This leads to less performance in distinguishing the most specific classes in the hierarchy. To avoid this behavior, we learn different weighting schemes for each hierarchy level. By this, the single weighting schemes can be learned without hierarchy conflicts. All weighting schemes are then combined to form a single weighting scheme. Here, the higher weight is given, the more specific the hierarchy level of the weighting scheme is. This is due to the fact that HAC starts at the bottom, forming the most specific classes first. To be more specific, each weighting scheme is weighted with

$$2 \cdot level\_depth - 1. \tag{4.11}$$

The complete weight learning process is summarized in Fig. 3. Unfortunately, it cannot be assumed that weight learning will succeed in producing a weighting scheme that violates no constraints. Therefore, another stopping criterion is needed. As gradient descent is performed, we stop, if the weight change is below a certain threshold. After weights have been learned, a hierarchical agglomerative clustering is performed on the weighted document vectors to get the final clustering of the documents.

4.3 Combining the Two (miHAC). As both approaches presented influence the clustering in different parts of the algorithm, a combination of both approaches might bring further improvements. We therefore also evaluated miHAC. This approach first learns weights as in mHAC. After that, the documents are clustered with iHAC, whereas the documents are represented by the weighted vectors.

learnWeights(class hierarchy H_c, documents D)
  Initialize w: $\forall i : w_i := 0$
  for all classes $c \in H_c$ **do**
    Compute class representatives
      $R_c = \text{kmeans}(5, D_c)$
  end for
  for all hierarchy levels l **do**
    Extract constraints MLB_l
    Initialize $w^{(l)}$: $\forall w_i^{(l)} : w_i^{(l)} := 1$
    repeat
      for all $(o_x, o_y, o_z) \in MLB_l$ **do**
        if $\text{sim}(o_x, o_y) - \text{sim}(o_x, o_z) \leq \epsilon$ **then**
          $\forall i : w_i^{(l)} := w_i^{(l)} + \eta \Delta w_i^{(l)}$
          $\forall i :$ if $w_i^{(l)} < 0$ then $w_i^{(l)} := 0$
          $sum_w = \sum_{j=1}^{n} w_j^{(l)}$
          $\forall i : w_i^{(l)} := w_i^{(l)} \frac{n}{sum_w}$
        end if
      end for
    until weight change is too small
    $\forall i : w_i := w_i + (2\text{depth}(l) - 1) \cdot w_i^{(l)}$
  end for
  $sum_w = \sum_{j=1}^{n} w_j$
  $\forall i : w_i := w_i \frac{n}{sum_w}$
  return w

Figure 3: Weight Learning in mHAC

5 Evaluation

We evaluated the approaches described in the previous section for its suitability to our learning task. As this task differs from research proposed in the literature, comparisons to other work were not possible. However, we used the standard HAC approach as a baseline. In the following, we first describe the used datasets and evaluation measures. Then we show and discuss the obtained results.

5.1 Datasets. As the goal is to evaluate hierarchical clustering, hierarchical datasets are needed. Unfortunately, such datasets are very rare, as usually only a flat class structure is used. Furthermore, we are interested in text documents here. Two datasets that are public and fulfill the given requirements are the banksearch

dataset[1] and the Reuters corpus volume 1[2]. From these datasets, we generated three smaller sub sets, which are shown in Fig. 4–6. The figures show the class structure as well as the number of documents directly assigned to each class. The first dataset uses the complete structure of the banksearch dataset but only the first 100 documents per class. For the Reuters 1 dataset, we selected some classes and subclasses that seemed to be rather distinguishable. In contrast to this, the Reuters 2 dataset contains classes that are more alike. We randomly sampled a maximum of 100 documents per class, while a lower number in the final dataset means that there were not enough documents in the dataset.

- **Finance** (0)
  - Commercial Banks (100)
  - Building Societies (100)
  - Insurance Agencies (100)
- **Science** (0)
  - Astronomy (100)
  - Biology (100)
- **Programming** (0)
  - C/C++ (100)
  - Java (100)
  - Visual Basic (100)
- **Sport** (100)
  - Soccer (100)
  - Motor Racing (100)

Figure 4: Banksearch dataset

- **Corporate/Industrial** (100)
  - Strategy/Plans (100)
  - Research/Development (100)
  - Advertising/Promotion (100)
- **Economics** (59)
  - Economic Performance (100)
  - Government Borrowing (100)
- **Government/ Social** (100)
  - Disasters and Accidents (100)
  - Health (100)
  - Weather (100)

Figure 5: Reuters 1 dataset

- **Equity Markets** (100)
- **Bond Markets** (100)
- **Money Markets** (100)
  - Interbank Markets (100)
  - Forex Markets (100)
- **Commodity Markets** (100)
  - Soft Commodities (100)
  - Metals Trading (100)
  - Energy Markets (100)

Figure 6: Reuters 2 dataset

All documents were represented with $tf \times idf$ document vectors. We performed a feature selection, removing all terms that occurred less than 5 times, were less than 3 characters long, or contained numbers. From the rest, we selected 5000 terms in an unsupervised manner as described in [6]. To determine this number we conducted a preliminary evaluation. It showed that this number still has a small impact on initial clustering performance, while a larger reduction of the feature space leads to decreasing performance.

Based on these three datasets, we created different settings reflecting different distributions of constraints. Setting (1) is a classification scenario, i.e. a setting, where all classes are known in advance. Two more settings were evaluated with parts of the hierarchy being unknown. In setting (2), a single leaf node class did not provide any labeled data. In setting (3), a whole sub-tree of the hierarchy was left unlabeled. Furthermore, we used different numbers of labeled data given per class. We specifically investigated small numbers of labeled data (with a maximum of 30) as we assume from an application oriented point of view that it is much more likely that labeled data is rare. The labeled data is chosen randomly. However, the same labeled data is used for all algorithms to allow a fair comparison. The constraints guiding the clustering process are created based on the labeled data. All constraints possible are extracted according to the class hierarchy.

5.2 Evaluation Measures. We used two measures to evaluate and compare the performance of our algorithms. First, we used the f-score gained in accordance to the given dataset, which is supposed to be the true class structure that shall be recovered. For its computation in an unlabeled cluster tree (or in a dendrogram), we followed a common approach that selects for each class in the dataset the cluster gaining the highest f-score on it. This is done for all classes in the hierarchy. For a higher level class, all documents contained in sub-classes are also counted as belonging to this class. Please note that this simple procedure might select clusters inconsistent with the hierarchy or multiple times in the case of noisy clusters. Determining the optimal and hierarchy consistent selection has a much higher time complexity. However, the results of the simple procedure are usually sufficient for evaluation purposes and little is gained from enforcing hierarchy consistency.

We only computed the f-score on the unlabeled data, as we want to measure the gain on the new data. As f-score is a class specific value, we computed two mean values: one over all leaf node classes (ll) and one over all non-leaf node classes (hl).

Applying the f-score measure as described potentially leads to an evaluation of only a part of the clustering, because it just considers the best cluster per class. Therefore, we introduce a second measure that aims at evaluating the total clustering. We call it the cluster

[1] Available for download at the StatLib website (http://lib.stat.cmu.edu); Described in [12]

[2] Available from the Reuters website (http://about.reuters.com/researchandstandards/corpus/)

error CE. It interprets the complete dataset in terms of MLB constraints. It then measures how many constraints were violated by the clustering in the dendrogram or cluster tree under investigation. For each document d with a given constraint $(d, (S_1, ..., S_m)) = (d, \mathcal{S})$, we can derive from the dendrogram of the current clustering the order in which documents are merged to d, producing another ordered list of document sets $\mathcal{R} = (R_1, ..., R_r)$. To compute CE, we count how often the order in $\mathcal{R}$ violates the order in $\mathcal{S}$, i.e. the order of two documents d_i and d_j is reversed. In addition, we assume violations between classes that are further apart in the hierarchy as more severe. Therefore, constraint violations are weighted. The distance between two classes is reflected in the distance between two constraint sets in the ordered list, which is therefore used for weighting. The overall cluster error CE is then computed by:

$$CE = \sum_{(d,\mathcal{S},\mathcal{R})} \sum_{\substack{(d_{i_{k,x}}, d_{j_{l,y}}) \\ k<l}} \left\{ \begin{array}{ll} x-y & \text{if } x > y \\ 0 & \text{else} \end{array} \right\}, \tag{5.12}$$

where $d_{i_{k,x}} \in R_k$, $d_{i_{k,x}} \in S_x$, $d_{j_{l,y}} \in R_l$ and $d_{j_{l,y}} \in S_y$. In our results, we will not show the absolute number computed by (5.12) but the relative proportion to the maximum number possible.

5.3 Results. In this section, we present the results obtained in our experiments. For the metric learning approach, we set an upper bound for the number of iterations of 50 to limit the necessary run-time of our experiments. Therefore, it might be possible that the best solution was not found (yet). In the first 30 runs, a learning rate of 10 was used, in the last 20 runs, a learning rate of 1. Using a high learning rate in the beginning speeds up the adaptation process, but soon oscillation becomes a problem, which requires a reduction of the learning rate. This simple procedure worked well in our experiments. However, it might be more difficult in general. Instead of using some fixed step size, it might be useful to apply more sophisticated strategies like simulated annealing. Furthermore, ϵ was set to 0.01.

Figures 7 and 8 show our results on the three datasets. Each column of the two figures corresponds to one evaluation measure. Each dataset is represented with 3 diagram rows, the first showing the results of the classification scenario (setting (1)), the second showing results with one leaf node class unknown (setting (2)), and the third showing results with a complete sub-tree unknown (setting (3)).

In general, it can be seen that all three approaches are capable of improving cluster quality. However, the specific results differ a lot over the different settings analyzed. In our discussion, we start with the classification scenario, using this to compare between the different datasets. For the banksearch dataset (row 1 in Fig. 7), cluster error decreases similarly for all approaches, with a favor for the combined approach miHAC. It gains an improvement of about 18% in the best case. Similar behavior can be found for the f-score measure. Comparing mHAC and iHAC, mHAC is better on the higher level, while iHAC contributes more on the leaf level. Nevertheless, miHAC succeeds in combining both, often providing the best results. In the best case, f-score gained about 6% on the higher level and about 13% on the leaf level.

For the reuters 1 dataset (row 4 in Fig. 7), mHAC does not succeed in improving the cluster error, whereas iHAC gains improvements up to 23%. Concerning f-score, iHAC clearly outperforms mHAC on the higher level, while mHAC is better on the leaf level. Again, miHAC usually provides a better result by the combination, gaining about 20% on the higher level and about 5% on the leaf level. The large gap on the higher level suggests that the used features based on term occurences might not be expressive enough to recover the structure in this dataset.

For the reuters 2 dataset (row 1 in Fig. 8), all algorithms are again capable of largely reducing the cluster error up to 32%. Comparing mHAC and iHAC, no clear winner can be stated, as in some cases mHAC is better and in other iHAC. However, miHAC again suceeds in producing an overall good result. It produces an f-score gain of 23% on the higher level and of 15% on the leaf level. This dataset benefits most from integrating constraints.

Summing up, the combined approach miHAC provides good results on all datasets in the classification scenario. There is no clear winner between the instance based and the metric based approach. Specific performance gains depend on the properties of the dataset. Most importantly, the chosen features need to be expressive enough to explain the class structure in the dataset.

Next, we look at the behavior in the case of unevenly scattered constraints. In general, the performance gain is less, if more classes are unknown in advance. However, this is also expected, as this means a lower number of constraints. Nevertheless, looking at the class specific values (which are not printed here), it shows that classes with no labeled data usually still increase in performance. This can be explained by the fact that knowledge about some classes not only helps in distinguishing between these classes but also reduces the confusion between known and unknown classes.

For the banksearch data (rows 1–3 in Fig. 7), mHAC

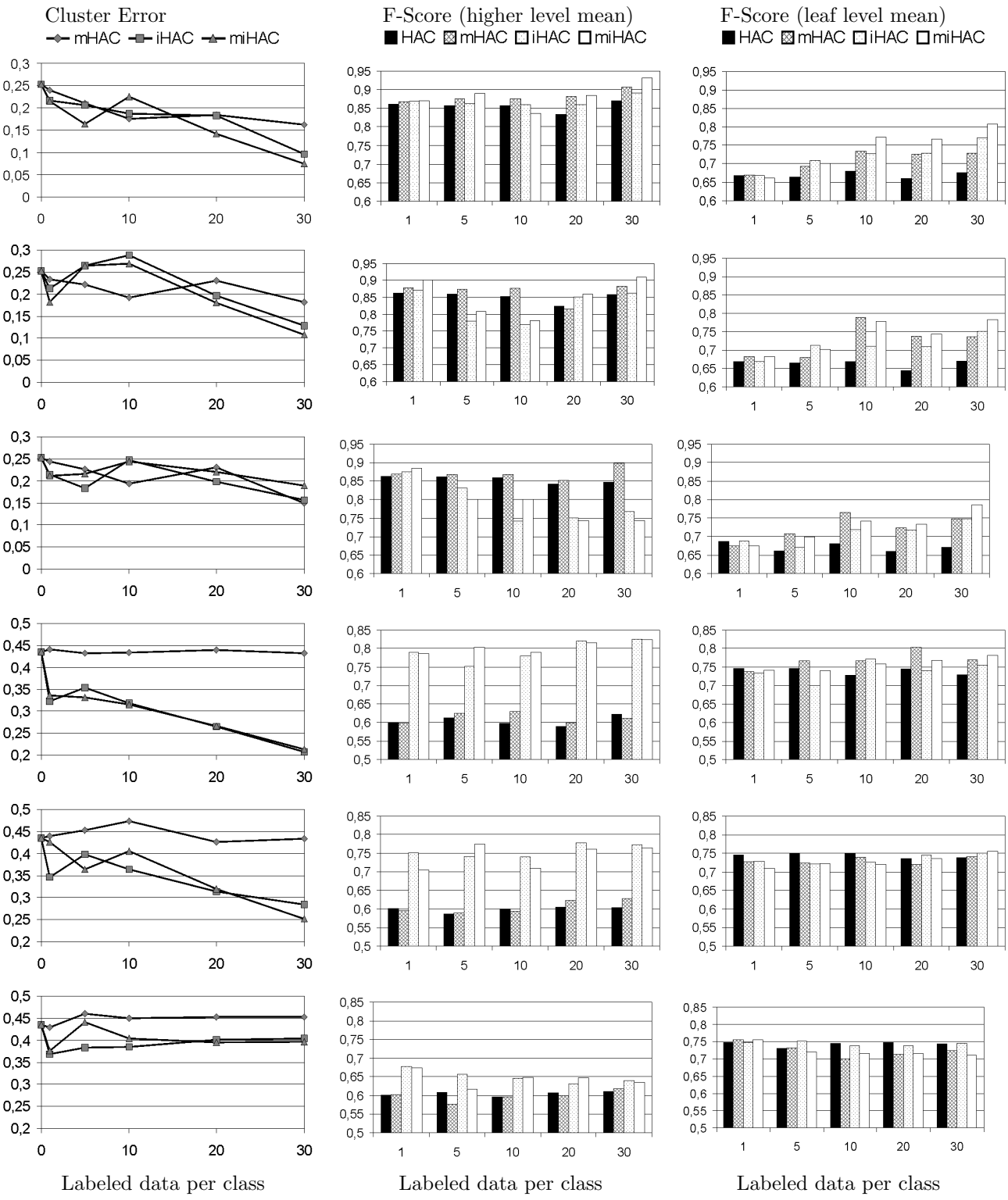

Figure 7: Results: rows 1–3: banksearch data, rows 4–6: reuters 1 data; rows 1, 4: classification scenario; rows 2, 5: one unknown leaf class; rows 3, 6: one unknown sub-tree

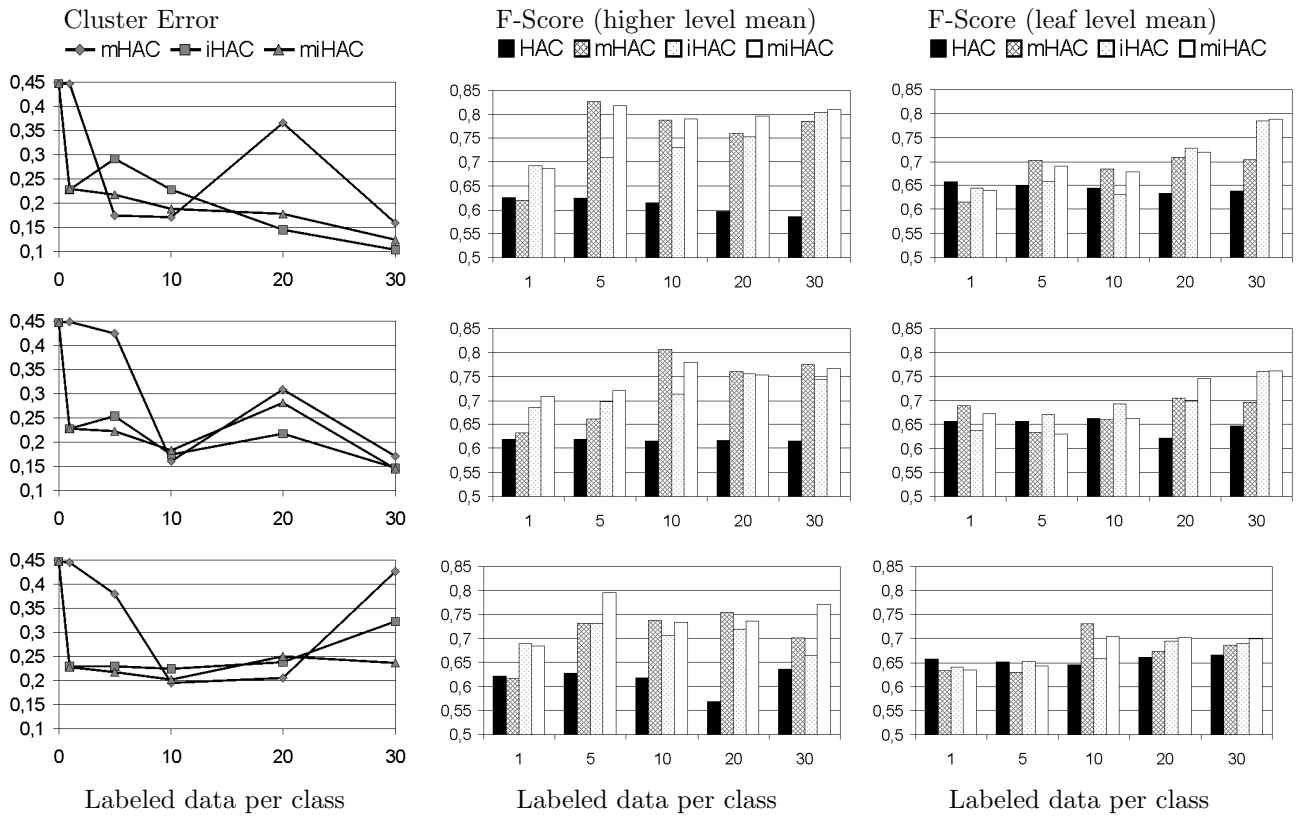

Figure 8: Results: reuters 2 data; row 1: classification scenario; row 2: one unknown leaf class; row 3: one unknown sub-tree

shows to have the most stable performance with increasing number of unknown classes. On the higher level, it increasingly outperforms iHAC as well as miHAC, which seems to be bound to the bad performance of iHAC. iHAC and miHAC deteriorate so much that results get worse than the baseline performance of standard HAC.

For the reuters 1 data (rows 4–6 in Fig. 7), iHAC looses its good performance on the higher level. This is probably due to the fact that iHAC does not generalize. Furthermore, the results of mHAC showed that generalization is not possible for this dataset. Therefore, the loss of iHAC (and with it also miHAC) is directly bound to the less number of constraints. The performance on the leaf level remains poor for all algorithms, with all algorithms having usually slightly less performance than the baseline of HAC.

For the reuters 2 data (rows 1–3 in Fig. 8), all algorithms slightly loose performance in comparison to the classification scenario. However, the general picture stays the same, with the combined approach miHAC the best way to go.

Summing up over all datasets, it gets more difficult to propose a general best algorithm. Although in most cases, the combined approach works well, it dramatically lost performance for the banksearch data. For this data, it seems to be bound too tight to the instance based component. Future work should analyze, whether this bound can be broken.

Presenting mHAC, we introduced a reduction of the constraints by applying k-means clustering on the labeled data of each class in advance. We briefly show here that this is reasonable to be done. Fig. 9 shows results on the reuters 2 dataset, comparing the constraint generation with and without initial clustering. Especially in the f-score measure, it shows that introducing the clustering step even increases performance, while at the same time, run-time reduces by an increasingly large amount. This is not that clear in the cluster error, i.e. on the whole dendrogram. Nevertheless, we think the results encourage the use of the clustering step.

Furthermore, we learn weighting schemes on different hierarchy levels. With table 1, we show that weights for different hierarchy levels are indeed conflicting. The results in table 1 were produced on the reuters 1 dataset with 30 labeled documents per class. Two weighting schemes are compared against the standard HAC re-

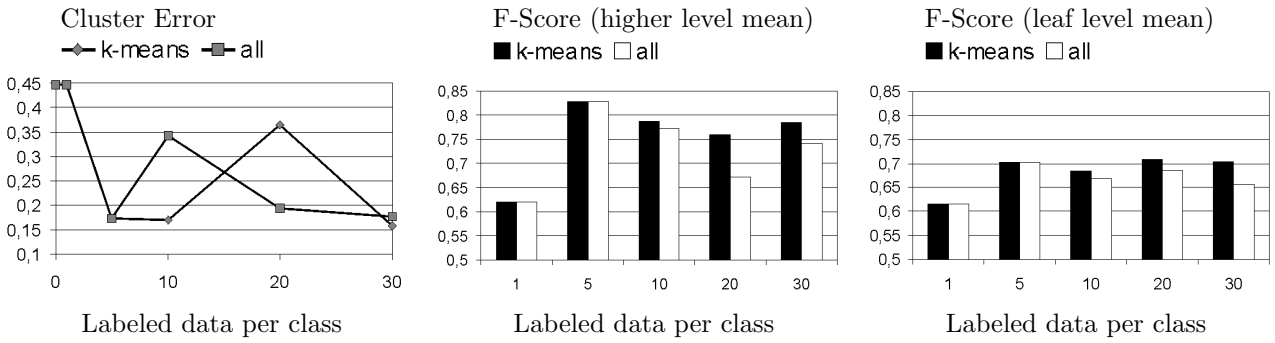

Figure 9: Comparison of learning a metric with and without initial clustering of the labeled data

sult: one optimized for the leaf level of the hierarchy and one optimized for the higher level. Both weighting schemes succeed in improving the cluster quality on the level they are optimized for, however on the costs of the other level. The combination of both weighting schemes into a single one will have a behavior that is bound to both extremes. Depending on how much both weighting schemes conflict each other, it is possible to get a weighting scheme with improved performance on both levels. However, it will usually not be possible to create a weighting with the maximum on both levels. Future work will deal with further investigation, how such conflicts could be handled.

Table 1: Comparison of weighting schemes optimized for different hierarchy levels

	ll optimized	hl optimized	standard
f-score (ll)	0.77	0.59	0.73
f-score (hl)	0.62	0.71	0.62

6 Conclusion

In this paper, we dealt with semi-supervised hierarchical clustering. The specific requirements of such a hierarchical setting were specified. To our knowledge, there is no related work dealing with the same setting, although it has several real-world applications. It was shown that the typical constraint formulation of must link and cannot link constraints is not appropriate. A solution based on MLB constraints was therefore suggested. Based on related work in constraint based flat clustering, two types of approaches, instance based and metric based approaches, were identified. For each type, an approach was presented solving the hierarchical setting.

All approaches were evaluated on three datasets. It could be shown that standard clustering performance can be significantly improved by our methods. Good results are already achieved for a small number of labeled data, which encourages the use in real-world applications, where labeled data is usually rare. Best results were naturally gained in a setting, where all classes are known in advance. Nevertheless, performance can also be improved in scenarios with a partially known hierarchy, which is expected to be more common in many real-world problems.

For our metric based approach as well as for metric based approaches in general, conflicts between classes in determining the best metric are a major issue. These conflicts occur between classes of the same level (which can also be found in flat clustering) as well as between classes on different hierarchy levels. Furthermore, with increasing number of classes, the problem is more likely to occur. Hierarchies usually tend to include more classes as considered in flat scenarios, as a high number of classes is still manageable by the user within hierarchies. Therefore, in hierarchical settings, we assume the problem to be more dominant. In this work, we already presented one method to handle conflicts between hierarchy levels. However, future work is necessary to appropriately deal with this problem. Here, we envision the use of more locally focused metrics rather than a single global one.

The result of our presented methods is always a dendrogram, i.e. a complete hierarchical representation of the data. However, from an application point of view, where a user is involved in interacting with the final clustering result, a dendrogram representation of the data will be of little use. Therefore, the extraction of a more coarse grained structure is required to allow efficient access to the data. Furthermore, cluster labels are necessary, so that the user is capable of identifying interesting clusters quickly. Like the clustering process itself, these tasks can also be viewed under semi-supervised aspects. The interested reader shall therefor be referred to [1].

References

[1] K. Bade, M. Hermkes, and A. Nürnberger. User oriented hierarchical information organization and retrieval. In *Proceedings of the 2007 European Conference on Machine Learning (ECML)*, 2007.

[2] K. Bade and A. Nürnberger. Personalized hierarchical clustering. In *Proceedings of the 2006 IEEE/WIC/ACM Int. Conference on Web Intelligence*, 2006.

[3] A. Bar-Hillel, T. Hertz, N. Shental, and D. Weinshall. Learning distance functions using equivalence relations. In *Proc. of the 20th International Conference on Machine Learning (ICML 2003)*, pages 11–18, 2003.

[4] S. Basu, A. Banerjee, and R. Mooney. Active semi-supervision for pairwise constrained clustering. In *Proc. of the 4th SIAM Int. Conf. on Data Mining*, 2004.

[5] M. Bilenko, S. Basu, and R. J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proc. of the 21st International Conference on Machine Learning (ICML04)*, pages 81–88, 2004.

[6] C. Borgelt and A. Nürnberger. Fast fuzzy clustering of web page collections. In *Proc. of PKDD Workshop on Statistical Approaches for Web Mining (SAWM)*, 2004.

[7] I. Davidson and S. S. Ravi. Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. In *Proc. of the 9th European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD05)*, pages 59–70, 2005.

[8] H. A. Kestler, J. M. Kraus, G. Palm, and F. Schwenker. On the effects of constraints in semi-supervised hierarchical clustering. In F. Schwenker and S. Marinai, editors, *Artificial Neural Networks in Pattern Recognition*, volume 4087 of *LNAI*, pages 57–66, 2006.

[9] H. Kim and S. Lee. An effective document clustering method using user-adaptable distance metrics. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 16–20, New York, NY, USA, 2002. ACM Press.

[10] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[11] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *Proceedings of Neural Information Processing Systems*, 2004.

[12] M. Sinka and D. Corne. A large benchmark dataset for web document clustering. In *Soft Computing Systems: Design, Management and Applications, Vol. 87 of Frontiers in Artificial Intelligence and Applications*, pages 881–890, 2002.

[13] S. Stober and A. Nürnberger. User modelling for interactive user-adaptive collection structuring. In *Postproceedings of 5th International Workshop on Adaptive Multimedia Retrieval (AMR07)*, 2008.

[14] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained k-means clustering with background knowledge. In *Proceedings of 18th International Conference on Machine Learning*, pages 577–584, 2001.

[15] E. Xing, A. Ng, M. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems 15*, pages 505–512. 2003.