# Clustering Trees with Instance Level Constraints

Jan Struyf[1] and Sašo Džeroski[2]

[1] Dept. of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, 3001 Leuven, Belgium
Jan.Struyf@cs.kuleuven.be
[2] Dept. of Knowledge Technologies, Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
Saso.Dzeroski@ijs.si

**Abstract.** Constrained clustering investigates how to incorporate domain knowledge in the clustering process. The domain knowledge takes the form of constraints that must hold on the set of clusters. We consider instance level constraints, such as must-link and cannot-link. This type of constraints has been successfully used in popular clustering algorithms, such as $k$-means and hierarchical agglomerative clustering. This paper shows how clustering trees can support instance level constraints. Clustering trees are decision trees that partition the instances into homogeneous clusters. Clustering trees provide a symbolic description for each cluster. To handle non-trivial constraint sets, we extend clustering trees to support disjunctive descriptions. The paper's main contribution is ClusILC, an efficient algorithm for building such trees. We present experiments comparing ClusILC to COP-$k$-means.

## 1 Introduction

Clustering methods partition a given set of instances into subsets (clusters) such that the instances in a given cluster are similar [1]. Traditional clustering algorithms, such as $k$-means and hierarchical agglomerative clustering (HAC), are unsupervised, that is, they only have access to the attributes describing each instance; no direct information about the actual assignment of instances to clusters is available. This distinguishes clustering from supervised classification, where the class of each instance is given.

Constrained clustering investigates how domain knowledge can improve clustering performance. Domain knowledge is given as a set of constraints that must hold on the clusters. We consider two common types of instance level (IL) constraints: must-link and cannot-link [2]. A must-link constraint ML($a$,$b$) specifies that instances $a$ and $b$ must belong to the same cluster, and a cannot-link constraint CL($a$,$b$) specifies that $a$ and $b$ must not be placed in the same cluster. IL constraints provide additional information about the assignment of instances to clusters. Clustering with IL constraints is therefore considered to be a form of semi-supervised learning.

IL constraints have been successfully incorporated in popular clustering algorithms, such as $k$-means [3,4,5,6] and HAC [7,8]. This paper shows how clustering

J.N. Kok et al. (Eds.): ECML 2007, LNAI 4701, pp. 359–370, 2007.

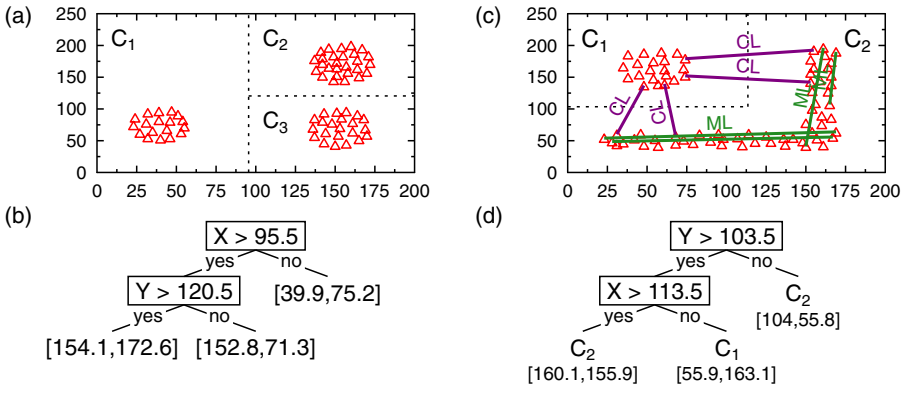


**Fig. 1.** (a) A simple data set with three clusters. (b) A clustering tree for (a). Each leaf is labeled with the cluster's centroid (the attribute-wise mean of the instances). (c) Data with must-link (ML) and cannot-link (CL) constraints. (d) A disjunctive clustering tree for (c), which takes the IL constraints into account.

trees can support IL constraints. Clustering trees are decision trees that are used for clustering [9] (Fig. 1.b). Each leaf of a clustering tree corresponds to a cluster and is labeled with the cluster's centroid. Similar to regular decision trees, the internal nodes of a clustering tree contain attribute-value tests. The main advantage of clustering trees is that they provide a symbolic description for each cluster (i.e., they perform conceptual clustering [10]). For example, cluster $C_2$ in Fig. 1.a is the set of instances for which $X > 95.5$ and $Y > 120.5$.

A disadvantage of clustering trees is that they only allow conjunctive cluster descriptions. This corresponds to rectangular clusters in the two-dimensional case (Fig. 1.a). One of the main goals of constrained clustering is dealing with non-trivial cluster shapes. In this paper, we therefore adapt clustering trees to support disjunctive cluster descriptions. To this end, we introduce cluster labels in the leaves of the clustering tree. All leaves that share the same label make up one cluster. We call a clustering tree with such labels a *disjunctive clustering tree*. For example, the L-shaped cluster $C_2$ in Fig. 1.c is represented by two leaves in Fig. 1.d and its disjunctive description is $Y \leq 103.5 \vee (Y > 103.5 \wedge X > 113.5)$. Note that this is similar to how classification trees represent disjunctive concepts, but here the labels are not given in the data.

## 2 Top-Down Induction of Clustering Trees

Clustering tree learning algorithms, such as Tilde [9] or Clus [11], are similar to top-down induction (TDI) algorithms for regular decision trees, such as C4.5 [12]. A TDI algorithm builds a tree starting from the root node in a depth-first manner. Given a set of instances, it considers all possible attribute-value tests, and selects the test $t^*$ that maximizes a certain heuristic function. Next,

it creates a new internal node, labels it $t^*$, and calls itself recursively to create a subtree for each subset in the partition induced by $t^*$ on the instances. If, at a given point, no suitable test can be found, then it creates a leaf.

To induce a clustering tree, the TDI algorithm computes the heuristic value of a test $t$ given instances $I$ as $H(t, I) = \mathrm{Var}(I) - \sum_{I_k \in \mathcal{P}(t,I)} \frac{|I_k|}{|I|} \mathrm{Var}(I_k)$, with $\mathrm{Var}(I)$ the variance of $I$, and $\mathcal{P}(t, I)$ the partition induced by $t$ on $I$. $H(t, I)$ takes all attributes into account, that is, $\mathrm{Var}(I)$ is the variance summed over all attributes. $H(t, I)$ guides the algorithm to a tree with homogeneous (low variance) leaves. If no test yields a significant reduction in variance, then the algorithm creates a leaf and labels it with the attribute-wise mean of the instances.

## 3 ClusILC

This section presents ClusILC, the main contribution of this paper. ClusILC is an algorithm that constructs a disjunctive clustering tree given a set of instances and a set of IL constraints. ClusILC performs soft constrained clustering [4,5,6], that is, the output is not guaranteed to satisfy all the given constraints.

### 3.1 ClusILC's Heuristic

Decision tree learners that follow the TDI approach (Section 2) employ a local heuristic: $H(t, I)$ only depends on the instances local to the node that is being constructed. ClusILC uses a global heuristic. Such a heuristic measures the quality of the entire tree and takes all instances into account. The heuristic that we propose for ClusILC is

$$H(T, I, IL) = (1 - \gamma) \cdot \frac{1}{\mathrm{Var}(I)} \sum_{l \in T} \frac{|I_l|}{|I|} \mathrm{Var}(I_l) + \gamma \cdot \frac{|\{c \in IL | \mathrm{violated}(T, I, c)\}|}{|IL|},$$

with $T$ the disjunctive clustering tree for which the heuristic is to be computed, $I$ the set of instances, $IL$ the set of IL constraints, and $I_l$ the instances in leaf $l$ of $T$. The first term of $H(T, I, IL)$ measures the average variance in the leaves of the tree, normalized by the data's total variance. The second term is the proportion of IL constraints that is violated by the tree. The heuristic trades off both terms by means of a parameter $\gamma$. Note that it is not possible to convert this heuristic into an equivalent local one because of the second term. This term cannot be split into a term for each leaf that only depends on local instances because IL constraints may link these instances to instances in other leaves.

### 3.2 ClusILC's Search Strategy

ClusILC (Fig. 2) searches greedily for a tree that minimizes $H(T, I, IL)$. It starts with a tree that consists of only a single leaf covering all instances. In each main loop iteration, it refines the current tree by replacing one of its leaves with a subtree consisting of a new test node and two new leaves. The candidate refined

```
procedure ClusILC(I,IL)
 1: T = leaf(I,C1,mean(I))
 2: h = H(T, I, IL)
 3: while true do
 4:     (T*, h*) = (null, h)
 5:     for each Tr ∈ Refine(T) do
 6:         hr = H(Tr, I, IL)
 7:         if hr < h* then
 8:             (T*, h*) = (Tr, hr)
 9:     if h* < h then
10:         (T, h) = (T*, h*)
11:     else
12:         return T
```

```
procedure Refine(T)
 1: R = ∅
 2: for each leaf l ∈ T do
 3:     Il = instances(l)
 4:     for each attribute a do
 5:         for each split point v do
 6:             t = "a > v"
 7:             (I1, I2) = apply(Il,t)
 8:             for each label pair (c1, c2) do
 9:                 l1 = leaf(I1,c1,mean(I1))
10:                 l2 = leaf(I2,c2,mean(I2))
11:                 n = node(t,l1,l2)
12:                 Tr = replace l by n in T
13:                 R = R ∪ {Tr}
14: return R
```

**Fig. 2.** The ClusILC algorithm

trees are constructed by the procedure Refine. ClusILC computes for each such tree its heuristic value and selects the one with the smallest heuristic value ($T^*$). If $T^*$ is better than the current tree, then $T^*$ becomes the current tree and the search continues. If, on the other hand, no refined tree is able to improve on the heuristic value of the current tree, then the search ends and the current tree is returned.

Refine computes the set of candidate refined trees $R$. It consists of four nested loops. The first loop iterates over all leaves of the current tree. For each such leaf, Refine considers all attributes that can be used to construct an attribute-value test. For each attribute, it considers all possible split points and constructs a test of the form $a > x$. This test introduces two new leaves in the tree, where each should be assigned a cluster label (Fig. 1.d). The label can be either a label that already appears in the tree, or it can be a new label. For each pair of such labels (one for each leaf), Refine creates an internal node with the test and the two leaves, and then uses this node to create a new refined tree.

Note that ClusILC does not follow the depth-first approach of most TDI algorithms. The reason is that such a search strategy is not suitable for optimizing a global heuristic. (For a local heuristic, both methods produce the same tree and TDI algorithms use depth-first construction because it is more efficient and easier to implement.)

The efficiency of the algorithm in Fig. 2 can be improved in several ways. The most obvious optimization is that the candidate generation and the evaluation part of the algorithm can be integrated. Instead of storing all candidate refined trees, ClusILC only stores the current best refinement. Each time a new refinement is generated, ClusILC immediately computes its heuristic value and updates the current best refinement if the new refinement is better. The next two sections discuss how to efficiently assign cluster labels and how to find the best split point for a numerical attribute.

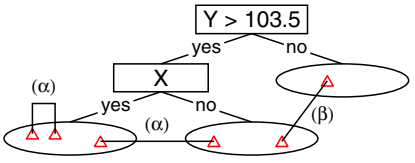


**Fig. 3.** We distinguish two constraint groups when refining the leaf marked "X": ($\alpha$) constraints either local to one of the new leaves, or connecting the two new leaves, and ($\beta$) constraints connecting one of the two new leaves to another leaf already in the tree

### 3.3 Assigning Cluster Labels

The second optimization is that ClusILC does not consider all pairs of cluster labels when generating refined trees. The reason is that the choices (select a label for each leaf) are mostly independent. We distinguish two groups of constraints when refining a leaf: $\alpha$ and $\beta$ (defined in Fig. 3). The number of violated constraints in group $\alpha$ does not depend on the actual labels $(c_1, c_2)$ of the new leaves (ClusILC only considers splits with $c_1 \neq c_2$). The heuristic value therefore only changes if $v_1^\beta(c_1) + v_2^\beta(c_2)$ changes, with $v_k^\beta(c)$ the number of violated group $\beta$ constraints when labeling leaf $k$ with $c$. Because the two leaves have disjoint sets of constraints in group $\beta$, the two terms in the sum can be optimized separately, that is, the optimal labels can be assigned sequentially to the leaves. The optimal label $c_k^*$ for leaf $l_k$ is the label that minimizes $v_k^\beta(c)$. There is, however, one problem with this approach: the constraint $c_1 \neq c_2$ introduces a dependency between the two labels. Therefore, two cases must be considered. The first case is to first select the optimal label $c_1^*$ for leaf $l_1$, then select the label $c_2 \neq c_1^*$ for leaf $l_2$ that minimizes $v_2^\beta(c_2)$. The second case is symmetrical, but starts with labeling $l_2$. ClusILC picks the case that minimizes $v_1^\beta(c_1) + v_2^\beta(c_2)$. The resulting labeling is the same as when the labeling would be obtained by considering all pairs $(c_1,c_2)$. This optimization makes the labeling step linear in the number of possible cluster labels (instead of quadratic).

### 3.4 Selecting a Split Point

When refining a given leaf, ClusILC considers for each attribute all possible split points in one pass over the leaf's instances. This is similar to how TDI algorithms such as C4.5 select the best split point for a numerical attribute. The main difference is that ClusILC computes a number of additional statistics to be able to efficiently compute, for each candidate split, the number of violated constraints. We first discuss the basic algorithm and then the required modifications.

BestSplit (Fig. 4) considers all possible tests $a > x$ in one iteration over the instances. It first sorts the instances by their value for $a$ from large to small. Each value $x$ in the middle of two subsequent values in this sorted list is a candidate split point (line 7). To compute the corresponding heuristic value, the algorithm needs to compute the variance in the two subsets induced by

**procedure** BestSplit($l$,$a$,$h_0$)
1: initialize $S_1$, $S_2$, $h^*$, $r^*$
2: initialize $v^\alpha$, $\mathrm{ML}_1^\beta, \mathrm{ML}_2^\beta, \mathrm{CL}_1^\beta$, and $\mathrm{CL}_2^\beta$
3: $a_{\mathrm{prev}} = \infty$
4: **for each** $i \in$ instances($l$) sorted by $a$ from large to small **do**
5: **if** $i[a] \neq a_{\mathrm{prev}} \wedge a_{\mathrm{prev}} \neq \infty$ **then**
6: $x = (i[a] + a_{\mathrm{prev}})/2$
7: $t =$ "$a > x$"
8: $v^\beta = \mathrm{AssignLabels}(\mathrm{ML}_1^\beta, \mathrm{ML}_2^\beta, \mathrm{CL}_1^\beta, \mathrm{CL}_2^\beta)$
9: $h = \mathrm{Heuristic}(h_0, S_1, S_2, v^\alpha + v^\beta)$
10: **if** $h < h^*$ **then**
11: $h^* = h$; $r^* =$ "refine $l$ using $t$"
12: $a_{\mathrm{prev}} = i[a]$
13: Update($S_1$, $i$, +1); Update($S_2$, $i$, -1)
14: **for each** $il \in IL^\alpha(i)$ **do**
15: Update($v^\alpha$, $il$)
16: **for each** $il \in IL^\beta(i)$ **do**
17: Update($\mathrm{ML}_1^\beta$, $\mathrm{CL}_1^\beta$, $il$, +1); Update($\mathrm{ML}_2^\beta$, $\mathrm{CL}_2^\beta$, $il$, -1)

**Fig. 4.** Selecting the split point for an attribute

the split on the instances. This can be done in constant time if appropriate statistics for the two subsets are available ($S_1$ and $S_2$ in the algorithm). Subset 1 contains the instances for which the test succeeds; subset 2 the instances for which it fails. $S_k$ summarizes the instances in subset $k$; it counts the number of instances, and for each attribute, the sum of its values and the sum of its squared values. Based on these numbers, the variances of the attributes can be computed ($\mathrm{Var}(a) = \overline{a^2} - (\overline{a})^2$). The algorithm starts with all instances in subset 2 (test $a > \infty$). $S_2$ is therefore initialized to represent all instances and $S_1$ is initialized to all zeros. Each iteration of the loop decreases the value of the split point $x$ (assuming no identical attribute values) and correspondingly moves one instance from subset 2 to subset 1. This is reflected in line 13, which adds the instance to $S_1$ and removes it from $S_2$. These updates simply correspond to adding (subtracting) the (squared) attribute values of the given instance to (from) the corresponding components of $S_1$ ($S_2$).

The first modification is required to be able to assign cluster labels for the two new leaves in $O(|C|)$ time with $C$ the set of cluster labels already in the tree. To this end, the algorithm uses the arrays $\mathrm{ML}_k^\beta$ and $\mathrm{CL}_k^\beta$. Similar to the $S_k$ statistics, there is one array of a given type for each of the two subsets. The arrays are used to count, for each cluster label, the number of ML and CL constraints in group $\beta$ (Fig. 3). For example, $\mathrm{ML}_k^\beta[c]$ counts the number of ML constraints that connect the instances in subset $k$ to one of the label $c$ leaves already in the tree. The number of group $\beta$ constraints $v_k^\beta(c)$ that are violated by assigning label $c$ to subset $k$ can now be computed as $v_k^\beta(c) = \mathrm{CL}_k^\beta[c] + \sum_{c_j \neq c} \mathrm{ML}_k^\beta[c_j]$; the number of constraints violated by assigning a new label is $v_k^\beta(new) = \sum_{c_j \in C} \mathrm{ML}_k^\beta[c_j]$

(all ML constraints violated). The former can be rewritten as $v_k^\beta(c) = \mathrm{CL}_k^\beta[c] + (v_k^\beta(new) - \mathrm{ML}_k^\beta[c])$. By first computing $v_k^\beta(new)$ and then computing for each $c \in C$, $v_k^\beta(c)$ using the second formula, all $v_k^\beta$ values can be computed in $O(|C|)$ time. The $v_k^\beta$ values are used to assign optimal labels (Section 3.3). Line 17 updates the $\mathrm{CL}_k^\beta$ arrays based on the group $\beta$ constraints in which instance $i$ participates. Such an update consists of retrieving the cluster label of the other instance participating in the constraint and updating the corresponding component of the ML or CL array (depending on the constraint's type). To make this step efficient, each instance stores its current cluster label and associated set of group $\beta$ constraints $IL^\beta(i)$.

To be able to compute the heuristic value BestSplit needs, in addition to the optimal labeling and its corresponding number of violated group $\beta$ constraints ($v^\beta$, line 8), also the number of violated group $\alpha$ constraints. It counts this number in the variable $v^\alpha$. Initially, all instances are in subset 2. As a result, all ML constraints of group $\alpha$ are satisfied and all CL constraints are violated. $v^\alpha$ is initialized to the latter. Each time an instance moves from subset 2 to subset 1, $v^\alpha$ is updated to take into account the group $\alpha$ constraints in which it participates. $v^\alpha$ is increased by one for each such constraint that becomes violated by moving the instance; it is decreased by one for each constraint that was violated before and becomes satisfied by moving the instance. Note that group $\alpha$ constraints change state (from violated to satisfied and the other way around) if one of their associated instances changes subset.

Based on the above statistics, BestSplit computes the heuristic value of a split $a > x$ and the optimal labeling of the corresponding new leaves (line 9). It uses $S_1$ and $S_2$ to compute the first term of the heuristic (the variance part) and the number of violated constraints $v^\alpha + v^\beta$ for the second term. Note that these statistics only account for the variance in the new leaves and the constraints associated to their instances. To compute the heuristic of the entire tree ($T$ with $l$ replaced by node($t$,$l_1$,$l_2$)), the algorithm adds the offset $h_0$. $h_0$ is computed as $H(T - \{l\}, I, IL - IL_l)$; $h_0$ takes the variance in the other leaves of $T$ and the violated constraints that do not have a participating instance in $l$ into account.

### 3.5 Algorithm Complexity

BestSplit($l$,$a$,$h_0$) sorts the instances, which takes $O(|I_l| \log |I_l|)$ time, with $I_l$ leaf $l$'s instances. Its main loop iterates over $I_l$. The main loop's most expensive steps are the call to AssignLabels, which takes $O(|C|)$ time ($C$ is the set of cluster labels), and updating the $S_k$, which takes $O(|A|)$ time ($A$ is the attribute set). BestSplit also processes the constraints in which the instances participate. Each such constraint is processed at most once ($\beta$) or twice ($\alpha$). As a result, the total cost of BestSplit is $O(|I_l| \log |I_l| + |I_l| \cdot (|C| + |A|) + |IL_l|)$, with $IL_l$ the constraints in which $l$ participates.

To iterate over all refinements of $T$, ClusILC calls BestSplit for each of $T$'s leaves and for each attribute in the data set. For a given attribute, the cost of calling BestSplit for all leaves is $O(|I| \log |I| + |I| \cdot (|C| + |A|) + |IL|)$ because each

instance occurs in at most one leaf and each constraint is included for at most two leaves. Each such iteration yields two additional nodes. As a result, the cost of building a tree with $N$ nodes is $O(N \cdot |A| \cdot (|I| \log |I| + |I| \cdot (|C| + |A|) + |IL|))$. This is more expensive than the TDI algorithm. The complexity of the latter is $O(D \cdot |A| \cdot (|I| \log |I| + |I| \cdot |A|))$, with $D$ the depth of the tree ($D < N$).

## 4 Experimental Evaluation

### 4.1 Setup

We present preliminary experiments with ClusILC, which has been implemented in the Clus system[1]. ClusILC has two parameters. The parameter $\gamma$ trades off the relative variance and the proportion of violated constraints in the heuristic. The parameter $m$ lower bounds the number of instances in each leaf. We set both parameters (ad-hoc) to their default values $\gamma = 0.5$ and $m = 2$.

We compare ClusILC to COP-$k$-means [3]. COP-$k$-means is a version of $k$-means that takes IL constraints into account. During each iteration, $k$-means assigns each instance in turn to its closest cluster. COP-$k$-means instead assigns each instance to the closest cluster center such that none of the constraints in which it participates are violated. If a given instance can't be assigned to one of the clusters without violating constraints, then COP-$k$-means fails (it performs hard constrained clustering).

The experimental setup is similar to that of Wagstaff et al. [3]. We use classification data sets from the UCI [13] repository as input (Table 1). The clustering algorithms only use the descriptive attributes. The class attribute is used to generate constraints. To generate a constraint, two instances are picked at random. If they belong to the same class, then the constraint becomes a ML constraint; if they belong to different classes then it becomes a CL constraint. We augment the IL constraints by adding all entailed constraints (during the initialization of the learners) [3]. The ML constraints represent an equivalence relation over the instances. We therefore add all constraints in the transitive closure of the original ML constraints. Assuming consistency, a CL constraint between two instances can be extended to their ML equivalence classes. That is, for each pair of ML equivalence classes $A$ and $B$ that are linked by at least one CL constraint, we add a CL constraint between every pair of instances $(a, b), a \in A, b \in B$.

The number of classes $k$ is given in each classification task. We use this number to set the parameter $k$ (number of clusters) of COP-$k$-means. We also introduce an upper bound on the number of cluster labels in ClusILC. This can be accomplished by changing the AssignLabels procedure so that it does not introduce a new cluster label if there are already $k$ labels in the tree.

We compare the result of the clustering algorithms to the correct labels in terms of the Rand index. Given two clusterings $C_1$ and $C_2$, $\text{Rand}(C_1, C_2) = \frac{a+b}{|I| \cdot (|I|-1)/2}$, with $a$ the number of instance pairs $(i_1, i_2)$ where $i_1$ and $i_2$ are in

[1] Available at `http://www.cs.kuleuven.be/~dtai/clus`.

**Table 1.** Data set properties: number of instances $|I|$, attributes $|A|$, and classes $k$

| | Name | $\|I\|$ | $\|A\|$ | $k$ |
|---|---|---|---|---|
| 1 | iris | 150 | 4 | 3 |
| 2 | hayes-roth | 160 | 4 | 4 |
| 3 | wine | 178 | 13 | 3 |
| 4 | glass | 214 | 9 | 6 |
| 5 | heart-statlog | 270 | 13 | 2 |
| 6 | ecoli | 336 | 7 | 8 |
| 7 | liver-disorders | 345 | 6 | 2 |
| 8 | ionosphere | 351 | 34 | 2 |
| 9 | balance | 625 | 4 | 3 |
| 10 | yeast | 1484 | 8 | 10 |
| 11 | image | 2310 | 19 | 7 |
| 12 | pendig | 7494 | 16 | 10 |

the same cluster in both $C_1$ and $C_2$, and $b$ the number of pairs where $i_1$ and $i_2$ are assigned to different clusters by both $C_1$ and $C_2$.

We report, besides the Rand index for all instances (i.e., cluster all instances and compute the Rand index for that clustering), also the cross-validated Rand index. The latter indicates how the algorithms perform on unconstrained instances. We use 10 fold cross-validation. In each iteration, constraints are generated for nine folds. The tenth fold is used to compute the Rand index. (The algorithms cluster the data in all folds.) The cross-validated Rand index is the average of the values computed for each of the folds. The results (both all data and cross-validated) are averages over 60 random sets of constraints[2].

### 4.2 Results

Fig. 5 presents the results. Consider first the curves for the clustering of all instances (labeled "All"). The Rand index of COP-$k$-means increases with the number of constraints and, in 8 out of 12 data sets, clearly surpasses that of ClusILC for a sufficiently large number of generated constraints. This was to be expected because COP-$k$-means only returns a clustering if it can satisfy all constraints: given enough constraints, the Rand index will become 1.0. (For the large data sets image and pendig COP-$k$-means requires many constraints.) ClusILC, on the other hand, may also return a solution that does not satisfy all constraints. This can happen either because, even if a solution exists, the greedy search fails to find it, or because the target concept cannot be expressed as a clustering tree with the given set of features. The result in both cases is a lower Rand index.

Next consider the cross-validation results (labeled "CV"). For these results, ClusILC does better for wine, ecoli, ionosphere, balance, image, and pendig (6 out of 12 data sets). It only does clearly worse for one data set (yeast). For the other 5 data sets it performs comparable to COP-$k$-means. One reason for the good generalization performance of ClusILC is that it can represent

[2] For wine, heart-statlog, liver-disorders, and ionosphere, COP-$k$-means was, for large constraints sets, unable to find a consistent clustering. To obtain results for these data sets, we generated in each trial different random constraint sets until it found a solution (up to $10^5$). The probability of finding a consistent solution strongly depends on the number of constraints. E.g., for wine, the peak was at 300 constraints and required on average 1087 sets before a consistent solution was found.

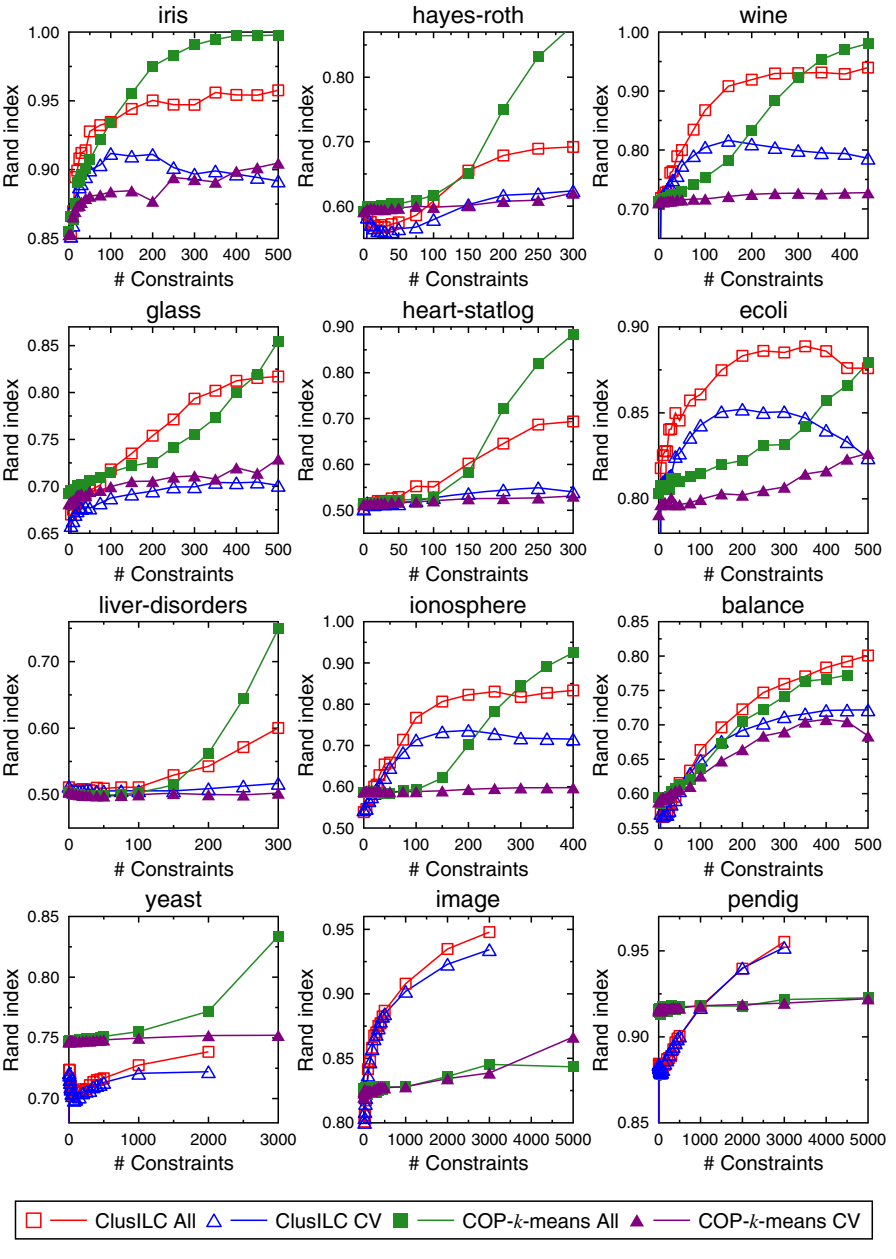


**Fig. 5.** Results for ClusILC and COP-$k$-means

more complex clusters than COP-$k$-means, which essentially assumes spherical clusters (i.e., a strong bias). Note also that not all constraints are useful and the possibility to ignore constraints can be beneficial [14].

We also measure the execution times and clustering tree sizes of ClusILC. The maximum execution time (over all sets of constraints) for one run ranges from 1.4 seconds (on balance) to 9 minutes (on heart-statlog). The maximum tree size ranges from 23 nodes (iris) to 773 nodes (yeast). Typically, the more constraints, the larger the tree. Note that this may yield to overfitting as can be seen, for example, from the graphs for iris and ecoli. The experiments were run on a cluster of AMD Opteron processors (1.8 - 2.4GHz, >2GB RAM) running Linux.

## 5 Conclusion and Further Work

Clustering trees are decision trees used for clustering tasks. The main advantage of such trees over other clustering methods is that they provide a symbolic description for each cluster. This paper shows how clustering trees can support instance level (IL) constraints. We extend clustering trees to be able to represent disjunctive descriptions by assigning cluster labels to the leaves. This modification is required to handle non-trivial constraint sets.

The main contribution is ClusILC, an algorithm that builds a disjunctive clustering tree given a set of instances and a set of IL constraints. ClusILC is a greedy algorithm guided by a global heuristic that takes the constraints into account. We discuss two important optimizations that are implemented in ClusILC: an algorithm for efficiently assigning cluster labels, and an algorithm for efficiently finding the optimal split point for a numeric attribute.

The experimental evaluation compares ClusILC to COP-$k$-means. While COP-$k$-means performs better on all data (its solution satisfies all constraints if it finds one), ClusILC has a better or comparable generalization performance.

We consider data sets with numeric attributes only. In future work, we plan to extend ClusILC to support data with mixed numeric and nominal attributes. The main modification that is required to this end is to redefine the variance metric used in the heuristic (e.g., to use the Gini index for nominal attributes [15]). ClusILC uses greedy hill-climbing search. We plan to investigate alternative search strategies, such as beam search, for which we have shown that it improves the performance of predictive clustering trees [16]. We also consider experiments comparing ClusILC to other constrained clustering algorithms, such as $k$-means algorithms that implement soft constrained clustering [4,5,6], metric learning approaches (e.g., [4,5]), and HAC [7,8]. Finally, we plan to investigate how other constraint types, such as constraints on the size of the tree [11], can be integrated in ClusILC.

**Acknowledgments.** Jan Struyf is a postdoctoral fellow of the Research Foundation - Flanders (FWO-Vlaanderen). The authors thank Siegfried Nijssen and Elisa Fromont for the fruitful discussion on clustering trees and IL constraints, and the anonymous reviewers for their constructive comments.

## References


1. Jain, A., Murty, M., Flynn, P.: Data clustering: A review. ACM Computing Surveys 31(3), 264–323 (1999)
2. Wagstaff, K., Cardie, C.: Clustering with instance-level constraints. In: 17th Int'l Conf. on Machine Learning, pp. 1103–1110 (2000)
3. Wagstaff, K., Cardie, C., Rogers, S., Schroedl, S.: Constrained K-means clustering with background knowledge. In: 18th Int'l Conf. on Machine Learning, pp. 577–584 (2001)
4. Bilenko, M., Basu, S., Mooney, R.: Integrating constraints and metric learning in semi-supervised clustering. In: 21st Int'l Conf. on Machine Learning, pp. 81–88 (2004)
5. Basu, S., Bilenko, M., Mooney, R.: A probabilistic framework for semi-supervised clustering. In: 10th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining, pp. 59–68. ACM Press, New York (2004)
6. Davidson, I., Ravi, S.: Clustering with constraints: Feasibility issues and the K-means algorithm. In: SIAM Int'l Data Mining Conf. (2005)
7. Davidson, I., Ravi, S.: Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. In: 9th European Conf. on Principles and Practice of Knowledge Discovery in Databases, pp. 59–70 (2005)
8. Klein, D., Kamvar, S., Manning, C.: From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In: 19th Int'l Conf. on Machine Learning, pp. 307–314 (2002)
9. Blockeel, H., De Raedt, L., Ramon, J.: Top-down induction of clustering trees. In: 15th Int'l Conf. on Machine Learning, pp. 55–63 (1998)
10. Michalski, R., Stepp, R.: Learning from observation: Conceptual clustering. In: Machine Learning: An Artificial Intelligence Approach, vol. 1, Tioga Publishing Company (1983)
11. Struyf, J., Džeroski, S.: Constraint based induction of multi-objective regression trees. In: 4th Int'l Workshop on Knowledge Discovery in Inductive Databases: Revised Selected and Invited Papers, pp. 222–233 (2006)
12. Quinlan, J.: C4.5: Programs for Machine Learning. Morgan Kaufmann Series in Machine Learning. Morgan Kaufmann, San Francisco (1993)
13. Merz, C., Murphy, P.: UCI repository of machine learning databases, University of California, Department of Information and Computer Science, Irvine, CA (1996), `http://www.ics.uci.edu/~mlearn/MLRepository.html`
14. Davidson, I., Wagstaff, K., Basu, S.: Measuring constraint-set utility for partitional clustering algorithms. In: 10th European Conf. on Principles and Practice of Knowledge Discovery in Databases, pp. 115–126 (2006)
15. Raileanu, L., Stoffel, K.: Theoretical comparison between the Gini index and information gain criteria. Annals of Mathematics and Artificial Intelligence 41(1), 77–93 (2004)
16. Kocev, D., Struyf, J., Džeroski, S.: Beam search induction and similarity constraints for predictive clustering trees. In: 5th Int'l Workshop on Knowledge Discovery in Inductive Databases: Revised Selected and Invited Papers (to appear, 2007)