CHAPTER 3
CONSTRAINED CLUSTERING ALGORITHMS

The high-level goal motivating this thesis is to develop a collection of clustering algorithms that can effectively incorporate background knowledge in the form of constraints. This chapter contains two major contributions. First, it develops a general method for transforming a generic clustering algorithm into an "intelligent" clustering algorithm. Second, it shows how this transformation can be effectively applied to two commonly used clustering algorithms and analyzes their behavior both formally and experimentally.

We begin by defining and describing instance-level hard constraints in Section 3.1.1. Section 3.1.2 presents our general method for creating "intelligent" clustering algorithms that can take advantage of domain knowledge expressed using those constraints. In addition, we show how the four example scenarios described in Chapter 1 can benefit from using a constrained clustering method (Section 3.2). Next, we describe the details of two such enhanced algorithms, COP-KMEANS (Section 3.3) and COP-COBWEB (Section 3.4), and provide a detailed formal analysis of each one. These algorithms are intelligent clustering versions of k-means and COBWEB respectively. Finally, we demonstrate how both modified algorithms can successfully make use of constraint information to improve their performance on artificial test problems (Section 3.5).

3.1 Intelligent Clustering

Our focus in this work is exclusively on partitioning algorithms, which divide the input items into a set of disjoint clusters. As described in Section 2.1, clustering algorithms partition a data set D into k non-overlapping groups (clusters). Intelligent clustering methods enforce an additional requirement that the output partition satisfy all specified constraints. This problem was formally defined in Section 2.2.

Before describing our general method for transforming a clustering algorithm to be able to accommodate domain knowledge, we will first formally define our encoding of that knowledge as pairwise constraints on the data set. Our goal is to construct constraints that express information about the underlying class structure in the data set, thereby enabling the algorithm to make more accurate choices about how to cluster instances.

3.1.1 Constraints

Every clustering algorithm has an objective function that guides its search through the space of possible solutions. This function evaluates candidate solutions and indicates how good (bad) each one is. The algorithm then attempts to maximize (minimize) the value of this function. When we present the algorithm with a set of constraints, we impose an additional requirement that the algorithm only consider solutions that fully satisfy the constraints.

Given the above description of clustering (partitioning), there is a natural way to express additional, domain-specific knowledge about the problem: we can use a set of instance-level pairwise constraints on the relative placements of items in the partition. We use two types of pairwise constraints to provide this kind of guidance:

- A **must-link** (inclusive) constraint specifies that two instances, d_i and d_j, have to appear in the same output cluster (though it does not identify which cluster). That is, $class(d_i) = class(d_j)$, where $class(d)$ represents the cluster that contains d. We will indicate that d_i **must-link** to d_j using the notation $d_i =_m d_j$. A set of must-link constraints defines an equivalence relation over $D \times D$.

- A **cannot-link** (exclusive) constraint specifies that two instances must *not* be placed in the same cluster. In other words, a partition that satisfies this constraint must have $class(d_i) \neq class(d_j)$. We will indicate that d_i **cannot-link** to d_j with $d_i \neq_c d_j$. The corresponding relation defined by a set of cannot-link constraints is symmetric but not transitive.

As noted in Chapter 2, there are several constraint types that could be considered useful for a clustering algorithm. These include global, cluster-level, feature-level, and instance-level constraints. We have argued that instance-level encodings are a useful general encoding, since they can support global and feature-level constraints as well. We will see examples of how feature-level constraints (Chapters 4 and 5) and global constraints (Chapter 6) arise in real problems.

Derived constraints (closure). Because the must-link constraints define an equivalence relation over the instances, we take a transitive closure over the constraints. This closure is defined as follows:

$\forall i, j, k$: given		produce
$d_i =_m d_j$	$d_j =_m d_k$	$d_i =_m d_k$
$d_i =_m d_j$	$d_j \neq_c d_k$	$d_i \neq_c d_k$
$d_i \neq_c d_j$	$d_j =_m d_k$	$d_i \neq_c d_k$

If $d_i \neq_c d_j$ and $d_j \neq_c d_k$, we cannot say anything about the relationship between d_i and d_k, since the cannot-link relation is not transitive. Computing the closure in this way allows us to extend the cannot-link relation from each item to its equivalence class. In addition, any inconsistencies or conflicts between the constraints will be detected at this point. As an example, consider the case where d_i must link *both* to d_j and to d_k, but d_j cannot link to d_k. The last constraint contradicts the implied must-link relationship between d_j and d_k. If no conflicts are found, the full set of derived constraints can then be presented to the clustering algorithm. If conflicts are detected, we can alert the user to the conflict and request a new set of conflict-free constraints.

Hard versus soft constraints. Note that both types of constraints are hard constraints, in that the clustering algorithm will not be allowed to violate any of them in the partition it produces. In our discussion of how to apply constrained clustering methods to real problems, we will see that good performance can often be obtained using hard constraints, but that in some cases we wish to encode domain knowledge that is heuristic or approximate. In such cases, hard constraints are inadequate; we require a way to express softer constraints on possible solutions. For this reason, we have also investigated the incorporation of soft constraints into clustering algorithms, which we will describe in Chapter 6. For the present discussion, however, we will focus on hard constraints.

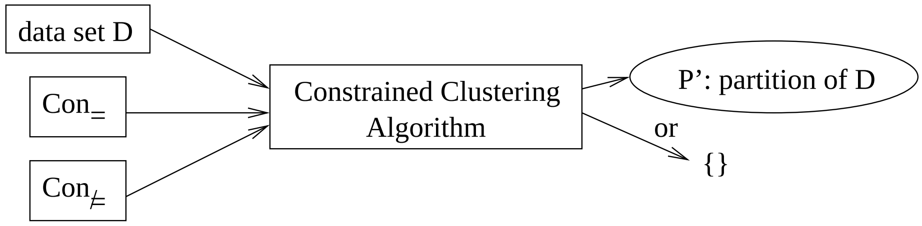


Figure 3.1: Constrained clustering architecture

3.1.2 Accommodating Constraints with Clustering

Given our definition of constraints, we can now describe how to modify clustering algorithms to accommodate them. The problem-specific constraints take precedence over the algorithm's built-in objective function, as in Tung et al. (2001). A generic partitioning algorithm takes as input a data set D and produces a partition P of the instances in D, optimizing an objective function f. In contrast, a constrained partitioning algorithm can additionally access a set of must-link constraints ($Con_=$) and a set of cannot-link constraints ($Con_{\neq}$). It will then return a partition P' of the instances in D that satisfies all specified constraints, or $\{\}$ if such a partition is not found. For unconstrained items, the constraint clustering algorithm follows the guidance the objective function f, as usual. This architecture is shown in Figure 3.1.

This description of constrained clustering is not specific to any particular clustering algorithm. Likewise, our method for transforming a clustering algorithm into a constrained clustering algorithm is not restricted to any single clustering algorithm. Instead, we present a general method for determining where and how a given clustering algorithm must be modified to allow it to incorporate instance-level constraints.

Recall that we are focusing on partitioning algorithms. Any such algorithm has at its heart a *cluster assignment* step, where data items are placed in their best host cluster. This may be part of an batch algorithm such as k-means (MacQueen, 1967), or it may occur in an incremental algorithm like COBWEB (Fisher, 1987). The cluster assignment step is where modifications can be made to enforce the domain knowledge that is provided to the algorithm as a set of constraints. Both must-link and cannot-link constraints, as defined above, can be accommodated by making changes to how the cluster assignment is done.

Incorporating Must-link Constraints

A must-link constraint is an immediate indicator that two items should be placed into the same cluster. The key modification for incorporating must-link constraints is to perform a check for any such guidance before creating the list of possible host clusters for $d \in D$. If there is a constraint $(d, d') \in Con_=$, and d' has already been assigned to cluster C_i, then d should also be assigned to C_i. Further, every other item that d must link to can also be immediately assigned to C_i.

Batch algorithms. As discussed in Section 2.1, batch algorithms differ from incremental algorithms in that they are used only when the entire data set available before clustering begins. In such a case, a further optimization is possible. The data set can be pre-processed,

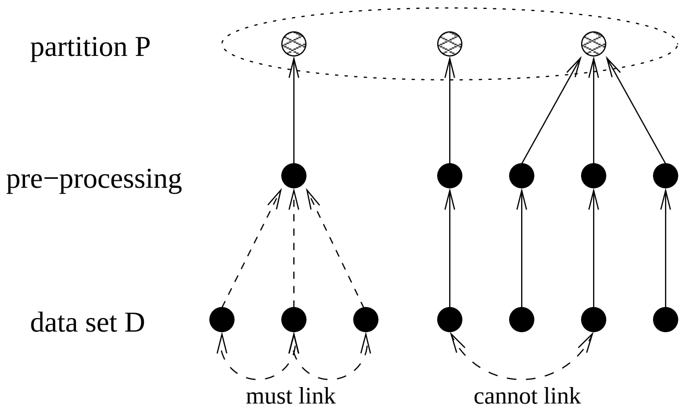


Figure 3.2: Constrained clustering process with a batch algorithm

in conjunction with $Con_=$, to replace each group of items that must be linked together with a single representative item, as shown in Figure 3.2. For numeric features in Euclidean space, the representative might be the mean or centroid of the group; for symbolic features, the representative might be a probability distribution of feature values. This representative is weighted by the number of items it replaces, and clustering then proceeds on the reduced data set. In a post-processing step, we copy the cluster assignment for the representative member to each of the original items it replaced.

Incremental algorithms. For incremental algorithms, the updates required when assigning $d \in D$ to a cluster may be slightly more complicated. In a truly online situation, we may not be able to compute the closure of the constraints before clustering starts. For example, before seeing item d, we have already processed some number of items and placed them into clusters. When we encounter d, we may discover that $d =_m d_1$ and $d =_m d_2$, where d_1 and d_2 have already been processed. This implies that $d_1 =_m d_2$, but since we did not know this when we assigned d_1 and d_2 to their clusters, they may be in different clusters. Thus, in online situations, satisfying must-link constraints may trigger further structural updates (by moving items around or merging clusters to accommodate the new information). Our use of incremental algorithms has been restricted to offline situations, where all constraints are known up front, so we have not had to explicitly address this issue. However, we make a note of it here as an interesting and useful extension of this work.

Incorporating Cannot-link Constraints

A constrained clustering algorithm must also ensure that it adheres to any specified cannot-link constraints when assigning items to clusters. Any assignment that violates a constraint must be abandoned in favor of a better choice. Therefore, the key modification to accommodate cannot-link constraints is to prune the list of possible host clusters to eliminate any solutions that would violate a constraint.

Batch versus incremental algorithms. When using must-link constraints, we described how batch algorithms have access to an additional pre-processing optimization. Although cannot-link constraints are useful for clustering algorithms, they do not provide a special optimization for batch algorithms. Figure 3.2 includes an example of a cannot-link constraint that has been preserved in the output partition; there are no pre-processing effects. However, as we will see, for a batch iterative algorithm such as k-means, cannot-link constraints can enable the selection of a better starting point in the search space.

Satisfiability. For algorithms where the number of clusters is fixed, it is possible for there to be no valid host cluster for a given instance, either because the algorithm made a bad choice early on, or because there truly is no solution that can satisfy all of the constraints. Handling this situation properly would then require a form of backtracking to achieve constraint satisfaction (or the determination that there is no possible satisfying solution). For other algorithms, where the number of clusters is not fixed, this situation might signal the need for the creation of a new cluster. In Section 3.5.3, we examine the satisfiability issue experimentally. For COP-KMEANS, a fixed-k constrained clustering algorithm, such "dead ends" manifest themselves most often for problems with an *intermediate* number of constraints. There are several ways the dead ends could be handled. We found that simply restarting with a different random cluster initialization was sufficient in practice to overcome this problem.

This section described a general method for transforming partitioning algorithms into constrained clustering algorithms. It also mentioned some of the interesting side issues involved in this transformation. We next proceed to show how real domain knowledge can be encoded using our constraint formulation.

3.2 Solving Concrete Problems

We now return to the four scenarios identified in Chapter 1 as situations that can benefit from the use of intelligent clustering methods. In general, converting domain knowledge into a useful representation is a non-trivial task. Knowledge representation is an active research area in artificial intelligence. In examining these examples, we will show how a variety of domain information can be encoded as hard, instance-level constraints for use by our methods.

3.2.1 Knowledge Not Expressed as Labels

Section 1.3.1 presented a data set that represents imaginary census data and indicated that there are kinds of information that the traditional feature-vector representation cannot encode. For example, this representation cannot incorporate the fact that two individuals are married to each other. We could easily indicate that an individual is *married* with a simple binary feature. The problem arises when we need to specify *to whom* a person is married. Adding a "spouse" feature which holds the spouse's name is not sufficient, because rather than making spouses appear to be more similar, it actually makes them appear to be more different. Clustering by similarity can then only produce groups of people who are married *to the same person*, not those who are married *to each other*. Alternatively, creating an additional (binary) feature for each item that corresponds to each potential spouse is

Table 3.1: The married relation

married	
Gary	Diana
Tim	Lyra
...	...

undesirable due to the explosion in dimensionality. Finally, we could generate a unique "married couple id" for each married pair and store that as a feature value. This increases the similarity of married pairs, but it does not guarantee that married pairs will be grouped together.

Domain knowledge: Pairwise relation between data items. The domain knowledge we wish to encode for this problem is a *pairwise relation between data items.* Table 3.1 augments the census data found in Table 1.1 with an additional source of information: we have defined a simple binary relation, married, on the items in the data set that indicates their marital relationships. This representation is compact and precise and can be directly incorporated by a constrained clustering algorithm.

We can convert this relation into a set of hard, instance-level constraints by creating a must-link constraint for each married pair. The set of constraints presented to the intelligent clustering algorithm would then be $\{$ Gary $=_m$ Diana, Tim $=_m$ Lyra, $\ldots\}$. When enforced, these constraints will guarantee that a married pair is never split across two clusters. More generally, for a pairwise relation $R \subseteq D \times D$ that encodes relationships that we wish to preserve in the clustering output, we simply create, for each relationship $(d_i, d_j) \in R$, a corresponding constraint $d_i =_m d_j$:

$$(d_i, d_j) \in R \Rightarrow (d_i, d_j) \in Con_=.$$

3.2.2 Partially Labeled Data Sets

As described in Section 1.3.2, some data sets possess labels for some, but not all, of their items. Traditional supervised algorithms would only be able to learn from the labeled data, and traditional unsupervised algorithms would have to ignore the labels that are present or have a facility for handling unknown values (for the items without labels). Further, unsupervised methods cannot provide a guarantee that the labels will be preserved in the final output cluster. In general, we may have a label L_i associated with data item d_i. Our goal is then to encode a *partial set of cluster labels* using our constraint formulation.

Domain knowledge: Partial set of cluster labels. When labels are present that represent the desired cluster structure, we would like the clustering algorithm to conform to them. We can achieve this by converting those labels into a set of instance-level constraints. For each pair $d_i, d_j \in D$, if L_i and L_j exist and $L_i = L_j$, then we create a constraint $d_i =_m d_j$.

Table 3.2: Partially labeled census data

Name	SSN	...	Label
Diana	111-11-1111	...	A
Gary	444-44-4444	...	B
Lyra	222-22-2222	...	?
Tim	777-77-7777	...	B
...	...	...	...

If $L_i \neq L_j$, we create $d_i \neq_c d_j$.

$$\begin{aligned} L_i = L_j &\Rightarrow (d_i, d_j) \in Con_= \\ L_i \neq L_j &\Rightarrow (d_i, d_j) \in Con_{\neq} \end{aligned}$$

Let us assume that Table 1.1 has been augmented with a partial set of labels, as shown in Table 3.2. Some individuals have been labeled with either an "A" or a "B". Others do not possess labels, like Lyra. For this example, we obtain the following constraints: { Diana $\neq_c$ Gary, Diana $\neq_c$ Tim, Gary $=_m$ Tim, ...}. Because Lyra's classification is unknown, she will not participate in any constraints. The constraints will ensure, however, that the known labels are preserved in the output partition.

3.2.3 Intelligent Exploratory Data Analysis

Section 1.3.3 describes a situation where clustering is used as an exploratory tool. We may wish to cluster the same census data in an attempt to discover interesting groups of individuals, without any specific notion of what the correct result should be. Regular clustering algorithms are well suited to this task. However, if we have any information about the desired result, even if it is not in the form of individual item labels, we would like to provide that information to a constrained clustering algorithm.

Domain knowledge: Feature-level consistency. In this case, the domain knowledge we are interested in encoding is a feature-level heuristic that contains information about the desired relative placement of items. We would like the resulting clusters to be consistent with the gender feature, so that individuals of different genders end up in different clusters. This does not necessarily mean that we wish to identify two final clusters, one composed solely of men and one composed of women. That division of the census data could be achieved without any clustering at all (assuming that gender is known for each person in the data set). Instead, we would like to find "useful" or "interesting" groups inside the data set that happen to preserve gender distinctions. Thus, we might end up with groups such as "female engineers" or "male dancers."

If a match on the feature value contributes to the similarity of the two items, we formulate a rule for generating constraints:

$$d_i.\text{feature} = d_j.\text{feature} \Rightarrow (d_i, d_j) \in Con_=.$$

If a mismatch on the feature value is strong enough evidence to signal that the two items should not be grouped together, we have

$$d_i.\text{feature} \neq d_j.\text{feature} \Rightarrow (d_i, d_j) \in Con_{\neq}.$$

Our task is to convert this additional information based on the gender feature into a set of instance-level constraints. In this case, we have

$$d_i.\text{gender} \neq d_j.\text{gender} \Rightarrow (d_i, d_j) \in Con_{\neq}.$$

By examining each person's value for the gender feature, we construct the following constraints: { Gary $\neq_c$ Diana, Gary $\neq_c$ Lyra, Tim $\neq_c$ Diana, Tim $\neq_c$ Lyra, ...}. These constraints will ensure that the resulting clusters are consistent with the gender feature.

3.2.4 Structured Data Sets

Finally, we return to the final scenario presented in Chapter 1. In Section 1.3.4, we discussed the problem of image segmentation, where the goal is to divide the pixels in an image into homogeneous regions. In addition to wanting all of the pixels in a cluster to have similar color or intensity values, we would also like neighboring pixels to be more likely to be assigned to the same cluster. Our formulation of constraints allows us to do just that. In this case, hard constraints are not appropriate. We do not want to require that each pair of adjacent pixels be placed into the same cluster; the only valid solution to that constraint is a single cluster that contains every pixel in the image. Instead, we would like to specify a preference, which is best encoded as a soft constraint. We will defer a more in-depth discussion of soft constraints and how image segmentation can benefit from their use to Chapter 6.

We have discussed several examples of where a constrained clustering algorithm would be useful and indicated how to encode three types of domain knowledge as a set of hard, instance-level constraints. The fourth kind of domain knowledge is best encoded as a set of soft, instance-level constraints. We now proceed to describe two constrained clustering algorithms we have developed. To create these algorithms, we start with two commonly used clustering algorithms, k-means and COBWEB, and apply the transformation outlined in Section 3.1.2.

3.3 K-means Clustering

K-means clustering is a method commonly used to automatically partition a data set into k groups (MacQueen, 1967). Most implementations begin by selecting k initial cluster centers and then iteratively refining them as follows:

1. Each instance d_i is assigned to the cluster C that will minimize total *variance* (sometimes referred to as *dispersion*).

2. Each cluster center C_j is updated to be the mean of its constituent instances.

The algorithm stops when there is no further change in assignment of instances to clusters. It is also possible to update the cluster centers after each item is assigned (Theiler and Gisler, 1997), although doing so makes the outcome of the algorithm dependent on the order in which the items are processed. The k-means algorithm is an iterative, hill-climbing algorithm that converges to a local optimum (Selim and Ismail, 1984). Krishna and Murty (1999) developed a hybrid algorithm, based on k-means and a genetic algorithm, which instead converges to the global optimum; this algorithm is much more computationally expensive.

K-means generally seeks to minimize the total squared variance associated with the output partition:

$$var(C_1 \ldots C_k) = \sum_{d \in D} dist(d, Ct_{d.cluster})^2 \tag{3.1}$$

where $Ct_{d.cluster}$ is the center of the cluster to which d is assigned. Each cluster C_j is represented by its center Ct_j, which is defined as the center of gravity of the points assigned to C_j. Equation 3.1 measures the "spread" of the data by examining how far each item is from its respective cluster center. A related objective function is the *sum-of-squares* criterion:

$$ssq(C_1 \ldots C_k) = \sum_{C_j} \sum_{d \in C_j} \sum_{d' \in C_j} dist(d, d')^2. \tag{3.2}$$

Both optimization problems have the same optimal solution, and both have been shown to be NP-hard (Garey and Johnson, 1979).

3.3.1 The COP-KMEANS Algorithm

Following the general procedure described above, we can modify the k-means algorithm to accommodate instance-level hard constraints as shown in Table 3.3 (Wagstaff et al., 2001). Changes from the original k-means algorithm are shown in bold. The algorithm takes in a data set D, a set of must-link constraints $Con_=$, and a set of cannot-link constraints $Con_{\neq}$. It returns a partition of D that satisfies all specified constraints, or the empty partition if one cannot be found.

Must-link optimization. Because k-means is a batch algorithm, it can make use of the must-link constraint optimization. The constraints in $Con_=$ are satisfied in a pre-processing step: each group of instances that must be internally linked is replaced by a single instance that represents all items in the group. This new item is weighted by the number of items that it represents. Then, when updating cluster assignments, our changes ensure that none of the specified cannot-link constraints are violated.

This reduction in the data set size also triggers an update of the cannot-link constraints. The algorithm must ensure that $Con_{\neq}$ is consistent with the reduced data set. A must-link group $D_m \subseteq D$ requires that for each pair of items $d_{m1}, d_{m2} \in D_m$, there is a must-link constraint between d_{m1} and d_{m2}. The entire must-link group is replaced by a single data item, d_m, and any cannot-link constraints that any of its members participated in will now apply to d_m itself. That is,

$$(d_{mi}, d') \in Con_{\neq} \Rightarrow Con_{\neq} := (Con_{\neq} \setminus (d_{mi}, d')) \bigcup (d_m, d').$$

Table 3.3: COP-KMEANS algorithm

COP-KMEANS(number of clusters k, data set D, must-link constraints $Con_= \subseteq D \times D$, cannot-link constraints $Con_{\neq} \subseteq D \times D$)

1. **Must-link optimization:** Identify each group of items that must be linked together. Replace each such group with a single item that is the mean of the items in the group, weighted by the number of items it represents. Update $Con_{\neq}$ to be consistent with the reduced data set.

2. Let $Ct_1 \dots Ct_k$ be the initial cluster centers.

3. For each instance d in D, assign it to the cluster C that will minimize total variance, **such that** VIOLATE-CONSTRAINTS**(d, C, $Con_{\neq}$) is false. If no such cluster exists, halt (return $\{\}$)**.

4. Update each cluster center Ct_i by averaging all of the points $d_j \in C_i$ that have been assigned to it.

5. Iterate between (3) and (4) until convergence.

6. Return the partition $\{C_1, \dots, C_k\}$.

VIOLATE-CONSTRAINTS(data point d, cluster C, cannot-link constraints $Con_{\neq} \subseteq D \times D$)

1. For each $(d, d_{\neq}) \in Con_{\neq}$: If $d_{\neq} \in C$, return true.

2. Otherwise, return false.

Item assignment. Like k-means, COP-KMEANS attempts to assign each point d to the cluster that would minimize total variance, as calculated by Equation 3.1. The cluster assignment will succeed unless a cannot-link constraint would be violated. If there is another point $d_{\neq}$ that cannot be grouped with d but is already in C, then d cannot be placed in C, regardless of the variance calculation. COP-KMEANS continues down the sorted list of clusters until it finds one that can legally host d. Constraints are never broken; if a legal cluster cannot be found for d, the empty partition $\{\}$ is returned. After convergence, each instance in a must-link group receives the same cluster assignment that its representative received.

Note that COP-KMEANS preserves the invariant that all specified constraints must be satisfied at all times, i.e., at each step of the iteration. Thus, intermediate solutions will always be "valid" with respect to the constraints. This allows the user to potentially make use of an intermediate solution (before convergence) if desired.

Cannot-link optimization. A further improvement is possible. The quality of the clusters that the original k-means algorithm creates is known to be very sensitive to the choice of initial cluster centers (Bradley and Fayyad, 1998). The usual way to select the initial centers, in the absence of any other information, is to choose them randomly from the data set. A common problem with this approach is that it can select two items as cluster centers which would be best placed in the *same* cluster, which means that the k-means algorithm will require a large number of iterations to arrive at the correct solution (and may never reach it, since it converges to a local optimum).

However, if k-means has access to constraints in $Con_{\neq}$, it can select better initial centers. Finding a satisfying solution to the specified constraints effectively requires finding a k'-clique in $Con_{\neq}$, for the largest $k' \leq k$. (Note that if a k'-clique exists in $Con_{\neq}$ for some $k' > k$, then there is no partition of D that satisfies the constraints.) Even if a k'-clique is not known at the outset, COP-KMEANS can still select better initial centers by choosing any two items that cannot link together as the first two cluster centers, and then randomly selecting the remaining $k - 2$ cluster centers.

3.3.2 Formal Analysis of COP-KMEANS

The formal properties of the k-means algorithm have been well studied. In this section, we examine how our changes to the algorithm affect its runtime and convergence properties. We find that asymptotic runtime increases for a single step of the iterative algorithm to accommodate constraints, but that empirically the constraints actually reduce runtime. The convergence guarantees of the algorithm are also affected, but this does not present a problem in practice.

Runtime. Our first concern is the runtime of the algorithm. The basic k-means algorithm first selects k points as the initial cluster centers; this requires $O(k)$ time. The algorithm then iterates between two steps. A single iteration proceeds as follows. The first step assigns each point to the cluster that will minimize overall variance. For each point, we must examine its distance to each possible host cluster. The distance calculation scales with f, the number of features in the data set. The total runtime for this step is $O(nkf)$. Pelleg and Moore (1999) have shown how the k factor can be further reduced, for data sets with low dimensionality ($f < 8$), using kd-trees. The second step updates each cluster center to be the mean of its constituents. Since each point is only assigned to one cluster, this update can be done in $O(nf)$ time. The total cost of a single iteration, then, is $O(nkf)$.

With our changes incorporated, the algorithm's runtime changes. Let m and c be the number of must-link and cannot-link constraints, respectively. These constraints are enforced in the data assignment step of the algorithm. The assignment of each item must be consistent with the constraints that involve that item; since the constraints are binary, each constraint will be examined at most twice. This will increase the runtime of the assignment step, originally $O(nkf)$, by $O(nkf + m + c)$. The cluster update step remains $O(nf)$, so the runtime for a single iteration becomes $O(nkf+m+c)$. The quantity $m+c$ is bounded above by $\frac{1}{2}n(n-1)$, so the iteration cost is equivalent to $O(nkf + n^2)$.

If the must-link pre-processing optimization is used, the cost of a single iteration is reduced. The pre-processing can be done in $O(mf + c)$ time, and the iteration cost becomes $O(nkf + c)$. Asymptotically, this is still $O(nkf + n^2)$. In reality, the pre-processing step shrinks the data set size (reducing n) and c is usually much smaller than n^2; in addition, c may be greatly reduced during the must-link pre-processing step.

Empirical runtime effects. Although the asymptotic complexity of the constrained clustering version of k-means is worse than that of the unconstrained version, empirically we find that both kinds of constraints can contribute to reducing the total runtime. Must-link constraints can be pre-processed to reduce n, the number of items which must be clustered. More precisely, n is reduced by m', where m' is the size of the *transitive reduction* of $Con_=$. The transitive reduction of $Con_=$ is the smallest set of pairwise relationships that transitively imply every relationship in $Con_=$. In addition, each cannot-link constraint may, for a given item d, also exclude a specific host cluster from consideration. This reduces the number of distance calculations that must be done to assign d to a cluster. In practice, we observe significant reductions in the number of iterations required to reach convergence. In our experiments with artificially-generated constraints (Section 3.5), we found that the mean number of iterations was roughly *halved* with the introduction of 100 random constraints for data sets of approximately 50 elements.

Convergence properties. The k-means algorithm is known to converge to a local optimum in terms of variance and to do so using gradient descent (Bottou and Bengio, 1995). More precisely, Bottou and Bengio showed that the k-means algorithm minimizes variance using Newton's method and that it therefore has the same (superlinear) convergence speed as Newton's method.

Our modifications to the algorithm impact its behavior in important ways. Since the final solution (and each intermediate solution) must satisfy all specified constraints, the constrained clustering algorithm effectively must also perform constraint satisfaction. Deciding whether a satisfying solution exists for a given set of constraints is itself NP-complete (Garey and Johnson, 1979). The current implementation of this algorithm does not attempt to find a fully satisfying solution. Instead, it does a one-step lookahead greedy assignment of points during the data assignment phase. This means that, even for a set of satisfiable constraints, a bad decision early on may result in a "dead end" where the algorithm has no valid cluster in which to place a given point. In such a case, we restart the algorithm with a different random initialization of cluster centers.

This is somewhat unsatisfying, since the original k-means algorithm can guarantee that it will return a partition that is a local minimum with respect to variance; that property is lost in this formulation of COP-KMEANS. In practice, there is little impact on the results we obtain. Our algorithm terminates immediately if this situation is encountered, and we can then restart with a different set of random initial cluster centers. This is standard experimental procedure for k-means already; since k-means is known to be very sensitive to the initial starting points it selects, most experimentalists run the algorithm several times on the same data set and retain the best result.

A possible improvement would be to retain, at each step, the partition produced by the previous iteration. Then, if the algorithm fails to find a satisfying solution in the current iteration, it can return the previous partition. In some cases, this may be better than simply restarting the calculation. However, this partition is not a "converged" result (otherwise the algorithm would have terminated with it), so it will not even be a locally optimal result. Consequently, we have not implemented this change for our experiments.

Another way to approach the problem, which is somewhat easier to analyze and has better convergence properties, is to start the algorithm with a valid solution (assignment of points to clusters) that satisfies all of the specified constraints but may have an arbitrarily large variance. Then, when updating assignments, we can disallow any changes in assignment that would violate a constraint. COP-KMEANS will then find the best solution, in terms of minimizing variance, inside the connected[1] space of solutions that includes the specified starting point. Of course, finding a valid starting point is a difficult problem. If the quantity $m + c$ is not too large, this is a reasonable approach to adopt—and it retains the attractive k-means guarantee of finding a local minimum.

3.4 COBWEB Clustering

As a second concrete example of how a clustering algorithm can be enhanced to make use of constraint information, we present COP-COBWEB, a constrained partitioning version of the COBWEB algorithm (Fisher, 1987).

COBWEB is an incremental clustering algorithm that employs the concept of category utility (CU) (Gluck and Corter, 1985) as its objective function. Clusters are not represented by the mean of their elements; instead, each cluster is represented as a set of probability distributions, one per feature of the data set. Probability is distributed over the possible values for a given feature. The category utility of a partition is measured by

$$CU(C_1, \ldots, C_k) = \frac{1}{k}((\sum_{c=1}^{k} \frac{|C_c|}{n} \sum_f \sum_{j \in V(f)} P(F_f = j | C_c)^2) - \sum_f \sum_{j \in V(f)} P(F_f = j)^2) \tag{3.3}$$

[1]Here "connected" means that each partition in the solution space is reachable from every other partition through a series of *valid* assignments (i.e., assignments that do not violate any constraints).

where k is the number of categories or clusters, C_c is a particular cluster, F_f refers to one of the features, and $V(f)$ is the set of discrete values for feature F_f. The values are discrete because COBWEB was designed for use with categorical, or symbolic, attributes. COBWEB creates a hierarchy of clusters, with more general clusters towards the top of the hierarchy and more specific clusters towards the bottom. It is not necessarily a *full* hierarchy, with a single cluster containing all of the instances at the top and n clusters at the bottom.

COBWEB considers four primary operators (add, new, merge, and split) that represent the possible ways to incorporate a new instance into the top level of the existing hierarchy. It applies each operator and selects the one that maximizes the category utility of the resulting hierarchy. COBWEB continues recursively by applying the same operators to that cluster's children to properly sort the new instance with respect to deeper levels, halting when the instance is placed in a leaf node.

Our focus in this work is on partitioning algorithms. Consequently, we will be using a partitioning version of COBWEB. In the absence of constraints, this version produces output that corresponds to the top level of the hierarchy produced by the regular version of COBWEB.

3.4.1 The COP-COBWEB Algorithm

Once again, we can use our general procedure for modifying clustering algorithms to update the COBWEB partitioning algorithm to accommodate instance-level hard constraints as shown in Table 3.4. In particular, as each instance d in the data set is encountered, we first check for any must-link constraints (step 2a). If there is a must-link constraint that indicates that d must be in the same cluster as some d_j that has already been incorporated into the partition, we enforce the constraint by including d in the cluster C that contains d_j. If not, we consider applying each of the new (step 2b), add (step 2c), and merge (step 2d) operators to determine where to place d.

Cannot-link constraints are checked during the add and merge cluster assignment steps. More specifically, when considering adding an instance to an existing cluster C_j, we check for the existence of any cannot-link constraints that would prevent d from joining C_j (VIOLATE-CONSTRAINTS is defined in Table 3.3). When considering merging two clusters, we once again must check for any cannot-link constraints that would invalidate the merge (for example, $(d_1, d_2) \in Con_{\neq}$ where $d_1 \in C_{max1}$ and $d_2 \in C_{max2}$). Whether or not there was a must-link constraint found in step 2a, we consider an application of the split operator (step 2e), which recurses on the subset of the instances contained in the best host cluster for d. Finally, the choice that results in the highest CU is selected as the new partition P (step 2f).

Note that, unlike COP-KMEANS, COP-COBWEB cannot fail to find a satisfying solution. This is due to the fact that COP-COBWEB is allowed to select an appropriate value for k, the number of clusters, while COP-KMEANS must enforce the additional constraint of a specific value for k.

3.4.2 Formal Analysis of COP-COBWEB

The formal properties of the COBWEB algorithm have not been closely examined. COBWEB is not an iterative algorithm like k-means, so we are not concerned with convergence.

Table 3.4: COP-COBWEB algorithm

COP-COBWEB(data set D, must-link constraints $Con_= \subseteq D \times D$, cannot-link constraints $Con_{\neq} \subseteq D \times D$)

1. Let P be the set of clusters, initially $\{\}$.
2. For each instance d in D:
   (a) **Must-link check:** If there exists some $(d, d_j) \in Con_=$ such that d_j is already in an existing cluster $C \in P$, then add d to C and store the new partition in P_{must}. Skip to step (e).
   (b) New: Let $P_{new} = P \bigcup \{C\}$ where $C = \{d\}$ is a new cluster.
   (c) Add: For each existing cluster C_j in P, create a new partition P_{add_j} which has d added to C_j, **unless** VIOLATE-CONSTRAINTS**(d, C_j, $Con_{\neq}$) is true**.
   (d) Merge: If there exist at least two clusters, let C_{max1} and C_{max2} be the two best hosts for d from step (c) as determined by the CU values of their resulting partitions. Merge those two classes, add d, and store the new partition in P_{merge}, **unless this would violate a cannot-link constraint**.
   (e) Split: If there exist at least two clusters, let C_{max} be the best host for d from step (a) or (c) as determined by the CU values. Split C_{max} by recursing on it; that is, replace it with COP-COBWEB($C_{max} \bigcup \{d\}, Con_=, Con_{\neq}$).
   (f) Let $m = argmax\ CU(P_k)$ for $k \in \{must, new, add_j, merge, split\}$. Update $P = P_m$.
3. Return P.

However, runtime remains very important. In this section, we develop an understanding of how COP-COBWEB behaves without constraints and then describe the impact that adding constraints has. We find that, empirically, constraints for this algorithm provide significant runtime benefits.

Termination. The basic COBWEB algorithm is recursive in nature. Recursion occurs when the split operator is applied. Termination is guaranteed because the split operator recurses on a strict subset of the current data set, and split is only considered when there is more than one cluster present in that subset (i.e., non-leaf nodes, in the original hierarchical version). The modifications in COP-COBWEB do not affect termination.

Runtime. For this algorithm, we are interested in the cost of processing each item, d. For each such item, there are four possible ways to incorporate it into the existing partition, and each of these possibilities must be examined by calculating the category utility (CU) of the partition. The CU calculation requires $O(\sum_i |V(i)|)$ time, where $V(i)$ is the set of possible values for feature i. Creation of a new cluster to contain d requires the initialization of the cluster's values for each of the features, so $T_{new} = f + \sum_i |V(i)|$. Adding d to each of the k_d existing clusters requires an update to each of the f features for each such cluster, so $T_{add} = k_d(f + \sum_i |V(i)|)$. Merging two clusters and including d requires the update of each of d's feature values in the cluster representation, but it also (in the worst case) requires an update to every possible feature value (due to the items present in the two clusters being merged). Therefore, $T_{merge} = \sum_i |V(i)| + \sum_i |V(i)| = 2\sum_i |V(i)|$. Performing a split requires a recursive call on one of the k_d clusters (C_{max}) and could theoretically incur the cost of re-processing each item in C_{max}. Thus, $T_{split} = R(|C_{max}|) + \sum_i |V(i)|$. In practice, we cache this information so that it is not re-calculated for subsequent splits, if performed on the same cluster.

The total time $R(n)$ required for the incorporation of n items into the partition is the sum of the time required for each item d_i, $T(d_i)$:

$$R(n) = \sum_{i=1}^{n} T(d_i) \tag{3.4}$$

$$T(d_i) \leq T_{new} + T_{add} + T_{merge} + T_{split}(C_{max}) \tag{3.5}$$

Merging and splitting are only considered if there are more than two existent clusters, i.e., $k_d \geq 2$. However, we are concerned only with worst-case runtime. After substituting in the individual costs of each operation, we obtain:

$$T(d_i) \leq (k_d + 1)f + (k_d + 4)(\sum_i |V(i)|) + R(|C_{max}|) \tag{3.6}$$

COP-COBWEB introduces changes that impact COBWEB's runtime. Let m_d and c_d be (respectively) the number of must-link and cannot-link constraints in which d participates. Before the four operators are considered, COP-COBWEB examines the must-link constraints. The cost of some of the individual operations also changes. The new step is unchanged. The add step, however, checks $Con_{\neq}$ to prevent any invalid additions. Likewise, the merge step must check whether merging the two clusters would violate any constraints in $Con_{\neq}$. It must check constraints that involve any items in the two clusters, not just c_d. The maximum number of constraints examined is $|C_{max1}| \cdot |C_{max2}|$. Accordingly, we update $T_(d_i)$ as follows:

$$T(d_i) \leq m_d + T_{new} + (c_d + T_{add}) + (|C_{max1}| \cdot |C_{max2}| + T_{merge}) + T_{split}(C_{max}) \qquad (3.7)$$

To summarize, adding the ability to incorporate user-specified constraints into the clustering process does have an impact on the algorithm's runtime. The algorithm must check, for each instance, all of the relevant constraints to ensure that they will not be violated. The asymptotic runtime therefore increases. However, we find in practice that runtime actually *decreases* significantly when using constraints, since many of the possible operations are excluded from consideration.

Empirical runtime effects. Once again, both kinds of constraints can contribute to reducing the total runtime. Must-link constraints allow COP-COBWEB to skip the new, add, and merge steps, since they dictate immediately which cluster the item should join. Cannot-link constraints exclude certain host clusters from consideration during the add step. For example, in experiments with artificially generated constraints, we find that runtime increases as the number of constraints increases, until an inflection point is reached. Beyond this point (which varies for each data set), additional constraints actually *decrease* COP-COBWEB's runtime. In our experiments on a real problem from Natural Language Processing, in Chapter 5, we observe runtimes as low as 0.14 (reduction of 86%) of the original (unconstrained) runtime.

3.5 Experiments with Artificial Constraints

In this section, we report on experiments with COP-KMEANS and COP-COBWEB using six data sets in conjunction with artificially-generated constraints. Five are from the UCI repository (Blake and Merz, 1998), and one (part-of-speech) comes from the domain of natural language processing. Each graph in this section tracks the change in accuracy as more constraints are made available to the algorithm. COBWEB is an incremental algorithm, while k-means is a batch algorithm. Despite their significant algorithmic differences, we found that both algorithms improved almost identically when supplied with the same amount of background information.

Methodology. The number of clusters in each of the six data sets is known. For experiments with COP-KMEANS, we provided the known k value as input. We initialized the clusters using instances chosen at random from the data set, with two caveats. First, we used the cannot-link optimization described above which allows us to select a pair of instances connected by a cannot-link constraint, when possible. Second, we ensure that no two randomly selected cluster centers are connected by a must-link constraint.

The data sets we used are composed solely of either numeric features or symbolic features. For numeric features, we used a Euclidean distance metric; for symbolic features, we computed the Hamming distance. Experiments with COP-COBWEB were restricted to data represented by symbolic features. No distance metric was needed, since COBWEB uses category utility as its objective function.

We generated the artificial constraints in the following way. For each desired constraint, we randomly picked two instances d, d' from the data set and checked their labels (which are available for evaluation purposes but are not visible to the clustering algorithm). If they had the same label, we generated a must-link constraint. Otherwise, we generated a cannot-link constraint.

$$\begin{aligned} d.label = d'.label &\Rightarrow (d, d') \in Con_{=} \\ d.label \neq d'.label &\Rightarrow (d, d') \in Con_{\neq} \end{aligned}$$

This process ensures that our artificially-generated constraints are fully consistent with the labels used for evaluation, and are therefore a good simulation of reliable domain knowledge.

3.5.1 Evaluation Method

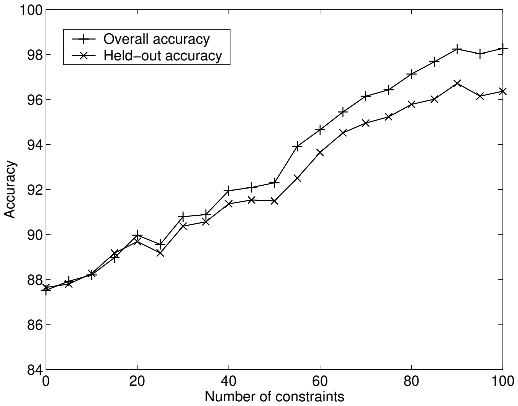
For evaluation, we compared the output partitions produced by COP-KMEANS and COP-COBWEB with the provided data labels. To calculate agreement between our results and the correct labels, we used the Rand index (Rand, 1971). This metric is commonly used to compute the similarity of two partitions, P_1 and P_2, of the same data set D. Each partition is viewed as a collection of $n(n-1)/2$ pairwise decisions, where n is the number of items in D. For each pair of points d_i and d_j in D, P_i either assigns them to the same cluster or to different clusters. Let a be the number of decisions where d_i is in the same cluster as d_j in both P_1 and in P_2. Let b be the number of decisions where the two instances are placed in separate clusters by both partitions. Total agreement can then be calculated using the following equation:

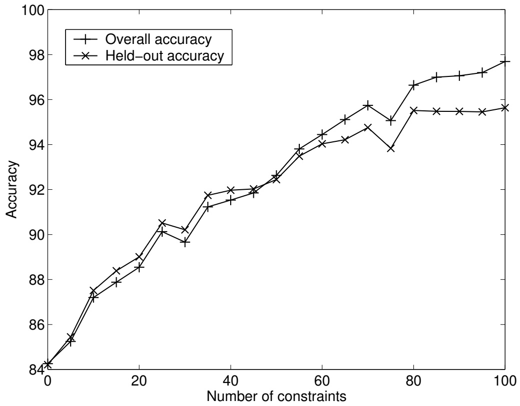
$$Rand(P_1, P_2) = \frac{a+b}{n(n-1)/2}. \tag{3.8}$$

We were also interested in testing our hypothesis that constraint information can boost performance even on unconstrained instances. Consequently, for experiments with artificial constraints, we present two sets of numbers: the overall accuracy for the entire data set, and accuracy on a held-out test set (a subset of the data set composed of instances that are not directly or transitively affected by the constraints). This is achieved via ten-fold cross-validation; we generate constraints on nine folds, cluster all ten folds together, and then calculate performance on the tenth fold. This enables a true measurement of improvements in learning: any improvements on the held-out test set indicate that the algorithm was able to generalize the constraint information to improve performance on the unconstrained instances as well.

3.5.2 Results using Artificial Constraints

In these experiments, we tested the behavior of both constrained clustering algorithms when presented with varying numbers of constraints. We here present results with six different data sets. We find consistent trends for all data sets: accuracy increases as the number of available constraints increases. We conclude that both algorithms are able to successfully incorporate constraints into the clustering process.





(a) COP-KMEANS (b) COP-COBWEB

Figure 3.3: Clustering accuracy with artificially generated constraints: the soybean data set (47 instances in four classes)

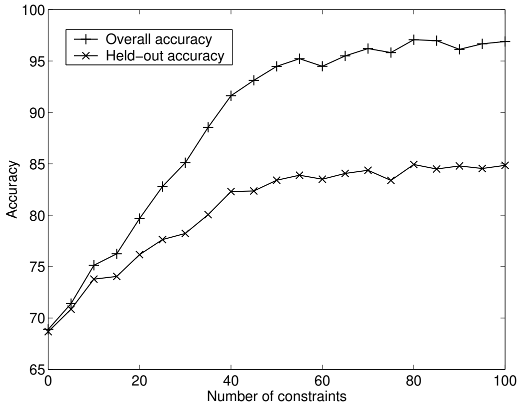
We conducted 50 trials on each data set, randomly re-ordering the data set each time, and averaged the results. Each trial is a 10-fold cross-validation run.

The soybean data set. The first data set of interest is soybean, which has 47 instances. Each instance is represented by 35 observations of a diseased soybean plant. There are four classes of soybean disease present in the data. Without any constraints, the k-means algorithm achieves a clustering accuracy of 88%, while COBWEB attains 84% (see Figure 3.3).

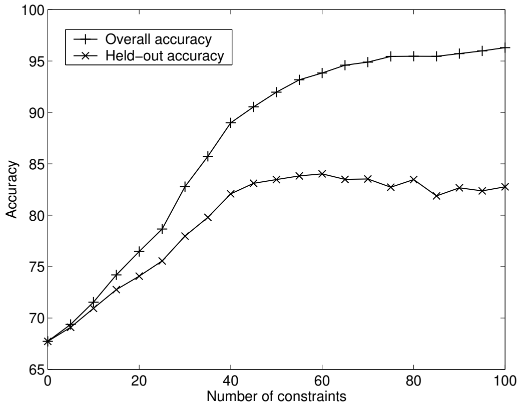
Overall accuracy steadily increases with the incorporation of constraints, reaching 98% for both algorithms after 100 random constraints. Lest the number of constraints seem high, we note that over 100 pairwise constraints can be achieved by labeling (at most) 15 items. In addition, background knowledge for a real problem may come in the form of general rules that generate large numbers of instance-level constraints. This occurs in our experiments with large data sets from real domains (see Chapters 4, 5, and 6). For the soybean data set, we find that accuracy on the held-out test set also improves, achieving 96% for both algorithms with 100 constraints. This represents a held-out improvement of 8–12% over the baseline (no constraints) for these algorithms.

An improvement in overall accuracy is unsurprising, since the clustering algorithm has access to more information that it would otherwise. However, we would not necessarily expect to see any improvement on the held-out instances at all. They do not have any guiding constraint information specified about them. The fact that we do see an improvement demonstrates that, for this data set, the knowledge encoded in the constraints is generalizable to the held-out set.

We also applied the Rand index (Equation 3.8) to the set of constraints versus the true partition. Because the Rand index views a partition as a set of pairwise decisions, this allowed us to calculate how many of those decisions were "known" by the set of constraints. (For clarity, these numbers do not appear in the figure.) For this data set, 100 random



(a) COP-KMEANS



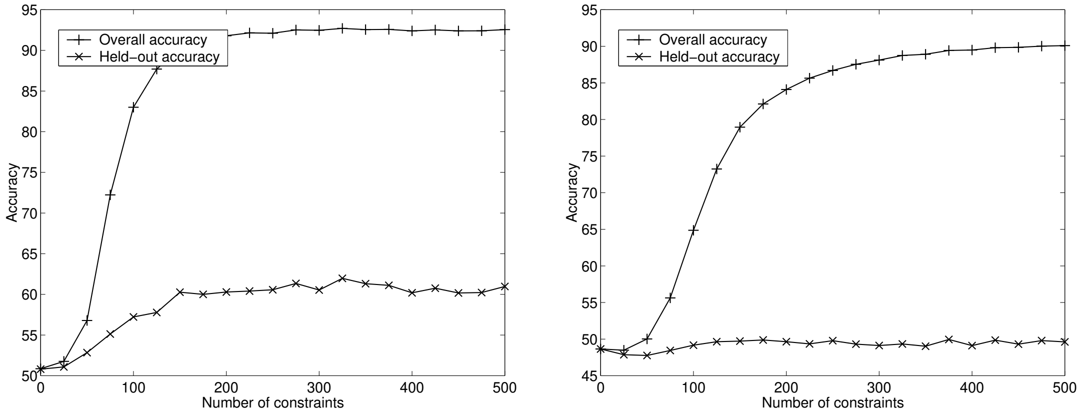
(b) COP-COBWEB

Figure 3.4: Clustering accuracy with artificially generated constraints: the mushroom data set (50 instances in two classes)

constraints achieve a mean accuracy of 48%. We can therefore see that combining the power of clustering with background information achieves better performance than either in isolation (96% > 88% and 96% > 48%).

The mushroom data set. We next turn to the mushroom data set, with 50 instances and 21 attributes, a subset of the full mushroom (agaricus-lepiota) data set. Each instance contains a (hypothetical) description of a single mushroom, and there are two classes of mushrooms: POISONOUS and EDIBLE. In the absence of constraints, the k-means algorithm divides the data set into two distinct clusters with an accuracy of 69%, while COBWEB attains 68% (Figure 3.4). After incorporating 100 random constraints, overall accuracy improves to 97% for COP-KMEANS and 96% for COP-COBWEB. In this case, 100 random constraints achieve 73% accuracy before any clustering occurs. Held-out accuracy climbs to 85% for COP-KMEANS and to 83% for COP-COBWEB, yielding an improvement of 15–16% over the baseline.

The part-of-speech data set. The third data set under consideration is the part-of-speech data set (Cardie, 1993). Unlike the other data sets studied in this section, it does not come from the UCI repository. Instead, the data set was generated to assist in the solution of a problem from Natural Language Processing. Each item represents an English word, and the label represents the word's part of speech. We used a subset of the full data set that contains 50 instances, each described by 28 attributes. There are three classes in this data set. Without constraints, the k-means algorithm achieves an accuracy of 54%, and COBWEB attains 56% (we have omitted the graphs for this data set, as COP-KMEANS and COP-COBWEB have very similar performance, just as shown in the previous two figures.). After incorporating 100 random constraints, overall accuracy improves to 91% for COP-KMEANS and 88% for COP-COBWEB. Here, 100 random constraints attain 56% accuracy.



(a) COP-KMEANS (b) COP-COBWEB

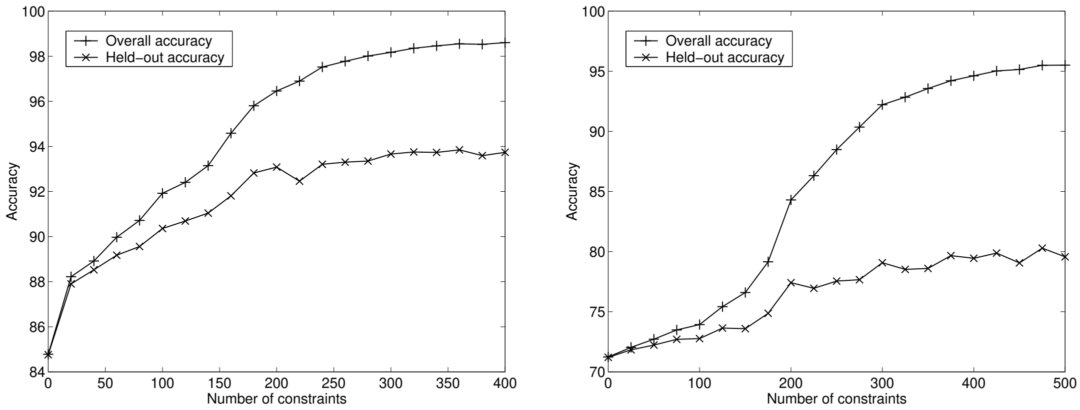
Figure 3.5: Clustering accuracy with artificially generated constraints: the tic-tac-toe data set (100 instances in two classes)

Held-out accuracy climbs to 71% for COP-KMEANS and to 74% for COP-COBWEB, yielding an improvement of 17–18% over the baseline.

The tic-tac-toe data set. Next, we focus on the tic-tac-toe data set, with 100 instances. Each instance has nine attributes that represent the positions on a tic-tac-toe board, and they are divided into two classes: a win for the X player or a win for the O player. Because this data set is larger, we experimented with more constraints. Without constraints, the k-means algorithm achieves an accuracy of 51%, and COBWEB attains 49% (Figure 3.5). In other words, the clustering algorithms are unable to do better than making a random choice between the two possible outcomes. This data set is known to be very challenging for inductive learning algorithms. As mentioned in Chapter 2, this is an example of a data set with feature values that do not correlate well with the true data labels. That is, two instances that appear very similar according to their feature values will not necessarily be in the same class.

Despite the difficulty of this problem, we are able to achieve large increases in clustering accuracy. After incorporating 500 random constraints, overall accuracy is 93% for COP-KMEANS and 90% for COP-COBWEB, although high performance is asymptotically approached well before than that. This set of constraints achieves 80% accuracy in isolation. Held-out accuracy reaches 61% for COP-KMEANS, achieving a 10% increase in accuracy. COP-COBWEB behaves somewhat worse on this data set, with held-out performance improving marginally to 50%. The lack of large gains on the held-out subset is consistent with our observations about how the feature values do not correlate well with the class labels. In other words, the constraints do not generalize well to the unconstrained held-out subset. Regardless, overall performance does improve greatly.

We believe that this data set is particularly challenging because the classification of a board as a win or a loss for the X player requires extracting relational information between



(a) iris (n = 150 items; k = 3)   (b) wine (n = 178 items; k = 3)

Figure 3.6: COP-KMEANS clustering accuracy with artificially generated constraints: the iris and wine data sets

the attributes—information not contained in our instance-level constraints. Our goal has been to simply demonstrate the benefits that constraints of any kind can provide, so we created constraints by random selection. However, if our goal were to improve performance as much as possible on this particular task, we could instead create constraints intelligently. One possibility would be to include the spatial relationships between the attributes. We could do this by creating a must-link constraint between two boards (instances) that have two adjacent X's (spatially) in the same positions.

In Chapter 2, we compared our methods to those of Klein et al. (2002), who developed a method for accommodating constraints by pre-processing the data set and using a standard clustering algorithm. In addition to adjusting the pairwise distances between items due to the specified constraints, Klein et al. also extend this information across the feature space. Thus, two items that are very similar in terms of their feature values may end up "sharing" constraint information as well. This enables a more efficient encoding of constraint information, since a handful of constraints can grow into a large set of constraints that apply to the entire data set. When true class values correlate well with feature similarity, this is a useful approach. However, we suspect that with data sets such as tic-tac-toe, automatically extending the constraints in this way may cause the algorithm's performance on unconstrained items to go down.

The iris and wine data sets. In contrast to the preceding experiments, which made use of data sets with symbolic (categorical) attributes, we also experimented with using COP-KMEANS on two UCI data sets with numeric (continuous) attributes (see Figure 3.6)[2]. The iris data set consists of 150 of different iris plants, each described by four attributes. There are three distinct classes, IRIS SETOSA, IRIS VERSICOLOUR, and IRIS VIRGINICA. It is well known that one class is linearly separable from the other two, which are not linearly separable.

[2]COBWEB cannot handle data with numeric attributes.

In our experiments, incorporating 400 random constraints with this data set yielded a 9% increase in held-out accuracy (from 85% to 94%). Overall accuracy climbed to 99%, and the set of constraints achieved 66% accuracy. The wine data set consists of 178 observations of different wines, each measured with 13 attributes. There are three classes in the data set, which correspond to which Italian cultivar produced the grapes. Behavior on this data set was also consistent with our other experiments; held-out accuracy improved 9% from 71% to 80%, and overall accuracy climbed to 96%. The constraint set achieved 68% in isolation.

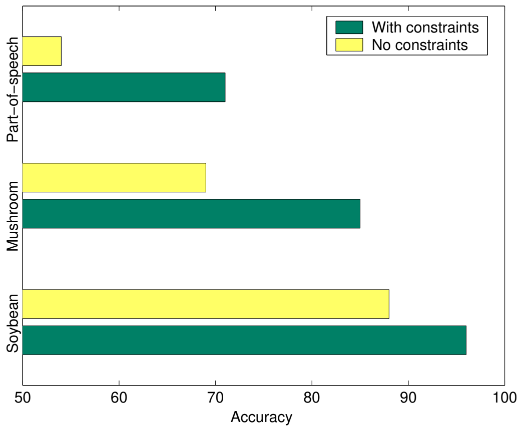
Discussion of results. Our experiments on six different data sets have shown that both of the constrained clustering algorithms exhibit significant accuracy improvements when using constraints. Figure 3.7 summarizes the improvements in held-out accuracy that we observed for certain numbers of constraints. For the three smallest data sets, soybean, mushroom, and part-of-speech, we show the accuracy results for COP-KMEANS (part (a)) and COP-COBWEB (part (b)) when incorporating 100 randomly generated constraints. Both algorithms perform remarkably similarly, showing improvements of 12–18% over the baseline (unconstrained) algorithms. We further show the results for COP-KMEANS on the three largest data sets, tic-tac-toe, wine, and iris, when incorporating 500 randomly generated constraints. We observe improvements of 9–10%.

We conclude from these experiments that even randomly generated constraints can improve clustering accuracy. As an additional, unexpected side effect, we have seen that individual constraints often generalize enough to improve performance on other items in the data set. This is shown in the held-out accuracy results for all six data sets. This behavior suggests that we can improve performance on real problems even if our domain knowledge is incomplete, as is often the case. The constraints that we do encode may boost performance on items without any explicit constraints as well.

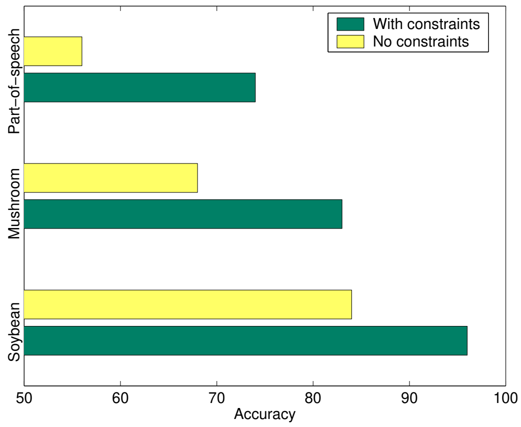
These experiments also demonstrate how our algorithms can easily handle constraint relations that apply to only a subset of the data. This contrasts with the work that has been done with contiguity constraints (see Chapter 2), where the constraint relation must encompass all data items. In addition, we accommodate constraints that indicate when two items should not be grouped together as well as those that indicate when they should be; this is a novel constraint formulation. Finally, our algorithms are not restricted to a single source of constraining information, as with previous work on image segmentation. Instead, we can accommodate a combination of information from a variety of sources, as long as it can be expressed as instance-level pairwise constraints.

3.5.3 Analysis of Artificial Constraint Results

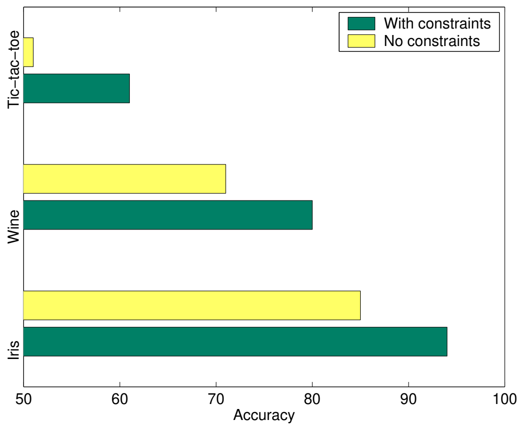
In addition to the accuracy improvements we have just presented, we also observed several other interesting results in our experiments with artificial constraints. First, we find that the use of constraints can aid COP-COBWEB in selecting an appropriate value for k, the number of clusters present in the data. Second, the constraint satisfaction problem exhibits behavior consistent with that observed by other researchers. Interestingly, the hardest constrained clustering problem instances are those with an intermediate number of constraints specified. Finally, we find that the addition of constraints results in a significant drop in the mean number of iterations that COP-KMEANS requires before convergence.



(a) COP-KMEANS: Soybean, mushroom, and part-of-speech data sets; 100 constraints



(b) COP-COBWEB: Soybean, mushroom, and part-of-speech data sets; 100 constraints



(c) COP-KMEANS: Tic-tac-toe, wine, and iris data sets; 500 constraints

Figure 3.7: Summary of held-out accuracy improvements obtained for COP-KMEANS and COP-COBWEB using artificial constraints

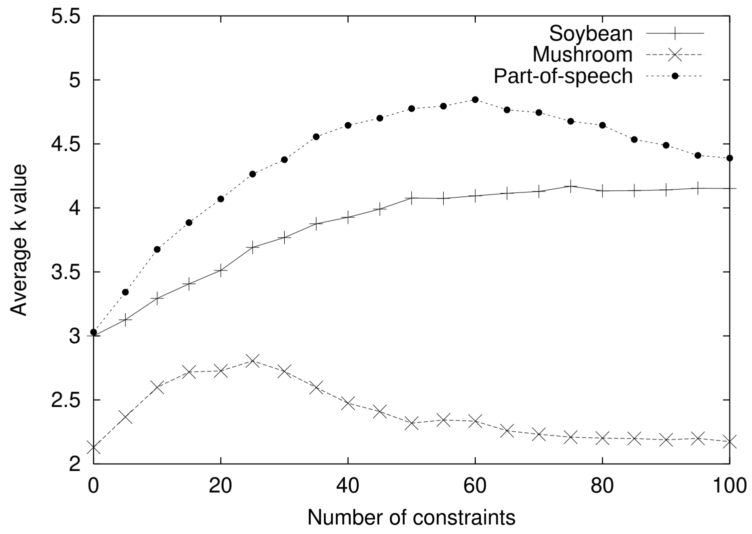
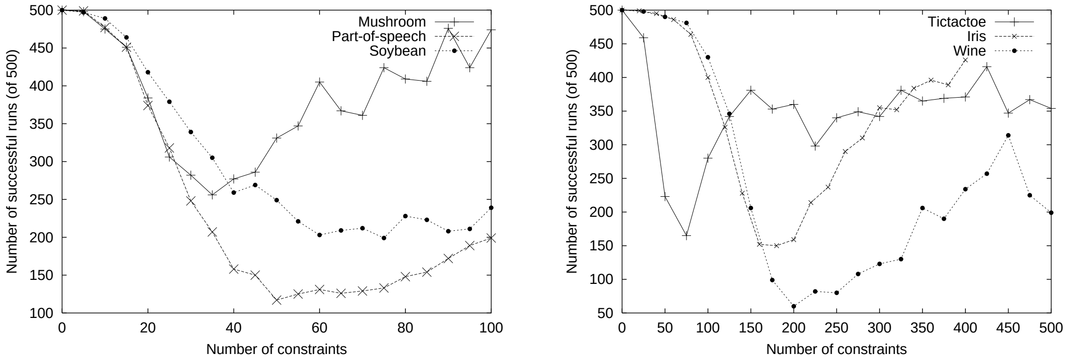


Figure 3.8: COP-COBWEB: Selected value for k, averaged over 500 runs

COP-COBWEB: Number of clusters. The COP-COBWEB algorithm attempts to automatically determine the best value for k, the number of clusters in the data. Figure 3.8 plots the mean value of k selected by COP-COBWEB for three data sets with varying numbers of constraints. In the absence of constraints, COP-COBWEB consistently selects $k = 3$ for the soybean data set, which conflicts with the data labels; they claim that there are four distinct clusters present. However, as more constraints are provided, the mean k value chosen by COP-COBWEB nears the "true" value. In contrast, COP-COBWEB chooses a good value for k for the mushroom data set in the absence of constraints. The algorithm creates too many clusters for small numbers of constraints, but it selects values close to $k = 2$ as more constraints are available. A similar trend is apparent with the part-of-speech data set. The average solution obtained without constraints has $k = 3$, but as the number of constraints increases, so does the mean number of clusters. COP-COBWEB's selection of k appears to be drifting away from the "correct" value of $k = 3$, but for larger numbers of constraints, the mean k value declines again. We conclude that small numbers of constraints may or may not help COP-COBWEB in selecting the correct value of k. Larger numbers of constraints, however, are much more effective.

COP-KMEANS: Constraint satisfaction hardness. In our experiments with COP-KMEANS, we observe an interesting trend in terms of the relative difficulty of finding a partition that satisfies all of the specified constraints. For small numbers of constraints, it is relatively "easy" to find a satisfying solution. As the number of constraints increases, the problem becomes more difficult. However, as the number of constraints increases further, the problem becomes easier again. This is consistent with observations made by Selman et al. (1996). They examined the computational difficulty of solving randomly-generated 3-SAT formulas by plotting the formula complexity (ratio of clauses to variables) against the amount of work done by the SAT solver. They found that the easiest formulas to solve



(a) Mushroom, part-of-speech, and soybean data sets

(b) Tic-tac-toe, iris, and wine data sets

Figure 3.9: COP-KMEANS: Number of successful runs, out of 500

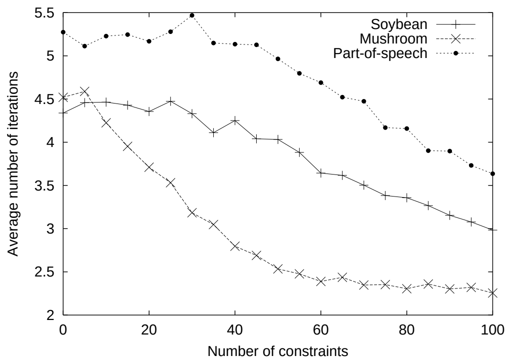
tended to be those either with low complexity or those with high complexity. The hardest problems were the intermediate ones.

Recall that k-means is a greedy hill-climbing algorithm. COP-KMEANS preserves this quality, which means that it may fail to find a satisfying solution for a given set of constraints. As previously indicated, running the algorithm multiple times with different random seeds alleviates this problem empirically. Figure 3.9 plots the number of successful runs, out of 500, for each data set and various numbers of constraints. A successful run is one that terminates with a satisfying solution. Each data set exhibits a minimum where it has the fewest successful runs. The exact location of this point depends on the complexity of the problem, which involves both the size of the data set and the number of clusters present. Since the hardest problems are the ones most likely to fail to find a satisfying solution, this trend corresponds very well to the hardness results reported by Selman et al.

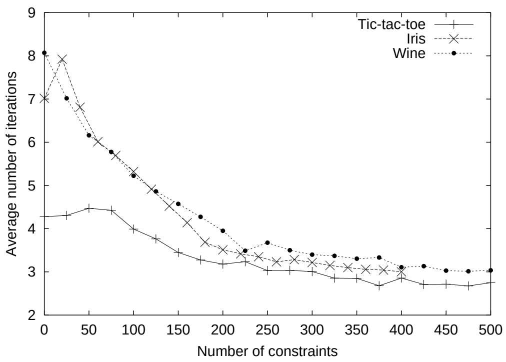
COP-KMEANS: Empirical convergence speed. In our discussion of the formal properties of COP-KMEANS, we mentioned that empirically, we find that constraints reduce the number of iterations required before convergence. This is demonstrated in Figure 3.10. For all six data sets, as the number of constraints available to the algorithm increases, the mean number of iterations required before convergence decreases. The effect is especially pronounced for the data sets shown in Figure 3.10b. These data sets are larger than those that appear in Figure 3.10a, so the number of iterations required in the absence of constraints is correspondingly higher. However, after the incorporation of only a modest number of constraints, the mean number of iterations required drops below four. In general, constraints provide a way to achieve convergence in fewer iterations.

3.6 Empirical Comparison to Seeded Clustering

In Section 2.6, we described the work of Basu et al. (2002), who incorporate domain knowledge expressed as a set of partial labels on a data set. We also compared their methods



(a) Mushroom, part-of-speech, and soybean data sets



(b) Tic-tac-toe, iris, and wine data sets

Figure 3.10: COP-KMEANS: Number of iterations before convergence, averaged over successful runs (max 500)

to ours, in general terms. The most significant distinction is that our instance-level encoding of domain knowledge can express a wider range of information than data labels can. It is always possible to encode any set of data labels as a set of instance-level pairwise constraints. However, there is not always a set of data labels that can express the same information for an arbitrary set of instance-level constraints.

After contrasting their approach to ours in more detail, we will present results comparing our systems empirically.

3.6.1 Seeded Clustering versus Constrained Clustering

The motivating goal of the work done by Basu et al. is to enable a clustering algorithm to focus on more promising regions of the search space. The k-means algorithm can become trapped in local minima, and their modified algorithms seek to avoid those areas by "seeding" the partition with user-defined labeled examples of correctly labeled items. Each group of items with the same label becomes one of the initial cluster groups used by the k-means algorithm. If k is larger than the number of distinct labels, then the seeded algorithms select the remaining required cluster centers randomly. In essence, they have developed a clever method for using partially labeled data to improve the starting point for the k-means algorithm.

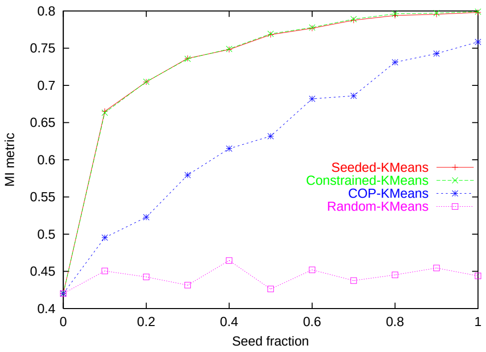
Basu et al. propose two algorithms based on this method. The first is Seeded-KMeans, which uses the labeled data to identify the initial cluster groups but then is free to modify the data assignments (i.e., break constraints) as clustering progresses. The second algorithm is Constrained-KMeans, which selects the initial cluster groups in the same way but then disallows any changes to the cluster assignments of the labeled items. COP-KMEANS is most like Constrained-KMeans; the main difference is that Constrained-KMeans requires that domain knowledge be specified as individual data labels.

In addition to guiding the k-means algorithm to more promising regions of the search space, it is also important to have the ability to override the objective function when appropriate. Constrained-KMeans and COP-KMEANS have this ability; they prioritize constraints over the output of the objective function. Seeded-KMeans is more conservative; it dictates a constrained starting point but then allows the objective function complete control over what happens from that point on. Basu et al. have shown that this enables Seeded-KMeans to be more robust in the presence of noisy constraints; it can ignore constraints that do not fit well with the objective function's decisions. Constrained-KMeans and COP-KMEANS do not recover well from noisy constraints, since they treat each one as a hard constraint. We have not experimented with noisy constraints. We take the view that hard constraints should only be used to express information that is known to be very reliable. If the background knowledge under question is approximate, noisy, or heuristic, soft constraints are a better formulation to use.

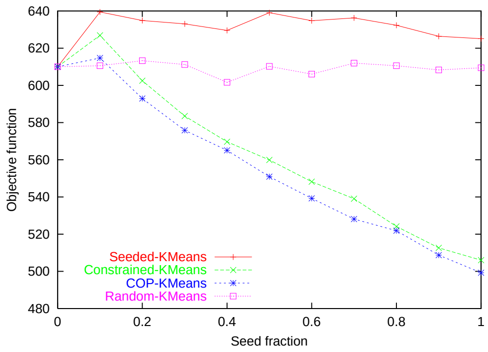
3.6.2 Empirical Comparison

Basu et al. conducted experiments on several data sets, of varying sizes, to compare their algorithms to their reimplementation of COP-KMEANS. They evaluated the performance of all three algorithms in two ways. First, they calculated the *mutual information* (MI) of each partition, which measures the amount of statistical information overlap between the cluster representations and the true data labels. In other words, a high MI indicates a large amount of agreement between the clusters and the correct categorization of the data. Second, they computed the value of the objective function obtained by the output of each algorithm. In this case, the goal was to *maximize* the objective function value. There are some problems with evaluating clustering based on the value of the objective function. As we have mentioned, constraints often represent information that overrides the objective function. Thus, a solution that satisfies the constraints may appear worse than one that does not, if we only examine the objective function value. Consequently, the MI values are more informative.

Figure 3.11a shows the MI performance for Seeded-KMeans, Constrained-KMeans, and COP-KMEANS on the Small-20 Newsgroups data set. A fourth algorithm, Random-KMeans, is included as a baseline. It is simply the original k-means algorithm, which initializes itself using items chosen randomly from the data set and does not make use of any constraints. Basu et al. report MI numbers for a single 10-fold cross-validation run. The x axis, "seed fraction," indicates the fraction of the training items that are labeled before clustering. For example, a seed fraction of 0.2 indicates that 20% of the 1800 documents in the training set, or 360 documents, have been labeled. In this figure, the seeded clustering algorithms achieve higher MI values than COP-KMEANS does. For the four other data sets analyzed in this fashion, COP-KMEANS performs very similarly to the seeded clustering methods (i.e., within 0.01 or 0.02 in terms of MI values). These data sets include the Newsgroups data set, which is a superset of Small-20 Newsgroups. It is not clear whether the lower performance of COP-KMEANS on Small-20 Newsgroups is caused by unusual features of this subset or simply due to the smaller data set size. It is also possible that with a different random initialization of



(a) MI versus increasing seed fractions for the Small-20 Newsgroups data set



(b) Objective function versus increasing seed fractions for the Yahoo! data set

Figure 3.11: Experimental comparison of seeded clustering to COP-KMEANS; figures from Basu et al. (2002)

cluster centers (i.e., another run of COP-KMEANS), the results on Small-20 Newsgroups would be more consistent with results on the other data sets.

Figure 3.11b instead reports on the objective function values obtained; the data set under consideration is the Yahoo! data set. For this data set, Basu et al. observe minimal improvements in MI (not shown) for all three algorithms. However, as Figure 3.11b shows, the use of hard constraints decreases the objective function value. This is consistent with our earlier comments on what happens when the constraints override the objective function. A lower objective function value is not necessarily good or bad. It simply indicates that the constraints disagree with the built-in bias of the algorithm; this is of concern only if the constraints are not reliable. The objective function values may provide insights into the data set or problem, since this behavior tells us how much the bias and constraints agree (or disagree). However, this evaluation method does not tell us anything useful about the performance of the algorithms.

Basu et al. also conducted experiments on the same iris data set that we used previously in this chapter. They evaluated the three algorithms in terms of MI when provided with about 40 labeled items (a "seed fraction" of 0.3) and varying amounts of noise in those labels.[3] This translates to 780 pairwise constraints. We have already shown that, for a randomly chosen set of 400 constraints, we observe a 14% improvement in clustering accuracy (to 99% accuracy), and a 9% improvement in held-out clustering accuracy, on this data set (Figure 3.6). Basu et al. found that both seeded clustering algorithms achieved an MI value of 0.94, while COP-KMEANS had a lower value of 0.88. However, these results were obtained from a single cross-validation run. Because the MI values for Random-KMeans over several trials ranged from 0.80 to 0.90, we suspect that more reliable conclusions could be drawn after several cross-validation runs, ideally with different splits of the data set.

[3] We are not concerned with noisy constraints, as explained earlier.

In this section, we have reviewed the semi-supervised clustering work of Basu et al. They developed two seeded clustering methods, Constrained-KMeans and Seeded-KMeans. When labeled examples of the true classes present in the data set are available, then these algorithms present useful methods for enforcing those constraints, especially in the presence of noisy labels. In some cases, these algorithms out-perform COP-KMEANS in terms of MI; in other cases, there is little difference. It may be possible to incorporate the ideas of Basu et al. in the clustering initialization phase of COP-KMEANS. K-means is sensitive to the choice of initial cluster centers, and Basu et al. have shown that an intelligent selection of those centers produces measurable benefits. We have already seen that COP-KMEANS performs well on the general case, where our domain knowledge may not be expressible as data labels. If we can extend the seeding approach to this case, we expect to see further gains on those problems as well.

3.7 Summary

This chapter examines the details of constrained clustering by first presenting a general method for enhancing an existing clustering algorithm to allow it to make use of domain knowledge. This domain knowledge must be expressed as a set of instance-level, pairwise, hard constraints on the data items. After demonstrating that the scenarios presented in Chapter 1 can be accommodated using this constraint formulation, we presented two enhanced clustering algorithms, COP-KMEANS and COP-COBWEB. In experiments with artificial constraints, we have shown that the new algorithms can successfully incorporate constraints to improve their clustering performance.