Mach Learn (2014) 94:371–399
DOI 10.1007/s10994-013-5397-9

Hierarchical constraints

Providing structural bias for hierarchical clustering

Korinna Bade · Andreas Nürnberger

Received: 12 February 2010 / Accepted: 17 June 2013 / Published online: 7 September 2013

Abstract Constrained clustering received a lot of attention in the last years. However, the widely used pairwise constraints are not generally applicable for hierarchical clustering, where the goal is to derive a cluster hierarchy instead of a flat partition. Therefore, we propose for the hierarchical setting—based on the ideas of pairwise constraints—the use of must-link-before (MLB) constraints. In this paper, we discuss their properties and present an algorithm that is able to create a hierarchy by considering these constraints directly. Furthermore, we propose an efficient data structure for its implementation and evaluate its effectiveness with different datasets in a text clustering scenario.

Keywords Constrained clustering · Hierarchical clustering · Semi-supervised learning

1 Introduction

Automatic structuring of data is a very helpful tool for improving a user's understanding of the data and providing an overview over the information contained therein. If the amount of data increases, a structuring into a hierarchy is especially useful since it can group the data on different levels of granularity. This gives a more detailed insight into the data and enables a user to concentrate on interesting parts of the data, analyzing details only when necessary. Hierarchical clustering methods were developed in the past to fulfill this task. The goal of these approaches is to group data only based on their features, i.e. information assigned to

Editor: Jennifer Dy.

K. Bade (✉)
Department of Computer Science and Languages, Anhalt University of Applied Sciences,
Lohmannstraße 23, 06366 Köthen (Anhalt), Germany
e-mail: k.bade@inf.hs-anhalt.de

A. Nürnberger
Fakultät für Informatik, Institut für Technische und Betriebliche Informationssysteme,
Otto-von-Guericke-University Magdeburg, 39106 Magdeburg, Germany
e-mail: andreas.nuernberger@ovgu.de

or extracted from the objects themselves. However, there is usually more than one way to create useful clusters. Without additional criteria, it cannot be decided, which of these is the best one. It depends on various aspects including the user's preferences, knowledge, task, or context. Therefore, more recent research analyzes how any kind of prior knowledge about an optimal final structure can be integrated into the clustering process. Pairwise constraints, as originally introduced by Wagstaff et al. (2001), gained much attention for this purpose. While their idea is simple and effective, pairwise constraints are limited to partitional clustering (Wagstaff and Cardie 2000). Therefore, this paper discusses a hierarchical constraint type that extends the ideas of pairwise constraints to hierarchical clustering.

The *main contribution* of this paper is to discuss how hierarchical clustering can be constrained appropriately to integrate prior knowledge about the target cluster structure. We suggest the use of must-link-before (MLB) constraints and formally discuss their properties here. We show the effectiveness of these constraints through an approach that extends hierarchical agglomerative clustering (Hastie et al. 2009, Chap. 14.3.12) to obey MLB constraints and its application in a document clustering task. In some former publications (Bade and Nürnberger 2006, 2008, 2009), we already presented some preliminary results of this research and different ideas of constraint integration. However, these articles only introduce briefly the general concepts of MLB constraints. In Bade and Nürnberger (2006) we proposed a first—from the current perspective unnecessary complex—formalization of MLB constraints. In the current contribution we provide a much clearer formal introduction of the concept and its properties. Furthermore, in prior work we focused on learning a modified metric rather than a direct, instance-based integration of constraints in the clustering process itself. In Bade and Nürnberger (2008, 2009) we presented basic ideas of the iHAC algorithm. However, no detailed description was given. This contribution provides additional details including an algorithm for an efficient implementation of the ideas. Furthermore, this paper includes a discussion on evaluation measures for hierarchies.

From the algorithmic point of view, we solve a hierarchical semi-supervised clustering task. The overall goal is to derive a hierarchical cluster structure guided through some kind of supervision. Likewise to clustering with pairwise constraints, we are dealing with similarly simple and general partial knowledge about the target cluster structure. Hence, our supervision is provided through a set of constraints which are specifically designed to work with the hierarchical clustering task. These constraints are formalized on the data object level, describing the (hierarchical) relationship between specific objects.

1.1 Application scenarios

As with pairwise constraints, MLB constraints naturally occur in quite diverse application scenarios. Some of those are briefly discussed in the following. Our research was mainly motivated in providing a solution for the task of hierarchical clustering documents in personal information management (Jones and Teevan 2007; Jones 2008). Here, a user is creating and maintaining a hierarchy of documents that s/he needs to re-access later on. An automatic method shall assist the user in this task. To take into account the user's preferences on the target hierarchy, constraints can be used to guide the automatic clustering process. Two different modes of interaction are possible, which should be supported by the clustering algorithm.

First option is an interactive feedback scenario, in which users rate selected items concerning their (hierarchical) relationship. Specifically, they would critique an initial clustering through the system similar to relevance feedback in information retrieval systems (Manning et al. 2008). This feedback is translated directly into MLB-constraints and applied to provide an improved clustering of the data.

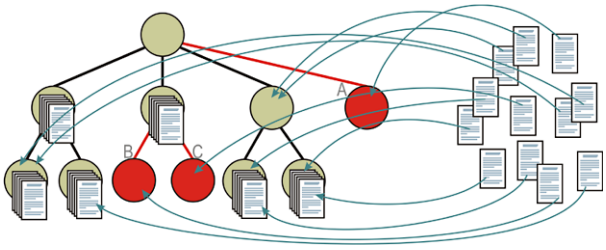

Fig. 1 Semi-supervised hierarchical clustering of documents (*left*: given hierarchy with training examples and refinement through clustering (*node A*: new class; *nodes B, C*: new subclasses); *right*: collection of new documents)

In the second mode, the system analyzes an existing hierarchical structure on old data and applies this to new available data. The existing hierarchy is translated into a (large) set of MLB-constraints, which is applied for clustering the new data. Here, the goal is to extend the user's existing information space (represented through a hierarchy filled with data) with new data. The existing hierarchy might not be sufficient to cover the new data as new topics naturally arise over time. Hence, we again need to learn a new hierarchical cluster structure, covering both old and new data. As we already know the user's structuring preferences on the old data, this is incorporated as constraints to the clustering. From a user interaction perspective, it is also a necessity that the user can easily navigate through the data. Therefore, it is crucial that the learned/adapted hierarchy is very similar to the existing one to improve the user's ease of use. The strongest limitation would be to only allow for hierarchy extension, which is visualized in Fig. 1. The given hierarchy on the left (which corresponds to the user collection) is extended with new nodes (A, B, C) based on the collection on the right describing categories not described by the user collection yet. The new nodes can correspond to new classes, which are sibling nodes of existing nodes in the hierarchy (e.g., node A), or to new subclasses, which refine existing nodes in depth (e.g., nodes B and C). The unlabeled documents on the right are inserted into this refined hierarchy, i.e., either in new or existing nodes. Of course, the tasks of hierarchy refinement and insertion of items are accomplished in parallel.

Please note that this scenario can be solved in different ways. One alternative is the use of incremental algorithms starting with the existing hierarchy and incrementally adding new data. Instead of reclustering the data, the existing hierarchy is modified through a certain set of operators. This is especially beneficial, if data points emerge one by one in a sequence over time. Examples for such approaches can be found in McKusick and Langley (1991), Fisher (1987), Choi and Peng (2004). Fisher (1987) introduced the COBWEB algorithm to build a cluster hierarchy. It uses a heuristic measure of category utility to decide where to add the new object and uses four different operators to modify the hierarchy: classifying the object with respect to an existing class, creating a new class, combining two classes into a single class, and dividing a class into several classes. McKusick and Langley (1991) proposed ARACHNE, an algorithm based on COBWEB. It has the same underlying idea but uses different operations to modify the tree structure. Specifically, they test nodes for vertical and horizontal fitting, which either moves nodes upwards in the hierarchy or merges them with a sibling. Choi and Peng (2004) split their proposed algorithm in two different phases. In the first phase, an initial hierarchy is learned using a batch learning algorithm. In the second phase, new items are incrementally added to the hierarchy. This is especially interesting, if we have already given a hierarchy as a starting point as in our example.

All of those algorithms could be extended to incorporate (hierarchical) constraints. They do not handle them so far as the operations for altering the current hierarchy might lead to constraint violations. In this paper, we decided on a batch learning algorithm to show the effectiveness of the proposed constraints. However, in the future, different algorithms (batch

as well as incremental) should be analyzed on their behavior with these constraints. This is out of the scope of this paper. Here, we focus on introducing the constraints itself and discussing their capabilities.

Another good source for constraints are spatial information. As also mentioned and further investigated by Wagstaff (2002), spatial information is better expressed through constraints than through straight forward methods like adding additional features. In this way, spatial constraints can be enforced by the algorithm or in other ways handled more explicitly than through a feature representation. Wagstaff (2002) worked with pairwise constraints. It is our hypothesis that hierarchical constraints as proposed in this paper are an interesting alternative. However, we have not done any experiments yet as the focus of this paper is on document data. This is left for future work. Nevertheless, we want to roughly discuss, how constraints could be applied for spatial data on two different scenarios.

One such application is image segmentation (for a more detailed introduction on image segmentation see, e.g., Gonzalez and Woods 2007). Here, the goal is to cluster pixels of an image to form coherent regions which are assumed to reflect certain objects. While image segmentation can mean to create a partition of pixels, a hierarchical segmentation can give a more detailed description of an image. For example, one could be interested in extracting the car as a whole as well as extracting its door or tire. A clustering approach can be used to create a partition or cluster hierarchy of pixels based on features like pixel color. However, this could lead to clusters with no spatial coherence, which is not useful for image segmentation. MLB constraints can be used to encode these spatial constraints. Those constraints can be computed automatically from the image without the need for manual interference. Through the MLB constraints, the clustering algorithm can be modified to find spatially coherent clusters. Furthermore, additional available information can be integrated in the same manner, not requiring a different approach.

The same is possible for other clustering tasks, where spatial position does matter, e.g., like in approaches for the automatic learning of management zones in precision agriculture (Khosla et al. 2010; Ruß and Kruse 2011). Here, we have geo-referenced data of individual regions on large farms. Depending on the properties, different regions are handled differently by the farmer in terms of watering, adding fertilizer and so on. Forming regions that should receive the same handling is the goal of management zone learning. As in image segmentation, these regions must have spatial coherence. Hence, we have an identical clustering problem of individual elements with certain properties and additional spatial constraints.

1.2 Structure of the paper

The paper is structured as follows. We start with a review on pairwise constraints in the next section. Then, we introduce hierarchical constraints in Sect. 3 and discuss their properties and applicability. In Sect. 4, we focus on constrained hierarchical clustering and discuss an algorithm that integrates MLB constraints efficiently into hierarchical agglomerative clustering (HAC). Finally, we evaluate the algorithm under different scenarios of supervision in terms of clustering quality and runtime in Sect. 5.

2 Pairwise constraints

The goal of pairwise constraints is to express the relative cluster assignment of two items (Wagstaff et al. 2001). As they are defined on pairs of items, they do not require the knowledge of explicit categories. It is sufficient to know whether two items belong to the same or

Fig. 2 An example of must-link (*left*) and cannot-link constraints (*right*)

to different categories without the need to name or describe these categories. Nonetheless, if training data is given in the form of instances and their respective target class, a representation with pairwise constraints can always be extracted. Hence, the usage of pairwise constraints allows for a more general applicability.

There are two types of pairwise constraints, *must-link* constraints and *cannot-link* constraints. A must-link constraint expresses that two items belong to the same cluster, and a cannot-link constraint describes an item pair that should be separated through clustering. An example is given in Fig. 2.

Several approaches were developed in the past that use pairwise constraints during clustering. They can be divided in two broad types of approaches, i.e., *instance-based* and *metric-based* approaches. *Instance-based* approaches use constraints 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., as in Kim and Lee 2002), by enforcing them during the clustering (e.g., as in Wagstaff et al. 2001), or by integrating them in the cluster objective function (e.g., as in Basu et al. 2004a). Furthermore, instance-based approaches can be divided into approaches that require hard constraint satisfaction (e.g., as in Wagstaff et al. 2001) and those that allow violation of some constraints (e.g., as in Basu et al. 2004a).

The *metric-based* approaches try to generalize the given knowledge by mapping the constraints to a distance metric or similarity measure, which is then used during the clustering process (e.g., as in Bar-Hillel et al. 2003). 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 of Bilenko et al. (2004) adapts the distance metric during clustering. Such an approach allows for the integration of knowledge from unconstrained items.

2.1 A review of selected approaches

In the following, specific approaches from the literature are presented in more detail. These papers form a representative selection of the existing literature showing the diversity of existing approaches.

Instance-based approaches The first one to propose pairwise constraints was Kiri Wagstaff. Wagstaff and Cardie (2000) first integrated them in the COBWEB clustering method, enforcing hard constraint satisfaction. Following up their work, Wagstaff et al. (2001) proposed COP-KMEANS. This approach alters the assignment of items to the cluster center in k-means. Items are only assigned to clusters such that no constraint is violated. If such an assignment is not possible at any time, the algorithm terminates without a solution. Basu et al. (2002) used labeled data as seeds for initializing cluster centers. Additionally, the labeled data remains fixed to the same clusters during re-computation of cluster assignment. Two years later, they defined an objective function for k-means that is a trade-off between the standard k-means objective and constraint violations (Basu et al. 2004a). They also initialize the clusters using the transitive closure of must-link constraints. Bilenko et al. (2004) combines this approach with metric learning. The Euclidean distance is weighted through a matrix, whereby each cluster adapts its own matrix. It is noteworthy that the metric is adapted during the clustering instead before clustering as done in most other approaches.

This work is also embedded in a framework based on hidden Markov random fields (Basu et al. 2004b).

Instance-based approaches were furthermore combined with other clustering methods. Grira et al. (2004) integrated pairwise constraints into fuzzy clustering. They propose a new objective function that combines the standard objective with constraint violations, similar to the work of Basu et al. (2004a). Based on this, they derived a new update formula for the individual refinement steps of the algorithm. Ruiz et al. (2007b) extended density-based clustering. Their algorithm ensures that all constraints are satisfied by design. This, however, leads to the creation of many small clusters. After an evaluation with benchmark data, they also applied their method on query log data (Ruiz et al. 2007a).

Metric learning Bar-Hillel et al. (2003, 2005) learn a Mahalanobis metric from must-link constraints only. They propose a Relevant Component Analysis which provides a closed form solution for different optimization problems. However, this is only possible because cannot-links are neglected. Nevertheless, they show comparable results to Xing et al. (2003) while requiring less computation time. Xing et al. (2003) also learned a weighted metric similar to the Mahalanobis distance using both types of constraints. A restricted problem using only a diagonal matrix is solved as an optimization problem while a gradient descent search is used for learning a full matrix. Both Bar-Hillel et al. (2005) and Xing et al. (2003) showed the superiority of metric learning over instance based approaches. Kim and Lee (2002) also learned a weighted metric similar to the Mahalanobis distance. They solved an optimization problem that aims at minimizing distance between must-links and maximizing distance between cannot-links with a gradient descent search.

A different approach was taken by Klein et al. (2002). They directly modified the similarity matrix, which can be used directly in hierarchical agglomerative clustering. First, entries of must-link pairs are set to 0. However, this can result in a violation of the triangular inequality for some points. This problem is fixed with a shortest-path-algorithm, which actually lets the must-link constraints also influence other item pairs. Then, all cannot-link pairs are set to a maximal value, which again destroys metric properties. However, it is argued that this does not prohibit using the matrix during clustering. In fact, it is identical to enforcing the constraints during clustering. No fixing is applied, i.e., cannot-link pairs only influence themselves.

Cohn et al. (2003) learn a feature weighting for a probabilistic model. Clustering is done with the EM algorithm. The weights are learned using a gradient descent. Schultz and Joachims (2004) and Finley and Joachims (2005) both use SVMs to learn a metric. Schultz and Joachims (2004) use relative comparisons between pairs of similarities to learn the matrix of a Mahalanobis distance. With these pairs of similarities an optimization problem is formalized and solved with an SVM algorithm. Finley and Joachims (2005) directly train a binary classifier that outputs an item pair's associated similarity value. This is done with the structural SVM algorithm. The learned similarity function is then used in correlation clustering.

2.2 Problems of constraint usage

Feasibility issues A number of interesting theoretical work concerning constraints was done by Davidson and Ravi. They analyzed the feasibility problem, i.e., the question whether a partition with k clusters can be derived without violating any constraints. Even though a solution might exist, the constrained clustering might not be able to determine this solution. They studied feasibility in connection with k-means (Davidson and Ravi 2005b, 2007a) and

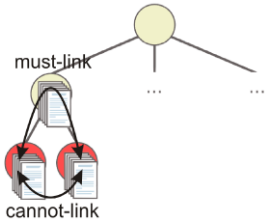

Fig. 3 Pairwise constraints for hierarchies: different constraints between the same objects on different hierarchy levels

HAC (Davidson and Ravi 2005a, 2009). In the latter work, they also consider the problem of irreducibility, which is the possibility to reach a dead-end during cluster merging, although a solution with a smaller number of clusters exists. In further work, they describe a heuristic to generate easy constraint sets from labeled data, i.e., constraint sets that do not suffer from the feasibility problem (Davidson and Ravi 2006). They also show that the number of clusters cannot be estimated efficiently from a constraint set and that using a too small value for k can be problematic for performance also in metric learning (Davidson and Ravi 2007b).

Beneficial constraints Although it is widely reported that introducing constraints largely improves clustering quality, not all constraints are equally suited to increase the effectiveness of clustering. Davidson et al. (2006) empirically show that not all constraints improve performance. They identify two properties of constraints which influence the clustering performance, i.e., informativeness and coherence. Informativeness describes how much information is provided that cannot be determined by the algorithm alone. Coherence is the amount of agreement between constraints themselves, given a certain metric. Other work addresses the different benefit of constraints through active learning that aims at identifying constraints that will provide the highest performance gain (Basu et al. 2004a; Klein et al. 2002).

Both problems are very important and also apply to the hierarchical constraints as discussed in this paper. Although it is out of the scope of this paper to discuss and analyze these problems in detail, they need to be kept in mind during algorithm design and evaluation of results.

2.3 Limitations for hierarchies

In a hierarchy of clusters, clusters are part of other, larger clusters. Therefore, items belong in fact to many clusters. As pairwise constraints are not related to a specific cluster and instead generally declare whether two items belong to the same or different clusters, they cannot distinguish between the different clusters an item can belong to in hierarchical clustering. Nevertheless, two items separated in specific clusters are always together in more general clusters—at least in the root (see Fig. 3). Therefore, pairwise constraints are not suitable in the hierarchical setting. This was already mentioned but not solved in the initial work on constrained clustering by Wagstaff and Cardie (2000). Kestler et al. (2006) also identified this problem. They use hierarchical agglomerative clustering and solve the problem by fixing the constraints to a certain dendrogram level. More specifically, they only consider constraints for the top-most dendrogram level. However, in most cases, it is not clear, on which dendrogram level the constraints should be applied.

Hierarchical learners vs. learning hierarchies The use of a hierarchical learner like hierarchical agglomerative clustering does not necessarily mean that the goal is to learn a

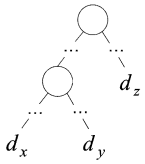

Fig. 4 A MLB constraint in the hierarchy

hierarchy. Those learners can as well be used to derive a cluster partition. However, this simplifies the task because a less complex target structure (partition instead of hierarchy) is derived and evaluated. There are many work, in which hierarchical clustering and pairwise constraints are analyzed (e.g., Davidson and Ravi 2005a, 2009; Bae and Bailey 2006; Klein et al. 2002). However, neither of this work aims at deriving a true cluster hierarchy. Therefore, these articles solve a different problem.

3 Hierarchical constraints

Because of the previously mentioned limitations of pairwise constraints, this work suggests another type of constraints that is suitable for hierarchical clustering tasks, the *must-link-before* (MLB) constraints. The goal is to stress the hierarchical order, in which data objects d_i are related. To this end, item pairs are replaced with item triples:

$$MLB_{xyz} = (d_x, d_y, d_z). \tag{1}$$

A constraint is labeled with *MLB* and has three indices relating to the three data objects involved. The order is important as it determines the exact hierarchical relationship according to an underlying cluster hierarchy H as defined in the following:

Definition 1 (Cluster Hierarchy) A cluster hierarchy H is a tuple $(C, \prec, root)$. C is the set of clusters. $\prec$ defines the hierarchical parent-child-relation between clusters, which is a strict partial ordering between the clusters. $root \in C$ is the root node of the hierarchy.

A constraint as in (1) expresses a relative relationship according to hierarchy levels of an underlying hierarchy H. This hierarchy need not to be specified explicitly. It can be implicitly given through the data. MLB constraints are a means to describe those implicit (or explicit) hierarchical relations. Specifically, MLB_{xyz} states that the data objects d_x and d_y are related on a more specific hierarchy level than the data objects d_x and d_z as shown in Fig. 4. This can be summarized in the following two properties:

Property 1 Given the MLB constraint MLB_{xyz}, any cluster c in the cluster hierarchy H that contains d_x and d_z also contains d_y: $\forall c \in H : d_x \in c \wedge d_z \in c \rightarrow d_y \in c$.

Property 2 Given the MLB constraint MLB_{xyz}, there exists at least one cluster c in the cluster hierarchy H, which only contains d_x and d_y but not d_z: $\exists c \in H : d_x \in c \wedge d_y \in c \wedge d_z \notin c$.

Based on the previous ideas, a must-link-before constraint is defined as follows:

Definition 2 (Must-Link-Before constraint) A Must-Link-Before (MLB) constraint MLB_{xyz} is a triple of items $MLB_{xyz} = (d_x, d_y, d_z)$, for which Properties 1 and 2 hold according to an underlying cluster hierarchy H.

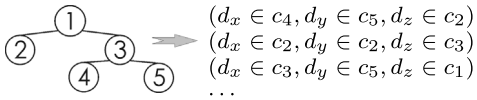

Fig. 5 Example of MLB constraint extraction from a hierarchy

Sources for constraints MLB constraints can come from different sources as shown through several examples in the introduction (see Sect. 1.1). Often they occur naturally (as item triples). Furthermore, existing hierarchical structures (like a folder hierarchy of documents or tagged documents in combination with an ontology between the tags) are a typical source of MLB constraints. Those hierarchies can be formalized through MLB constraints to be used as source of supervision in (clustering) algorithms. This is demonstrated in the following, where we describe how to extract the clustering constraints from a given training set and class hierarchy. A simple example is shown in Fig. 5. On the right side, some MLB constraints are shown, which were extracted from the hierarchy on the left. The first one expresses that a data object in class 4 is closer to a data object in class 5 than to a data object in class 2. The second constraint formalizes that two data objects of class 2 are closer to each other than to a data object in class 3. The third one states that a data object in class 3 is closer relative to a data object in class 5 than with a data object in class 1, which holds because 3 and 5 are grouped together on a more specific hierarchy level than 3 and 1. Of course, it is useful to extract all possible constraints in a systematic manner to gain as much information from the hierarchy as possible.

Generality Although MLB constraints specifically target hierarchical clustering, they can also be applied to flat clustering. A partition of clusters can be viewed as a very simple hierarchy with a root node and all clusters of the partition as direct subclusters of this root node. In this case, the following MLB constraint can be formulated: two data objects belonging to one cluster c_i must-link-before a data object of a different cluster c_j: $MLB_{xyz} = (d_x \in c_i, d_y \in c_i, d_z \in c_j)$. Hence, a cluster partition can be extracted as a special hierarchy.

However, please note the following difference between MLB constraints and pairwise constraints, which might impact the clustering behavior. MLB constraints add a further meta-relationship on top of the relative cluster assignment relationship defined by pairwise constraints. In fact, MLB constraints compare the strength of two must-link constraints stating that the must-link between the first and the second item is stronger than the must-link between the first and the third item. However, nothing is said about the quantity of the difference. Hence, MLB_{xyz} can describe data objects, which are just one hierarchy level away, but also data objects, which are much further apart, e.g., over 5 or 10 hierarchy levels. In fact, this means that MLB constraints enforce the creation of a hierarchy, which is at least detailed enough to ensure the hierarchical differences of data objects as described by the constraint set. However, they allow for an arbitrarily finer hierarchy as well.

In the case of a cluster partition, the partition (as represented through the simple hierarchy described above) is the least detailed hierarchy that can be learned. However, further sub-clusters are allowed through the constraint set and, therefore, could be possibly derived through a hierarchical clustering algorithm. However, if a partitioning algorithm is used, sub-clusters are not possible and the MLB constraint set exactly describes the partition like it is the case for pairwise constraints. In fact, MLB constraints can be transformed to pairwise constraints in this case (and only in this case). Specifically, one MLB constraint (d_x, d_y, d_z) can be transformed to one must-link constraint $con_=(d_x, d_y)$ and two cannot-link constraints $con_\neq(d_x, d_z)$ and $con_\neq(d_y, d_z)$. Therefore, if it is explicitly known in advance that the clus-

tering goal is a partition rather than a hierarchy, there is no need to use MLB constraints instead of pairwise constraints.

The MLB constraints have the same generality as must-link and cannot-link constraints as they also do not require knowledge about class labels. Formulating an MLB constraint merely requires to specify a relative relatedness between three data objects. As mentioned before, no statement is made on how strong this relationship is. A constraint can consider data objects which are close together—concerning an underlying hierarchy—as well as data objects very far apart. It might be useful to explicitly weigh the relationship, similar to soft must-link and cannot-link constraints (Wagstaff 2002). A weighted MLB constraint therefore associates a weight to a document triple:

$$MLB_{xyz} = (d_x, d_y, d_z, w). \tag{2}$$

However, constraint weighting is not further considered here. It is mentioned here as a side note for the sake of completeness and as a basis for further discussion in the future.

Transitivity of MLB constraints From a set of must-link constraints and a set of cannot-link constraints, one can transitively infer further constraints (Basu et al. 2008, Chap. 1). Likewise, transitive inference is possible for MLB constraints. This can increase the constraint set. Of course, such an inference requires non-conflicting constraints, i.e., no errors or noise in the given constraints. In the following, we discuss and prove three theorems describing transitive inference for MLB constraints.

Theorem 1 (Symmetry) *The first two data objects of an MLB constraint are symmetric*:

$$(d_1, d_2, d_3) \quad \Rightarrow \quad (d_2, d_1, d_3).$$

If d_1 and d_2 are related on a more specific hierarchy level than d_1 and d_3, this also implies that d_2 and d_3 are grouped together only after d_1 and d_2. This can be proven using the properties of MLB constraints given before.

Proof Given is the constraint (d_1, d_2, d_3). Therefore, it holds according to Properties 1 and 2 that

$$(1) \quad \forall c \in H : d_1 \in c \wedge d_3 \in c \rightarrow d_2 \in c \quad \text{and} \quad (2) \quad \exists c \in H : d_1 \in c \wedge d_2 \in c \wedge d_3 \notin c.$$

(d_2, d_1, d_3) is also a valid constraint, if it can be shown that both properties hold, i.e.,

$$(\text{I}) \quad \forall c \in H : d_2 \in c \wedge d_3 \in c \rightarrow d_1 \in c \quad \text{and} \quad (\text{II}) \quad \exists c \in H : d_2 \in c \wedge d_1 \in c \wedge d_3 \notin c.$$

The second property (II) holds as it is identical with the given fact (2). The first property (I) can be shown by contradiction. The question is: Does a cluster exist that contains d_2 and d_3 but not d_1? If so, the following assumption would be true:

$$(\text{a}) \; \exists c \in H : d_2 \in c \wedge d_3 \in c \wedge d_1 \notin c,$$

which contradicts (I). Data objects in a hierarchy that are assigned to the same node (at one certain hierarchy level) stay together in all ancestor nodes on more general hierarchy levels as given through the parent-child-relation $\prec$. Hence, they cannot be split up again.

Our previously made assumption (a) therefore implies that a cluster containing d_1 and d_2 but not d_3 cannot exist, because otherwise one would need to split up a node again:

$$\neg\exists c \in H : d_1 \in c \wedge d_2 \in c \wedge d_3 \notin c.$$

This violates fact (2). Therefore, the assumption made is wrong and (I) holds for (d_2, d_1, d_3). Hence, the symmetry theorem is proven. □

Further transitive inference is possible, if two MLB constraints are combined that overlap in two of three items.

Theorem 2 (Transitivity inside a subtree) *Given two MLB constraints with an identical data object to identify a subtree and an identical reference data object in a different subtree* (*at the third position*), *a third MLB constraint with the same reference data object can be inferred through*:

$$(d_1, d_2, d_4) \wedge (d_2, d_3, d_4) \quad \Rightarrow \quad (d_1, d_3, d_4).$$

Theorem 3 (Transitivity over different hierarchy levels) *Given are two MLB constraints with two identical data objects. The first data object identifies the intermediate hierarchy level and is positioned third in one constraint and first or second in the second constraint. The second data object identifies the deeper hierarchy level and is positioned first or second in both constraints. Then, a third constraint can be inferred between the lower and the higher level without the intermediate level*:

$$(d_1, d_2, d_3) \wedge (d_1, d_3, d_4) \quad \Rightarrow \quad (d_1, d_2, d_4).$$

Both theorems can be proven again based on the properties of MLB constraints. Let us start with proving Theorem 2.

Proof Given are the constraints (d_1, d_2, d_4) and (d_2, d_3, d_4). Therefore, it holds according to Properties 1 and 2 for both constraints that

$$\begin{aligned}
&(1) \quad \forall c \in H : d_1 \in c \wedge d_4 \in c \rightarrow d_2 \in c \quad \text{and} \\
&(2) \quad \exists c \in H : d_1 \in c \wedge d_2 \in c \wedge d_4 \notin c \quad \text{and} \\
&(3) \quad \forall c \in H : d_2 \in c \wedge d_4 \in c \rightarrow d_3 \in c \quad \text{and} \\
&(4) \quad \exists c \in H : d_2 \in c \wedge d_3 \in c \wedge d_4 \notin c.
\end{aligned}$$

(d_1, d_3, d_4) is also a valid constraint, if it can be shown that both properties hold, i.e.,

$$\begin{aligned}
&(\text{I}) \quad \forall c \in H : d_1 \in c \wedge d_4 \in c \rightarrow d_3 \in c \quad \text{and} \\
&(\text{II}) \quad \exists c \in H : d_1 \in c \wedge d_3 \in c \wedge d_4 \notin c.
\end{aligned}$$

To prove the first part, we can combine the given facts (1) and (3):

$$\forall c \in H : d_1 \in c \wedge d_4 \in c \overset{(1)}{\rightarrow} d_1 \in c \wedge d_4 \in c \wedge d_2 \in c \overset{(3)}{\rightarrow} d_1 \in c \wedge d_2 \in c \wedge d_3 \in c \wedge d_4 \in c,$$

which reduces to (I). To prove the second part, we need again the definition of a cluster hierarchy. So have in mind that items in a hierarchy that are assigned to the same node (at

one certain hierarchy level) stay together in all ancestor nodes on more general hierarchy levels as given through the parent-child-relation $\prec$. Without loss of generality, we assume that d_2 is grouped together with d_1 earlier than with d_3. The first cluster (on the most specific hierarchy level) containing both, d_2 and d_3, therefore also contains d_1. Because of (4), it cannot contain d_4, otherwise the cluster in (4) would not exist. Therefore, this cluster looks as follows:

$$d_1 \in c \wedge d_2 \in c \wedge d_3 \in c \wedge d_4 \notin c.$$

Property (II) applies for this cluster, which proves that such a cluster exists. Hence, the transitivity is proven. □

And finally, here is the proof for Theorem 3.

Proof Given are the constraints (d_1, d_2, d_3) and (d_1, d_3, d_4). Therefore, it holds according to Properties 1 and 2 for both constraints that

(1) $\forall c \in H : d_1 \in c \wedge d_3 \in c \rightarrow d_2 \in c$ and (2) $\exists c \in H : d_1 \in c \wedge d_2 \in c \wedge d_3 \notin c$ and

(3) $\forall c \in H : d_1 \in c \wedge d_4 \in c \rightarrow d_3 \in c$ and (4) $\exists c \in H : d_1 \in c \wedge d_3 \in c \wedge d_4 \notin c$.

(d_1, d_2, d_4) is also a valid constraint, if it can be shown that both properties hold, i.e.,

(I) $\forall c \in H : d_1 \in c \wedge d_4 \in c \rightarrow d_2 \in c$ and (II) $\exists c \in H : d_1 \in c \wedge d_2 \in c \wedge d_4 \notin c$.

To prove the first part, we can combine the given facts (3) and (1):

$$\forall c \in H : d_1 \in c \wedge d_4 \in c \overset{(3)}{\rightarrow} d_1 \in c \wedge d_4 \in c \wedge d_3 \in c \overset{(1)}{\rightarrow} d_1 \in c \wedge d_2 \in c \wedge d_3 \in c \wedge d_4 \in c,$$

which reduces to (I). To prove the second part, we start with fact (4) and apply fact (1), which yields:

$$\exists c \in H : d_1 \in c \wedge d_3 \in c \wedge d_4 \notin c \wedge d_2 \in c,$$

which can be reduced to (II). Hence, the transitivity is proven. □

The MLB constraints introduced in this section can be used as basis for constraining hierarchical clustering. In the following approach, it is assumed that Theorems 1–3 are applied to an initially given constraint set to get the most out of it.

4 Constrained hierarchical clustering

In the following, we present an instance-based approach that directly integrates MLB constraints into hierarchical clustering. In general, this approach addresses the task of constrained hierarchical clustering as defined in Definition 3 beneath.

Definition 3 (Constrained Hierarchical Clustering) Given is a data collection D that shall be structured. Furthermore, a set of must-link-before constraints *MLB* on some data object triples on D is given. The constrained hierarchical clustering task is to build a cluster hierarchy $H = (C, \prec, root)$ and find a mapping for the data objects in D to the clusters in C such that *MLB* is not violated.

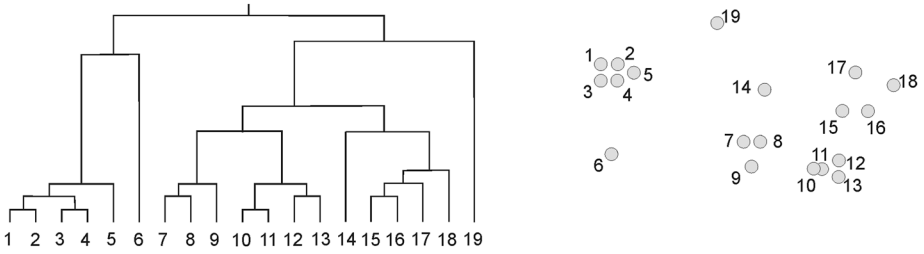

Fig. 6 Example of a dendrogram for the two-dimensional dataset *on the right*. The similarity of combined clusters is often displayed through the height of the bars that represent a grouping of two clusters

We extended the hierarchical agglomerative clustering (HAC) algorithm (Hastie et al. 2009, Chap. 14.3.12) for this purpose. HAC is a natural choice for the hierarchical clustering problem as it directly extracts the hierarchical relations between data objects through building a dendrogram (see Fig. 6). However, the final hierarchy is very detailed since it splits the data until only individual objects remain in the leaf nodes and several intermediate splits are present. If a more coarse grained structure is desirable, e.g., for user interaction, this needs to be extracted in a postprocessing step as it was proposed by Bade et al. (2007) for unsupervised and semi-supervised clustering.

MLB constraints can be naturally integrated into HAC. In contrast to must-link and cannot-link constraints, they do not suffer from the problem that a dendrogram level needs to be specified (cf. Sect. 2.3). This allows for a straight forward integration of MLB constraints into HAC clustering using our instance-based approach iHAC presented in the following.

4.1 Integrating MLB constraints with iHAC

The HAC algorithm is modified in such a way that merges occur only in accordance with the given MLB constraints. HAC initially places each data object in an individual cluster and repeatedly merges the two closest clusters until a single cluster is left. The merging step is modified in iHAC such that not only similarity but also constraint violations are taken into account for determining the next two clusters to merge. In fact, the two most similar clusters are selected from those cluster pairs that do not violate any (or at least the fewest, as discussed later) constraints. This means for merging that for all constraints (d_x, d_y, d_z), the cluster containing d_z can only be merged with a cluster that either contains both, d_x and d_y, or neither. An example is given in Fig. 7. Here, six data objects shall be clustered, given the constraint (d_x, d_y, d_z). As shown in a), this forbids two merges in the first step, which are merging d_x with d_z and merging d_y with d_z. Assuming that d_w and d_x were most similar, the dendrogram after the first merge looks as in b). Again, two merges are forbidden. If the newly formed cluster is merged with d_y in the next step (c), the constraint cannot be violated anymore and all merges are allowed for the rest of the clustering. This includes the merge with d_z as shown in d). A pseudocode description of the overall iHAC algorithm is shown in Fig. 8.

Please note that the proposed method is just one way to integrate constraints into the clustering. The strict enforcement of constraints as done by us is based on the assumption that it is of high importance to follow user given constraints. In a practical scenario, the user will expect to find his provided input reflected in the results. However, there might be other scenarios, where strict enforcement is less important, especially if constraints are generated through an automated method and might contain errors. In this case, it might be useful to

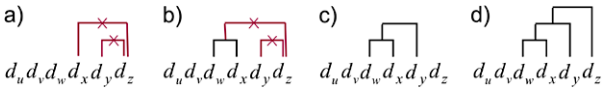

Fig. 7 Example of cluster merges in iHAC under the constraint (d_x, d_y, d_z) (**a**) after initialization, (**b**) after first, (**c**) second, and (**d**) third merge

iHAC(D, MLB)
Initialize dendrogram DG by adding the lowest level $C_0 = \{c_1, \ldots, c_n\}$ with $\forall d_i \in D : c_i = \{d_i\}$
for $l = 1$ to n **do**
Choose two clusters $c_1, c_2 \in C_{l-1}$ and merge them: $c_m = c_1 \cup c_2$, whereby the merge of c_1 and c_2 violates the fewest constraints in MLB and from all cluster pairs satisfying this condition c_1 and c_2 are the most similar
$C_l = (C_{l-1} \setminus \{c_1, c_2\}) \cup \{c_m\}$
Add C_l to DG
end for
return DG

Fig. 8 The iHAC algorithm

define a trade-off function between constraint enforcement and item similarity. However, this is not further analyzed in this paper.

If constraints are not directly conflicting with each other,[1] there always exists a solution with a complete dendrogram that does not violate any constraints. If the constraints were created from a given class hierarchy, they are always non-conflicting. Previous work on must-link and cannot-link constraints by Davidson and Ravi (2005a, 2005b, 2007a, 2009) indicates, however, that a solution with no constraint violations might not be found by the constrained clustering approach even though it exists. This is referred to as feasibility problem. For HAC, this would mean that the clustering would stop in a dead end before reaching the root node. This means that any merge possible at this moment would violate a constraint. This is referred to as irreducibility (cf. Sect. 2.2).

Considering MLB constraints, irreducibility can also occur. An example is given in Fig. 9. Here, six data objects shall be clustered, given the constraints (d_x, d_y, d_z) and (d_u, d_v, d_w). The top row shows an iteration of the algorithm that reaches a dead end with three clusters. Nonetheless, several solutions with a complete dendrogram exist. One is depicted at the bottom of Fig. 9. The example shows that irreducibility is possible, but how severe is this problem for MLB constraints? So far, no specific experiments were performed to further study this question. However, in all experiments performed with iHAC by us up to this point, the case of a dead end never occurred. It is our hypothesis that the problem of irreducibility is less significant for MLB constraints than for must-link and cannot-link constraints as a higher number of "unfavorable" merges are required to produce a dead end solution.

Nevertheless, a dead end is possible. As we judge the overall improvement of the clustering to be more important than the strict enforcement of all constraints, we weakened the condition of the merging step to choose the merge with the fewest constraint violations. This ensures that also in the case of irreducibility a complete dendrogram is built.

[1] Two conflicting constraints are, e.g., (d_x, d_y, d_z) and (d_x, d_z, d_y).

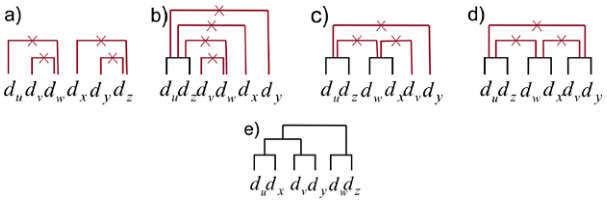

Fig. 9 Example of irreducibility in iHAC under the constraints (d_x, d_y, d_z) and (d_u, d_v, d_w); (**a**)–(**d**) iterations of iHAC with a dead end; (**e**) solution with a complete dendrogram; please note that the item order changes for a better visualization of the dendrogram

4.2 An efficient data structure

A critical issue for the overall runtime complexity of iHAC is whether the two clusters to be merged in each iteration can be determined and merged efficiently. Clearly, the straight forward approach of counting in each iteration how many constraints are violated for all data object pairs is very time consuming. Instead, the following data structure was developed, which is inspired by the similarity matrix from standard HAC. However, instead of pure similarity values, the *constrained matrix CM* includes more complex data. Each matrix entry $cm_{i,j}$ consists of the similarity value $sim_{i,j}$ and a set $cv_{i,j}$ for storing the constraints that are violated by a merge of the two clusters. For this purpose, constraints between data objects are transformed to constraints between clusters. As HAC starts with clusters containing only a single data object, the initial constraints can directly be expressed through replacing data objects by clusters. During the creation of the dendrogram, the interest is only on the currently highest dendrogram level. Therefore, constraints are passed on to clusters on the higher levels as long as they are relevant. This process is described below in more detail.

On that account, the set $cv_{i,j}$ contains all constraints that have the two clusters c_i and c_j at first and third position (i.e., both, MLB_{ixj} and MLB_{jxi}). These two kinds of constraints hinder the algorithm to merge the clusters c_i and c_j as this would violate the constraints. Rather, the merge between the cluster at the first position (either c_i or c_j) and the cluster at the second position (a third, different cluster c_x) has to be done first by the algorithm. Please note that we can omit the exact item order in the constraint and create a symmetric matrix. For the algorithm, we only need to know, how many constraints are violated for a certain cluster merge and not the specific constraints. To be able to detect, which constraints are already solved and therefore no longer active, we need to know the involved third cluster c_x. This is explained later. Therefore, we store in the set $cv_{i,j}$ the clusters c_x of all violated constraints together with the number of constraints associated to these clusters. The latter is necessary as there can be several constraints between the same set of clusters, especially on higher dendrogram levels. As the number of violated constraints is important for the merging criterion, it is stored.

Summarizing, the constrained matrix *CM* can be described as in (3) below. It is furthermore important to note that it is not necessary to compute the complete matrix. The upper triangular matrix is sufficient because the constrained matrix is symmetric. Additionally, it is beneficial to have an index vector *cmi* that points to the best entry of each row as shown on the right of (3) (Day and Edelsbrunner 1984; Cathey et al. 2007). For standard HAC that would be the entry of largest similarity. For iHAC, this is the entry with the fewest constraint violations and highest similarity in the case of the same number of violations. As shown by Day and Edelsbrunner (1984), time complexity can be further reduced, if priority queues are used for each row of the matrix, however, with an increased memory complexity. In the

removeCluster(CM, cmi, $index_{rem}$)
  for all rows $r < index_{rem}$ **do**
    Delete $cm_{r,index_{rem}}$ from CM
    if $cmi_r > index_{rem}$ **then**
      $cmi_r := cmi_r - 1$
    else if $cmi_r = index_{rem}$ **then**
      Go through all entries in row r of CM to find new value of cmi_r
    end if
  end for
  Delete row $index_{rem}$ from CM and cmi

Fig. 10 An algorithm for removing a cluster from the constrained matrix in iHAC

implementation for this paper, the index vector cmi was used. The constrained matrix CM looks as follows:

$$CM = \begin{pmatrix} - & cm_{1,2} & \cdots & \cdots & cm_{1,n} \\ - & - & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ \vdots & & & - & cm_{n-1,n} \\ - & \cdots & \cdots & - & - \end{pmatrix}, \qquad cmi = \begin{pmatrix} cmi_1 \\ \vdots \\ cmi_n \end{pmatrix} \tag{3}$$

with

$$cm_{i,j} = (sim_{i,j}, cv_{i,j}) \tag{4}$$

and

$$cv_{i,j} = \left\{(c_x, n_x) | (\exists MLB_{ixj} \vee \exists MLB_{jxi}) \wedge n_x = \text{count}(MLB_{ixj}) + \text{count}(MLB_{jxi})\right\}. \tag{5}$$

With the constrained matrix CM, the two clusters for the next cluster merge can be efficiently determined with linear complexity ($O(n)$), because a single iteration over the index cmi is sufficient. Updating the matrix to execute the merge is, however, a little more difficult. It consists of two steps: deleting the columns and rows of the two old clusters c_{old1} and c_{old2} and adding a row and column for the new cluster c_{new}.

For deleting a cluster from the matrix, the specific row in the matrix must be deleted. Furthermore, all rows above the one to remove contain one element concerning this cluster. All rows below do not include such an entry, because this belongs in the lower triangular matrix, which is not stored. Removing an entry from the row might require an update of the index vector cmi. In the worst case, an iteration over all elements of that row is needed to find the new best value. Figure 10 summarizes the algorithm for removing a cluster from the matrix. The time complexity of this algorithm is between $O(1)$, if the first row is deleted, and $O(n^2)$, if the last row is deleted and all other rows require a re-computation of the index vector.

For adding the merged cluster to the matrix, a new element is added to each row. This new element can be computed based on the old values of the two former clusters. The similarity value is updated as in standard HAC. For the group average linkage method (also called UPGMA), which we used in our implementation, this is done as shown in the algorithm in Fig. 11 (Manning and Schütze 1999, Chap. 14). The new set of constraint violations can

addCluster(CM, cmi, $cm_{*,old1}$, $cm_{*,old2}$, c_{old1}, c_{old2}, c_{new})
  for all rows r of CM **do**
    Add new element $cm_{r,n+1}$ at the end of row r
    Set $sim_{r,n+1}$ as in HAC; for the UPGMA methods this means:
    $sim_{r,n+1} = (sim_{r,old1}|c_{old1}| + sim_{r,old2}|c_{old2}|)/(|c_{old1}| + |c_{old2}|)$
    for all constraint violations $cv' = (c', n')$: $cv' \in cv_{r,old1} \vee cv' \in cv_{r,old2}$ **do**
      if c' is not on the highest dendrogram level **then**
        Replace c' by its ancestor cluster on the highest dendrogram level
      end if
      if $c' = c_{new}$ **then**
        Ignore cv' for this and all higher dendrogram levels, because it is no longer relevant
      else
        Add (c', n') to $cv_{r,n+1}$
      end if
    end for
    if $cm_{r,n+1}$ is better than cm_{r,cmi_r} **then**
      $cmi_r = n + 1$
    end if
  end for
  Add new (empty) row $n + 1$ to CM and cmi

Fig. 11 An algorithm for adding a cluster to the constrained matrix in iHAC

basically be created by merging the two old sets. However, constraints that are no longer relevant (because the first two items were already clustered and the constraint can therefore not be violated anymore) are removed from the matrix. This is the case, if the stored constraint clusters are descendant clusters of the new cluster based on the part of the dendrogram that was already built. As the stored clusters in the cv values not necessarily refer to the highest dendrogram level, it is necessary to update them. For this purpose the stored clusters are replaced by the most general ancestor elements on the highest dendrogram level built so far. If the most general ancestor element is the newly formed cluster, we can remove this element from cv. Otherwise, we update cv by replacing the stored cluster with the most general ancestor element.

The update of the index vector cmi is fairly easy, because it only changes, if the new cluster has a better value than the best current one (in each row). The entire algorithm is shown in Fig. 11. This algorithms has a time complexity between $O(n)$ and $O(n^2)$, depending on whether constraints are relevant for this merge or not. The worst case occurs, if the considered clusters have constraints to all other remaining clusters in the currently highest dendrogram level.

Based on the constrained matrix and the algorithms presented above, the overall iHAC implementation has a time complexity between $O(n^2)$ and $O(n^3)$. This is identical to the time complexity of the standard HAC implementation (Day and Edelsbrunner 1984). Hence, with the constrained matrix, a time complexity very similar to the baseline given by HAC can be achieved. However, for an increased number of constraints, an increase in runtime can be expected on average. Section 5.4 further compares runtime complexity with experimental measurements.

5 Evaluation

This section presents the experiments that were performed to evaluate the iHAC method. First, we discuss measures for evaluation in hierarchical clustering and describe the selected measures. Then, we describe the experimental setup, introducing the evaluation data used and experiment preparation. The following sections present the experimental results in terms of clustering quality and runtime.

5.1 Hierarchy evaluation measures

Evaluation of clustering methods can be divided in two different approaches: evaluation based on *internal* measures and evaluation based on *external* measures. *Internal measures* aim at evaluating a clustering solely based on the dataset itself and the clustering solution. These measures are usually based on a comparison between intra-cluster and inter-cluster similarity. Items assigned to the same cluster should be as similar as possible while items assigned to different clusters should be as dissimilar as possible. In contrast to this, *external measures* use a given external structure that describes the underlying cluster structure in the data that shall be recovered by the clustering method. The clustering result is compared to this gold standard to measure how well the method was capable of identifying the known structure. Evaluation with external measures therefore requires benchmark datasets as for classification. Recent discussions about evaluation measures for clustering approaches can be found in Borgelt (2005), Halkidi et al. (2002a, 2002b), Amigó et al. (2009).

Here, we use external evaluation measures as the semi-supervised character of the approach suggests that a certain structuring exists in the data and shall be uncovered. Hints on this structure are provided in the form of constraints or labeled data. Therefore, it is reasonable to measure how well this structure was uncovered. The "hidden" structure is the class structure in the benchmark dataset, on which basis the constraints are also generated. Although there exist several measures for comparing flat partitions, measures for the evaluation of hierarchies are rare. In the following, we present the measures used by us and a review of alternatives.

5.1.1 Hierarchically applied F measure

The F measure (van Rijsbergen 1979) is a widely applied evaluation measure for classification and external clustering evaluation (Sebastiani 2002). This class specific measure is a trade-off between precision and recall. The precision of a class c is the fraction of all items predicted to belong to c that also actually belong to c. In contrast to this, the recall of c is the fraction of all items belonging to this class that are also predicted as such.

To be able to apply the F measure to clustering, a mapping between the created clusters and the classes in the dataset is necessary. There are two ways to gain such mapping: either a class is assigned to every cluster or a cluster is assigned to every class. The first case allows that a class is represented through multiple clusters. However, the goal should be that a class can be represented through a single cluster in the hierarchy. Therefore, we applied the second method, i.e., for each class in the reference hierarchy, a cluster was selected from the dendrogram based on which the F-Score of this class is computed. Specifically, from all clusters in the dendrogram the one with the highest F-Score on the class at hand is chosen. For higher level (i.e., non-leaf) classes, all items contained in subclasses are also counted as belonging to this class. Please note that this simple procedure might select clusters inconsistent with the reference hierarchy or multiple times for different classes in the case

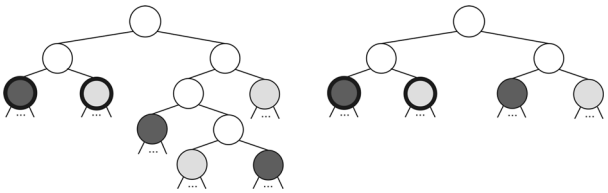

Fig. 12 Example for F-Score in hierarchical clustering: both hierarchies would gain the same F-score based on the bold, selected clusters for the two classes although the right, non-selected branch differs

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.

In our evaluation, we computed the F-Score only on the unlabeled data, i.e., the data not used for constraint generation. This is done to estimate the gain on new data and not optimistically biasing the measure. As the F measure is a class specific value, average values are computed for overall hierarchy comparison. Here, we distinguished between the average of all leaf node classes and the average of all non-leaf node classes to evaluate the effects on different levels of the hierarchy.

Applying the F measure as described potentially evaluates only a part of the cluster hierarchy because it just considers the best cluster per class. Some branches of the learned dendrogram might not be part of any cluster selected. As an example consider the two hierarchies in Fig. 12. Two classes exist in the reference hierarchy (which is not depicted). Both hierarchies in the picture are possible solutions of a clustering algorithm. The left branch in both hierarchies is identical. The bold circled clusters were selected for F-Score computation. Therefore, the F-score on both hierarchies is identical. The right branch of both hierarchies is not considered in this computation. Nevertheless, both hierarchies differ largely in this part. While the right branch of the left hierarchy is very noisy, the right branch in the right hierarchy again nicely splits both classes. That is why the right hierarchy should be considered as structuring the data better. From a user's perspective, it might be sufficient to focus on the best cluster for each class as a user would anyway focus on a single cluster that fits his information need. And the better this single cluster, the better for the user. For comparing the performance of different algorithms, it is nonetheless interesting to evaluate the complete hierarchy to better grasp changes to the structuring by different methods. For this, a different evaluation measure is needed.

5.1.2 H-correlation

Such a measure that can compare two entire hierarchies is *H-Correlation* (Bade and Benz 2010). Similar to the Rand index (Rand 1971), which compares cluster partitions, the clustering is evaluated based on item assignment without requiring a node mapping between the hierarchies. The Rand index thereby looks at all item pairs. Two cluster partitions cluster two instances identically, if both partitions either assign both instances to the same cluster or to different clusters. The Rand Index is the fraction of all item pairs that are identically clustered by both partitions. Instead of item pairs, H-Correlation considers item triples. More specifically, H-Correlation measures the overlap of the MLB constraint sets that can be determined from both hierarchies. Based on these two sets, both, a symmetric and an asymmetric measure, can be defined. In its simplest form, H-Correlation can measure the overlap of the two sets. This can be extended by weighing individual triples differently. The symmetric H-Correlation H_s between the learned hierarchy $\mathcal{H}_l$ and the reference hierarchy

$\mathcal{H}_r$ from the dataset is defined as:

$$H_s(\mathcal{H}_l, \mathcal{H}_r) = \frac{\sum_{\tau \in MLB_l \cap MLB_r} (w_l(\tau) + w_r(\tau))}{\sum_{\tau \in MLB_l} w_l(\tau) + \sum_{\tau \in MLB_r} w_r(\tau)}, \tag{6}$$

whereby MLB_l and MLB_r are all must-link-before triples in the learned and reference hierarchy, respectively, and w_l and w_r assign a weight to a triple depending on the learned and reference hierarchy, respectively.

Furthermore, an asymmetric H-Correlation H_a is of interest, if one hierarchy is allowed to be more detailed than the other. As a dendrogram is always a more detailed hierarchy than a given class hierarchy, this measure is especially of interest for our evaluation. In fact, we want to measure, how much of the hierarchical relationship given through a hierarchy is reflected in the dendrogram, while neglecting that the dendrogram contains several more hierarchical relationships as it contains all pairwise merges performed by the clustering. Formally, a hierarchy $\mathcal{H}_l = (C_l, \prec_l, root_l)$ is more detailed than a hierarchy $\mathcal{H}_r = (C_r, \prec_r, root_r)$, if:

$$C_l \supset C_r \wedge (\forall c_i, c_j \in C_r : c_i \prec_r c_j \rightarrow c_i <_l c_j) \wedge root_l \geq_l root_r, \tag{7}$$

whereby $\prec$ refers to a direct parent-child-relation and $<$ refers to the ancestor relation. Then the asymmetric H-Correlation H_a measures how many must-link-before triples from the reference hierarchy $\mathcal{H}_r$ can also be found in the learned hierarchy $\mathcal{H}_l$:

$$H_a(\mathcal{H}_l, \mathcal{H}_r) = \frac{\sum_{\tau \in MLB_l \cap MLB_r} w_r(\tau)}{\sum_{\tau \in MLB_r} w_r(\tau)}. \tag{8}$$

The most simple triple weighting function w gives equal weight to all triples. However, there are a lot more triples describing relations in the upper levels of the hierarchy than triples describing specific relations. Therefore, this simple weighting makes the value of H-Correlation strongly dependent on whether the top-level separations were done correctly. The following triple weight gives equal weight to each hierarchy node:

$$w\big((d_1, d_2, d_3)\big) = \frac{1}{|\{(d_1', d_2', d_3') | ca(d_1, d_3) = ca(d_1', d_3')\}|} \tag{9}$$

with $ca(a, b)$ is the most specific common ancestor node of a and b in the hierarchy. Through this weighting, the impact of all triples on the overall value is normalized based on the most general node that is passed by the triple, which is the common ancestor of the first and the third item. Experimental evaluation of the behavior of H-Correlation and the different weightings can be found in the paper by Bade and Benz (2010). Based on these findings, we use the asymmetric H-Correlation using the triple weighting in (9) in this paper.

5.1.3 Alternative measures for hierarchical comparison

Bade and Benz (2010) give a current review on existing measures for comparing hierarchies from the field of clustering as well as ontology engineering. This paper shows that hierarchies can be compared concerning different dimensions, i.e., the hierarchical structure, the assignment of items to this structure, and the assignment of labels (i.e., cluster names) to this structure. Measures that depend on cluster labels are not applicable for the evaluation of clustering methods because label generation is usually not part of the clustering.

Table 1 Banksearch dataset

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

However, label based measures are predominant in ontology engineering. An exception is the OntoRand index proposed by Brank et al. (2006) and its extended version proposed by Bade and Benz (2010). This measure is also based on the Rand index (Rand 1971). The OntoRand index softens the decision between identically and differently clustered items by using a distance function between clusters for gradual comparison taking into account how far apart two items are placed in the hierarchy. Although this measure is in principle an alternative for the hierarchy evaluation, it is not suited for our evaluation as it only allows for a symmetric hierarchy comparison. We need an asymmetric measure as we compare a more detailed dendrogram with a given class hierarchy.

Very recently Amigó et al. (2009) proposed the extended BCubed measure, which aims at evaluating overlapping partitions in general with hierarchies as a specific case. The original BCubed measure was defined by Bagga and Baldwin (1998) as an algorithm and presented as a measure by Amigó et al. (2009). It also uses item pairs, however, focuses on those that were placed in the same cluster in at least one of the two partitions. It consists of a precision and a recall component. The precision measures the fraction of item pairs assigned to the same cluster that are also in the same cluster in the reference hierarchy. Recall measures the opposite viewpoint by the fraction of item pairs assigned to the same cluster in the reference hierarchy that are also clustered together by the clustering algorithm. The extension for multiple class assignment then replaces individual cluster comparison by a comparison of sets of clusters in which an item pair occurs. Again, this measure compares hierarchies symmetrically and is therefore less suited for our evaluation.

5.2 Experimental setup

Evaluation was done with three datasets shown in Tables 1, 2, and 3. All consist of text documents structured in a class tree. These datasets are subsets from the publicly available banksearch dataset[2] consisting of websites and the Reuters corpus volume 1[3] consisting of news articles. Tables 1–3 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. We manually picked rather distinct topics from different parts of the Reuters hierarchy. In contrast to this, the Reuters 2 dataset contains classes that are more alike. This was done by picking only topics from one branch of the Reuters hierarchy. We randomly sampled a maximum of 100 documents per class, while a lower number in the final dataset means that there were less than 100 documents in the dataset.

All documents were preprocessed to obtain a $tf \times idf$ document vector representation (Salton and Buckley 1988). We selected features, 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 by Borgelt and Nürnberger (2004). To

[2] Available for download at the StatLib website (http://lib.stat.cmu.edu); Described in Sinka and Corne (2002).

[3] Available from NIST (http://trec.nist.gov/data/reuters/reuters.html).

Table 2 Reuters 1 dataset

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

Table 3 Reuters 2 dataset

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

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.

As the previous work on constrained clustering using pairwise constraints is not applicable on cluster hierarchies, we could not compare these methods with ours. Therefore, we compare in our experiments iHAC with the standard, unsupervised hierarchical agglomerative clustering (HAC) algorithm to show the effectiveness of the constraints. Hence, we show the difference between unsupervised and semi-supervised hierarchical clustering. Both HAC implementations use the group average linkage method (or UPGMA) as cluster similarity measure and the cosine similarity to determine document similarities (Manning and Schütze 1999, Chap. 14).

As mentioned earlier, we follow an evaluation approach with an external measure as usually applied for evaluation of supervised methods. This is reasonable because the goal of our semi-supervised clustering method is to uncover a specific cluster hierarchy. In the motivating example from the beginning, this is the hierarchy intended by the user. In our evaluation, these are the hierarchies given through the dataset. They are presumed to reflect the view of a certain user. To fulfill our given clustering task, the algorithm must derive a dendrogram that reflects the underlying hierarchy as much as possible.

The standard procedure for evaluation in such a setting is to split the data in training and test set. While the training set is used for learning a model (or in our case to define the prior knowledge on the target structure), the test set is used to evaluate the effectiveness of the algorithm (which is in our case the clustering of this data in relation to each other and the training data). This approach was also followed by us. The training data refers to the current user hierarchy as on the left side of Fig. 1. The test data is the collection of new documents yet unclustered as on the right side of Fig. 1. The overall class hierarchy in the complete data set describes the final user hierarchy, which is intended by the user, if he would organize all data on his own. Hence, this is the structure the algorithm shall also derive.

Both methods (HAC and iHAC) were evaluated with five different samples of training data (i.e., set of constraints) for each specific setup. The samples were drawn at random with a uniform distribution. The same samples were used for both algorithms to allow a fair comparison. Mean and variance of the results over these five runs were computed. In the following diagrams, only the mean will be shown for the sake of clarity. The variance was usually very small anyway. We sampled training data and generated all possible MLB constraints from it as described in Sect. 3, which were used to constrain the clustering.

Different numbers of labeled data per class were sampled. Specific attention was paid to small numbers of labeled data (ranging from 5 to 30 items) because it is much more likely from a practical point of view that labeled data is rare. Furthermore, different settings were created to reflect different distributions of constraints. Setting (1) assumes evenly distributed constraints. Labeled data was sampled from all classes. Hence, the hierarchy is already completely represented in the training data and new data always belongs to already known topics of the user. However, as discussed in the beginning of this article, it is much more likely that new data also contains new topics. Therefore, two more settings were evaluated for every dataset with parts of the hierarchy being unknown. In these settings, either a single leaf node class (setting (2)) or a whole subtree (setting (3)) of the hierarchy was not used to sample labeled data. Through this, constraints are unevenly distributed. Those scenarios are even more challenging than the first one.

5.3 Results: clustering quality

As discussed in Sect. 5.1, clustering quality was measured with H-Correlation and F-Score. The results are shown in Fig. 13 and 14. At first, we want to discuss setting (1), where constraints were sampled equally from all classes (row 1 and 4 in Fig. 13 and row 1 in Fig. 14). The results show that iHAC can successfully integrate domain knowledge into the hierarchical clustering, improving its performance towards a specific reference structure. Looking at the overall clustering performance as measured with H-Correlation, it can be observed that with an increasing number of constraints cluster quality increases. Depending on the data, the gain reached varies. While constraints strongly improve clustering quality of the banksearch and the reuters 2 dataset, the reuters 1 data rarely benefits from them. This indicates that the nature of the data and especially its feature representation influences the effectiveness of the constraints. In fact, the features of the reuters 1 dataset cannot explain the top level class separation as the documents in the different subclasses have hardly anything in common (from a bag of words point of view). Hence, constraints cannot compensate an inappropriate feature set.

Looking at the different hierarchy levels separately with the F measure, different behavior can be observed. iHAC is very successful on the higher levels, because it causes the larger, already merged subclusters to be grouped together correctly for the higher levels (if this would have been done falsely by the original clustering). A few constraints are sufficient to achieve this, because the subclusters are already built. Hence, an increasing number does not bring more benefit. In contrast to this, a constant increase of performance can be measured on the leaf level.

After having evaluated the case of evenly distributed constraints, let us now analyze the impact of an increasing imbalance in the available constraints, e.g., as caused through a partially known hierarchy. These results are shown in row 2 and 5 in Fig. 13 and row 2 in Fig. 14 for setting (2) and in row 3 and 6 in Fig. 13 and row 3 in Fig. 14 for setting (3). In general, the results with unevenly distributed constraints are similar to the results with evenly distributed constraints in terms of differences between the datasets, the number of labeled data, and sensitivity to the hierarchy level. The overall observed performance gain is usually a bit smaller, which is reasonable because unknown classes also mean fewer constraints.

The most notable difference can be found in the F-Score of the higher hierarchy levels. With an increasing number of unknown parts of the hierarchy, the performance can strongly drop for the higher hierarchy level, even below the original baseline given through the unconstrained HAC algorithm. Furthermore this problem gets more severe with more constraints. The reason for this can be found in the erroneous clustering of some instances

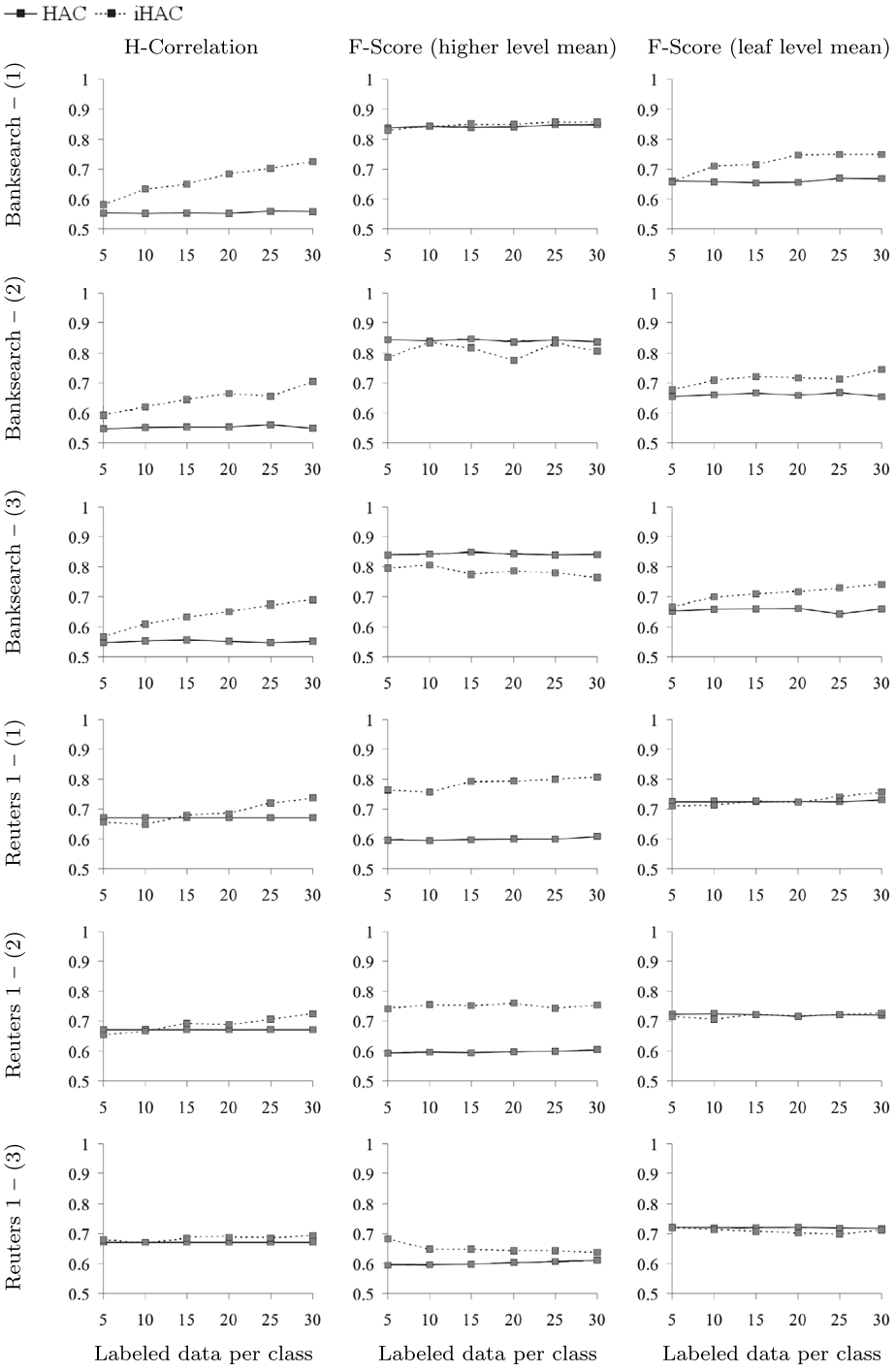

Fig. 13 Results for the Banksearch and the Reuters 1 data set

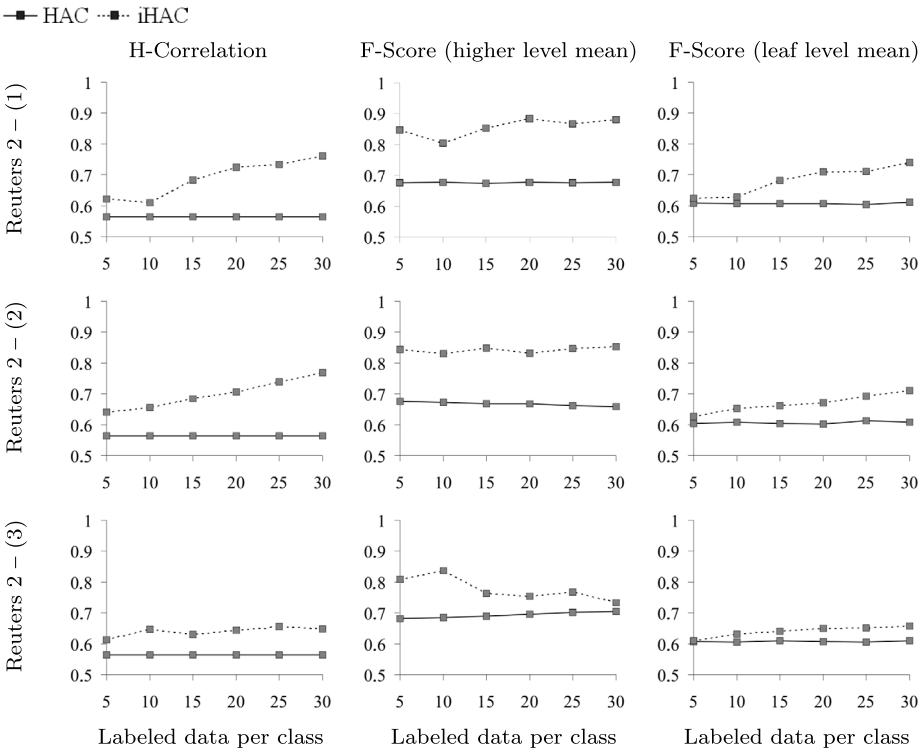

Fig. 14 Results for the Reuters 2 data set

by the HAC algorithm. Even if a single item (known to belong to a certain class) is merged falsely with a cluster representing an unknown class, the resulting cluster is merged according to this single item and not according to the dominating class which is unknown. Due to this, a whole subcluster can be merged to the wrong subtree, which terribly reduces its F-Score measure.

5.4 Runtime performance

Besides clustering quality, runtime and scalability is usually an important criterion for the applicability of a method. It was already said before that hierarchical agglomerative clustering has a rather high computational complexity. Our implementation has a complexity between $O(n^2)$ for the best case and $O(n^3)$ for the worst case, which, however, is very unlikely (cf. Sect. 4.2). The same bounds hold for iHAC. This section shall show the differences in the average runtime performance through actual time measurements to discuss the impact of the integration of constraints.

Figure 15 shows runtime measurements for the clustering experiments on the Reuters 2 dataset. The measurements were done on a PC with a Intel Core 2 Duo CPU E6750 with 2.66 GHz, 4 GB RAM, a Windows XP 32-bit operating system, and a Java 6 virtual machine. However, please note that the experiments were not done under completely identical conditions. Most importantly, the work load of the PC was varying. Therefore, the displayed numbers might vary to some (rather small) extent because of this. However, the tendency of the runtime behavior is still clearly visible.

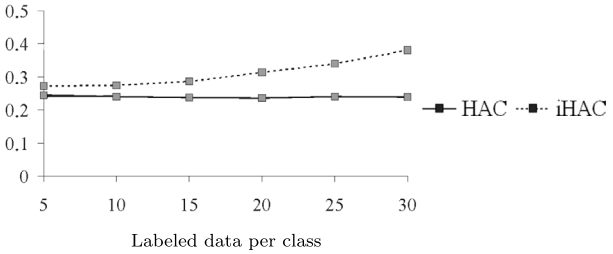

Fig. 15 Measured runtime of the clustering (in minutes)

It can be noted that the average runtime of iHAC increases with an increasing number of constraints. This is also expected because iHAC requires additional runtime to keep track of the forbidden merges due to constraint violations. The used constrained matrix (cf. Sect. 4.2) can determine the next (allowed and best) cluster merge with the same efficiency as the HAC implementation itself. However, this comes at the price that updating it after each merge might but does not necessarily require some additional effort. The more constraints are involved, the more time is needed for the update as can be seen from the time measurements. Nevertheless, the additional effort is very small in comparison to the number of constraints involved. For this, you should keep in mind that a linear increase in the number of labeled data causes an exponential increase in constraints. As an example, consider the hierarchy of the Reuters 1 dataset. Five labeled items per class generate 37,200 constraints, which increases to 307,800 constraints in the case of ten labeled items per class. Summing up, the developed iHAC algorithm allows for a quite efficient integration of constraints into the clustering process.

6 Conclusion

In this paper, we presented MLB-constraints which are intended to constrain the formation of a cluster hierarchy through some prior knowledge on the target hierarchy. We discussed the limitations of typical pairwise constraints on this problem and formally introduced MLB-constraints and their properties. In the second part of the paper, we focused on an instance-based constrained clustering approach to integrate these constraints into hierarchical agglomerative clustering. While the approach itself is rather simple, we focused on efficiency, which is very important for handling large constraint sets as typical in the hierarchical scenario. We evaluated this approach in terms of both clustering quality and runtime performance. With three different datasets, we showed that MLB constraints can successfully increase the clustering quality with the instance-based approach iHAC, while requiring only a little extra runtime. However, if parts of the target hierarchy are unknown during clustering and not represented by the constraint set, small cluster errors can be intensified through the constrained clustering. To tackle this problem, we suggest the development of metric-based methods based on MLB constraints. Some ideas for such approaches can be found in our previous work on the topic (Bade and Nürnberger 2006, 2008, 2009). These approaches are not the focus of this paper. However, it shall be mentioned that combining both types of approaches showed to be most beneficial. Having an efficient instance-based method as presented in this paper is therefore a prerequisite for further experiments on combined methods.

Acknowledgements This work was partially supported by the German Research Foundation (DFG) with contract no. NU131/1-3.

References

Amigó, E., Gonzalo, J., Artiles, J., & Verdejo, F. (2009). A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information Retrieval*, *12*, 461–486.

Bade, K., & Benz, D. (2010). Evaluation strategies for learning algorithms of hierarchies. In *Advances in data analysis, data handling and business intelligence, proceedings of the 32nd annual conference of the German classification society (GfKl'08), studies in classification, data analysis, and knowledge organization* (pp. 83–92).

Bade, K., & Nürnberger, A. (2006). Personalized hierarchical clustering. In *Proceedings of the 2006 IEEE/WIC/ACM international conference on web intelligence* (pp. 181–187). Washington: IEEE Computer Society.

Bade, K., & Nürnberger, A. (2008). Creating a cluster hierarchy under constraints of a partially known hierarchy. In *Proceedings of the 2008 SIAM international conference on data mining* (pp. 13–24).

Bade, K., & Nürnberger, A. (2009). Learning a metric during hierarchical clustering based on constraints. In *Proceedings of the LWA 2009 workshop*.

Bade, K., Hermkes, M., & Nürnberger, A. (2007). User oriented hierarchical information organization and retrieval. In J. N. Kok, J. Koronacki, R. L. de Mántaras, S. Matwin, D. Mladenic, & A. Skowron (Eds.), *Lecture notes in computer science: Vol.* 4701. *Proceedings of the 18th European conference on machine learning (ECML07)* (pp. 518–526). Berlin: Springer.

Bae, E., & Bailey, J. (2006). Coala: a novel approach for the extraction of an alternate clustering of high quality and high dissimilarity. In *Proceedings of the sixth international conference on data mining ICDM'06* (pp. 53–62). Washington: IEEE Computer Society.

Bagga, A., & Baldwin, B. (1998). Entity-based cross-document coreferencing using the vector space model. In *Proceedings of the 17th international conference on computational linguistics (COLING-ACL'98)* (pp. 79–85).

Bar-Hillel, A., Hertz, T., Shental, N., & Weinshall, D. (2003). Learning distance functions using equivalence relations. In *Proceedings of the 20th international conference on machine learning (ICML'03)* (pp. 11–18).

Bar-Hillel, A., Hertz, T., Shental, N., & Weinshall, D. (2005). Learning a mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research*, *6*, 937–965.

Basu, S., Banerjee, A., & Mooney, R. J. (2002). Semi-supervised clustering by seeding. In *Proceedings of the 19th international conference on machine learning (ICML'02)* (pp. 27–34).

Basu, S., Banerjee, A., & Mooney, R. (2004a). Active semi-supervision for pairwise constrained clustering. In *Proceedings of the 4th SIAM international conference on data mining* (pp. 333–344).

Basu, S., Bilenko, M., & Mooney, R. J. (2004b). A probabilistic framework for semi-supervised clustering. In *Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'04)*. (pp. 59–68).

Basu, S., Davidson, I., & Wagstaff, K. L. (Eds.) (2008). *Constrained clustering: advances in algorithms, theory, and applications*. London/Boca Raton: Chapman & Hall/CRC.

Bilenko, M., Basu, S., & Mooney, R. J. (2004). Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the 21st international conference on machine learning (ICML'04)* (pp. 81–88).

Borgelt, C. (2005). *Prototype-base classification and clustering*. Habilitation, Otto-von-Guericke-University Magdeburg.

Borgelt, C., & Nürnberger, A. (2004). Fast fuzzy clustering of web page collections. In *Proceedings of the workshop on statistical approaches to web mining (SAWM04) at PKDD04, ECML/PKDD organization committee*, Pisa, Italy (pp. 75–86).

Brank, J., Mladenic, D., & Groblenik, M. (2006). Gold standard based ontology evaluation using instance assignment. In *Proceedings of the 4th workshop on evaluating ontologies for the web (EON'06)*.

Cathey, R. J., Jensen, E. C., Beitzel, S. M., Frieder, O., & Grossman, D. (2007). Exploiting parallelism to support scalable hierarchical clustering. *Journal of the American Society for Information Science and Technology*, *58*(8), 1207–1221.

Choi, B., & Peng, X. (2004). Dynamic and hierarchical classification of web pages. *Online Information Review*, *28*(2), 139–147.

Cohn, D., Caruana, R., & McCallum, A. (2003). *Semi-supervised clustering with user feedback* (Technical Report TR2003-1892). Cornell University.

Davidson, I., & Ravi, S. S. (2005a). Agglomerative hierarchical clustering with constraints: theoretical and empirical results. In *Proceedings of the 9th European conference on principles and practice of knowledge discovery in databases (PKDD'05)* (pp. 59–70).

Davidson, I., & Ravi, S. S. (2005b). Clustering with constraints: feasibility issues and the k-means algorithm. In *Proceedings of the 2005 SIAM international data mining conference* (pp. 138–149).

Davidson, I., & Ravi, S. S. (2006). Identifying and generating easy sets of constraints for clustering. In *Proceedings of the twenty-first national conference on artificial intelligence and the eighteenth innovative applications of artificial intelligence conference* (pp. 336–341).

Davidson, I., & Ravi, S. S. (2007a). The complexity of non-hierarchical clustering with instance and cluster level constraints. *Data Mining and Knowledge Discovery*, *14*(1), 25–61.

Davidson, I., & Ravi, S. S. (2007b). Intractability and clustering with constraints. In *Proceedings of the 24th international conference on machine learning (ICML'07)* (pp. 201–208).

Davidson, I., & Ravi, S. S. (2009). Using instance-level constraints in agglomerative hierarchical clustering: theoretical and empirical results. *Data Mining and Knowledge Discovery*, *18*, 257–282.

Davidson, I., Wagstaff, K., & Basu, S. (2006). Measuring constraint-set utility for partitional clustering algorithms. In *Knowledge discovery in databases: PKDD 2006, 10th European conference on principles and practice of knowledge discovery in databases* (pp. 115–126).

Day, W. H. E., & Edelsbrunner, H. (1984). Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification*, *1*(1), 7–24.

Finley, T., & Joachims, T. (2005). Supervised clustering with support vector machines. In *Proceedings of the 22nd international conference on machine learning (ICML'05)* (pp. 217–224).

Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, *2*(2), 139–172.

Gonzalez, R. C., & Woods, R. E. (2007). *Digital image processing*. New York: Prentice-Hall.

Grira, N., Crucianu, M., & Boujemaa, N. (2004). Fuzzy clustering with pairwise constraints for knowledge-driven image categorization. In *European workshop on the integration of knowledge, semantics and digital media technology (EWIMT)* (pp. 299–304).

Halkidi, M., Batistakis, Y., & Vazirgiannis, M. (2002a). Cluster validity methods: Part I. *ACM SIGMOD Record*, *31*(2), 40–45.

Halkidi, M., Batistakis, Y., & Vazirgiannis, M. (2002b). Clustering validity checking methods: Part II. *ACM SIGMOD Record*, *31*(3), 19–27.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference and prediction* (2nd ed.). Berlin: Springer. http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

Jones, W. (2008). *Keeping found things found*. San Mateo: Morgan Kaufmann.

Jones, W., & Teevan, J. (Eds.) (2007). *Personal information management*. Seatle: University of Washington Press.

Kestler, H. A., Kraus, J. M., Palm, G., & Schwenker, F. (2006). On the effects of constraints in semi-supervised hierarchical clustering. In F. Schwenker & S. Marinai (Eds.), *LNAI: Vol.* 4087. *Artificial neural networks in pattern recognition* (pp. 57–66).

Khosla, R., Westfall, D. G., Reich, R. M., Mahal, J. S., & Gangloff, W. J. (2010). *Spatial variation and site-specific management zones* (1st ed., pp. 195–219). Berlin: Springer.

Kim, H., & Lee, S. (2002). An effective document clustering method using user-adaptable distance metrics. In *Proceedings of the 2002 ACM symposium on applied computing* (pp. 16–20).

Klein, D., Kamvar, S., & Manning, C. (2002). From instance-level constraints to space-level constraints: making the most of prior knowledge in data clustering. In *Proceedings of the 19th international conference on machine learning (ICML'02)* (pp. 307–314).

Manning, C. D., & Schütze, H. (1999). *Foundations of natural language processing*. Cambridge: MIT Press.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge: Cambridge University Press.

McKusick, K. B., & Langley, P. (1991). Constraints on tree structure in concept formation. In *Proceedings of the 12th international joint conference on artificial intelligence* (pp. 810–816).

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, *66*, 622–626.

van Rijsbergen, C. J. (1979). *Information retrieval* (2nd ed.). London: Butterworths.

Ruiz, C., Menasalvas, E., & Spiliopoulou, M. (2007a). Constraint-based query clustering. In *Advances in intelligent web mastering, proceedings of the 5th Atlantic web intelligence conference (AWIC'07)* (pp. 304–309).

Ruiz, C., Spiliopoulou, M., & Menasalvas, E. (2007b). C-dbscan: density-based clustering with constraints. In *Rough sets, fuzzy sets, data mining and granular computing; proceedings of the 11th international conference (RSFDGrC'07)* (pp. 216–223).

Ruß, G., & Kruse, R. (2011). Exploratory hierarchical clustering for management zone delineation in precision agriculture. In *LNAI: Vol.* 6870. *Proceedings of the industrial conference on data mining 2011* pp. 161–173). Berlin: Springer.

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, *25*(5), 513–523.

Schultz, M., & Joachims, T. (2004). Learning a distance metric from relative comparisons. In *Proceedings of neural information processing systems*.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM Computing Surveys*, *34*(1), 1–47.

Sinka, M., & Corne, D. (2002). A large benchmark dataset for web document clustering. In *Frontiers in artificial intelligence and applications: Vol.* 87. *Soft computing systems: design, management and applications* (pp. 881–890).

Wagstaff, K. (2002). *Intelligent clustering with instance-level constraints*. PhD thesis, Cornell University.

Wagstaff, K., & Cardie, C. (2000). Clustering with instance-level constraints. In *Proceedings of the seventeenth international conference on machine learning (ICML'00)* (pp. 1103–1110).

Wagstaff, K., Cardie, C., Rogers, S., & Schroedl, S. (2001). Constrained k-means clustering with background knowledge. In *Proceedings of 18th international conference on machine learning* (pp. 577–584).

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