CHAPTER 2
RELATED WORK

Computers have been weak in their ability to understand and process information that contains abstractions and complex webs of relationships, but they are improving.
— Raymond Kurzweil, The Age of Intelligent Machines, 1990

In the preceding chapter, we outlined the limitations of general clustering methods in terms of their inability to accommodate domain knowledge about a problem. We are not the first to point out this shortcoming; several modified clustering algorithms have been developed that attempt to accommodate problem-specific information in unsupervised approaches. These new algorithms are neither supervised nor unsupervised but fall somewhere in between; they are sometimes referred to as *semi-supervised* (Cohn et al., 2000; Basu et al., 2002). In this chapter, we first provide some background on the various types of clustering algorithms (Section 2.1) before moving on to discuss how others have enhanced those algorithms to allow them to use domain knowledge.

Domain knowledge is a very broad term, and background information can come in a variety of formats. We have divided this information into four categories: structural information about the data (*global constraints*), minimum or maximum cluster capacities (*cluster-level constraints*), heuristic rules (*feature-level constraints*), or partial labels and individual relationships between data items (*instance-level constraints*). Section 2.2 states the constrained clustering problem formally, and Sections 2.3 through 2.6 discuss relevant work with each kind of domain knowledge in turn. We conclude with an analysis of how the novel methods that we propose fit into this family of work and which kinds of domain knowledge our methods accommodate.

2.1 Clustering Algorithms

Clustering algorithms seek an organization P of a data set D that optimizes an objective function $f : P \to \mathcal{R}$. Informally, the goal is to create clusters such that items within the same cluster are very similar, and items in different clusters are very different. Many algorithms make use of a distance function $d : D \times D \to \mathcal{R}$ to measure the similarity or difference of two points.

In general, finding the optimal solution to a clustering problem is NP-hard (Garey and Johnson, 1979; Křivánek, 1991). Anderberg (1973) notes that the number of possible ways to partition n items into k clusters is given by $\mathcal{S}_n^{(k)}$:

$$\mathcal{S}_n^{(k)} = \frac{1}{k!} \sum_{i=0}^{k} (-1)^{k-i} \begin{pmatrix} k \\ i \end{pmatrix} i^n$$

For example, this means that the number of ways to sort 25 items into 5 groups is prohibitively large:

$$\mathcal{S}_{25}^{(5)} = 2,436,684,974,110,751$$

An exhaustive search through all possible solutions will be infeasible even for moderately-sized data sets. Consequently, several methods have been developed that can reach a good, if not optimal, solution in a reasonable amount of time. A comprehensive survey of all (partitioning) clustering algorithms is beyond the scope of this thesis; for references, see Anderberg (1973), Hartigan (1975), and Gordon (1981). A partitioning algorithm takes in a data set D and returns a set of clusters $P = \{C_1, \ldots, C_k\}$ that form a partition of the items in D. This means that the clusters C_i are non-overlapping ($C_i \bigcap C_j = \emptyset, i \neq j$) and completely cover the data set ($\bigcup_i C_i = D$). This contrasts with hierarchical clustering algorithms, which create a set of multiple partitions of the same data, in order of increasing generality. In a full hierarchy, the topmost partition is a single cluster that contains every item in the data set, and the lowest partition contains n clusters, one per item.

We will next present an overview of the basic partitioning algorithms before proceeding to a discussion of the ways that they have been enhanced to accommodate additional information not present in the data set. We will reserve a direct comparison to our methods for the next section, when we discuss constrained clustering. This overview is purely for background purposes; several of these algorithms will be mentioned again in the next section.

There are two major kinds of partitioning algorithms: batch and incremental. *Batch* clustering algorithms examine the entire data set at once to determine the best way to organize the data. *Incremental* algorithms develop a partition of the data one step at a time, where each step incorporates a single data item. Batch algorithms require that all of the data be present before processing begins, while incremental algorithms are suited for on-line applications where the data is an incoming stream of observations.

2.1.1 Batch Partitioning Algorithms

There are three common batch approaches to the clustering problem. The first method operates on a set of instances represented as feature vectors and attempts to find a set of clusters that organizes this data well. The second approach starts with a dissimilarity matrix and successively merges pairs of similar items. The third method views the input items as nodes in a graph, connected by edges that indicate the distance between each pair of points.

Feature-vector clustering. *K-means clustering* (MacQueen, 1967), also known as *nearest centroid sorting* and *iterative relocation* (Forgy, 1965; Jancey, 1966), is a greedy, hill-climbing method. It searches for the best set of k cluster centroids, which also determines the structure of the partition by assigning each instance to its nearest centroid. The centroid of a cluster is a point that represents the mean or center of gravity of the cluster's items. Starting with some initial configuration of the data set, the algorithm iteratively improves its estimate of the cluster centroids until no further changes are possible. It does this by alternating between assigning all points to their closest cluster centers and updating the cluster centers. In the field of data mining, several variations on the basic k-means algorithm have been developed to improve its scaling properties (e.g., BIRCH (Zhang et al., 1996) and CURE (Guha et al., 1998)). We will develop a *constrained* clustering version of the k-means algorithm in Chapter 3.

The alternating nature of the k-means algorithm is reminiscent of Expectation-Maximization (EM) techniques (Dempster et al., 1977), and in fact, various k-means algorithms can be implemented using EM (Mitchell, 1997). The Autoclass clustering algorithm (Cheeseman et al., 1988) is a batch partitioning method that makes use of EM methods in a Bayesian framework.

Dissimilarity matrix clustering. Other batch clustering algorithms operate on a matrix of dissimilarities between items in the data set. The *single link* (Sibson, 1973) and *complete link* (Anderberg, 1973) methods are the most common algorithms in this category. In each case, the algorithms begin with a set of n clusters, with each item in its own cluster. They then successively merge clusters together until a stopping criterion is met (number of clusters, maximum cluster diameter, minimum cluster separation, etc.). These algorithms differ in how they assess the distance between clusters. Building on the point distance function $d(x, y)$ for points $x, y \in D$, the distance between clusters C_i and C_j is given by Equation 2.1 for single-link clustering and by Equation 2.2 for complete-link clustering.

$$d(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y) \tag{2.1}$$

$$d(C_i, C_j) = \max_{x \in C_i, y \in C_j} d(x, y) \tag{2.2}$$

Because the single-link method merges clusters based on the closest elements in each cluster, it is good at finding long "chaining" clusters. In contrast, the complete-link method is better at finding well-separated clusters, because it only merges clusters if the two most widely separated items are sufficiently close.

Graph clustering. Another common batch clustering method views the data set as a *graph* with n nodes and $\frac{1}{2}n(n-1)$ edges. Each pair of points is connected by an edge that is weighted by the distance between the points. It is then possible to compute the minimum spanning tree (MST) over this graph and break the $k-1$ edges with largest weight to obtain a partition of k clusters (Zahn, 1971). Others have used different graph structures in the same way (e.g., Gabriel graph or RNG (Urquhart, 1982)).

We have described three general batch partitioning approaches: methods that operate on feature vectors, dissimilarity matrices, or distance graphs. We next review incremental partitioning algorithms.

2.1.2 Incremental Partitioning Algorithms

Incremental algorithms are also referred to as *online* algorithms. Rather than having access to all of the data at once, these algorithms must continually adapt the clustering to an incoming stream of data. Therefore, incremental algorithms are sensitive to the ordering of the data. In some cases, such sensitivity is in fact desired. For example, some clustering algorithms aim to emulate human learning behavior, which is order-sensitive (Fisher, 1987). There are two major types of incremental partitioning algorithms: capacity clustering algorithms and algorithms derived from a corresponding incremental hierarchical algorithm.

Capacity clustering. One incremental clustering method is known as *region growing* or *sequential clustering*, where each cluster has a given capacity. Each item d in D is successively examined and added to the closest cluster C. If C reaches its maximum capacity, it is removed from consideration for the rest of the data set. Brown and Gunderson (2001) used a sequential algorithm to analyze computer criminal attacks and determine criminal preferences. Their goal is to use these preferences to facilitate a warning to similar sites of the likelihood of an impending attack. This kind of algorithm is appropriate in situations where specifying a cluster capacity makes sense. We will return to the notion of cluster capacities later in this chapter, where the capacity is specified as a constraint on the behavior of the algorithm.

Variants of hierarchical clustering algorithms. Hierarchical clustering algorithms are the alternative to partitioning clustering algorithms. Rather than creating a partition of the data, hierarchical algorithms create several levels of partitions, each at a different level of generality (e.g., each with a different number of clusters). Some of these hierarchical algorithms are also incremental. A simple way to create an incremental partitioning algorithm is to specify a method for selecting one of the partitions created by a hierarchical method. For example, COBWEB (Fisher, 1987) is an incremental algorithm that returns a partial hierarchy of the items in D. We constructed a partitioning version of COBWEB (Wagstaff and Cardie, 2000) that simply returns a partition corresponding to the top level of COBWEB's hierarchy. In Chapter 3, we will also develop a constrained clustering version of COBWEB.

2.2 Constrained Clustering

The clustering algorithms that we have presented vary greatly in their details, but they share a common attribute: they all operate solely on a data set, usually composed of individual feature vectors. However, we often would like to combine information from multiple sources. Specifically, we identify the data set D as the *observations* and a supplemental source Con (for *Con*straints) as the *domain knowledge* or *problem-specific knowledge* we would like to incorporate. In this section, we will first discuss two ways to incorporate this kind of information in methods that do not use clustering, then indicate the connections to our work. Next, we will formalize the constrained clustering problem, which is the focus of this dissertation.

2.2.1 Related Approaches with Non-clustering Methods

The motivation behind our methods is a desire to supplement regular learning methods with the ability to incorporate additional knowledge about the problem or domain. Two important techniques with the same goal have been developed for use with supervised learning methods: co-training and learning with hints.

Semi-supervised learning with multiple views. *Co-training* is a learning architecture that incorporates two supervised learning algorithms that collaborate by alternating roles as teacher and student (Blum and Mitchell, 1998). Each learner has access to the same data items, but they see a different set of features or *view* of those items. The goal is to combine the knowledge embedded in the two views into a unified organization of all of the items. This approach is semi-supervised in that it starts with a subset of the data set already labeled. Each learner trains on the labeled subset and then classifies each of the unlabeled items. They also indicate their confidence in each of the labels they generate. The most confidently labeled examples from each of the learners are added to the labeled subset of the data, and the process iterates. The hypothesis is that, under certain assumptions about the independence of the two views, the learners will complement each other. Each may be able to confidently label items that the other learner could not. This is possible because, even if the learners have the same weaknesses, they have access to a different set of observations and trends in the data, and their inductive classifications will therefore not be subject to the same influences.

Although the motivation behind co-training is similar to that of our work, there are important differences between the two approaches. First, co-training requires the use of supervised learning algorithms that can output a confidence with each labeling decision they make. In contrast, our methods are built on unsupervised methods, so they can operate in the absence of explicit data labels. Second, the co-training accuracy guarantees depend critically on the independence of the two data views. If the two views are not independent, then the two learners may inadvertently reinforce each other's bad decisions. In practice, this independence can be difficult to obtain (or determine). We do not need to impose strict requirements about the independence of data views in our work. Instead, we are able to combine two sources of knowledge (the data observations and the domain knowledge encoded as constraints) even if there is overlap in the information they contain. In fact, we will demonstrate that our methods work in cases where the constraints are very consistent with the data labels based on the feature values (see our experiments with UCI data sets in Section 3.5) and in cases where they are not (i.e., the constraints are a truly complementary source of knowledge; see our experiments with the tic-tac-toe UCI data set).

Supervised learning with hints. Another related area of research uses domain knowledge in the form of "hints" with supervised learning methods (Abu-Mostafa, 1995). Hints are defined as "properties of the target function that are known to us independently of the training data." The hints are used to create virtual examples whose feature values are consistent with the specified target function properties. For example, if the classification of objects in the data set is invariant under rotations of those objects, this approach might create virtual examples that are rotated versions of other input objects but have the same labels as the originals. These hints supply knowledge not already available in the observations and have been shown to enable the creation of much more reliable models of financial markets than without such hints (Abu-Mostafa, 2001).

To cast supervised hints in our language, the hints function as hard constraints on the output (since the supervised learning method attempts to reduce the classification error on the virtual examples to zero). By specifying different error calculations for the virtual examples, Abu-Mostafa allowed for the hints to function as soft constraints as well (by tolerating some misclassification error for the virtual examples).

The major difference between this work and what we propose is that it operates in a supervised framework, while we focus on unsupervised methods. Thus, the hint-supervised approach is of use for situations where a full set of data labels is already available. If a data item does not have a label, then the hints cannot be applied: the system can create the appropriate virtual example, but it will be unable to label that example, and the learning system will be unable to use it. Again, our constrained clustering approach can be used whether or not a full set of labels is available. In addition, the goal of the supervised methods is to learn from examples; in other words, they attempt to induce a function from the input examples to a set of possible labels. Clustering algorithms do not learn from examples; instead, they attempt to organize observations into meaningful groups based on patterns inherent in the observations. To conclude, although the motivation is very similar, the hint-supervised approach will be used in different contexts and on different problems than our constrained clustering approach will be.

2.2.2 The Constrained Clustering Problem

Having reviewed important non-clustering work that combines domain knowledge with learning methods, we now present a formal description of the constrained clustering problem. A clustering task can be phrased as an optimization problem:[1] the goal is to select a partition P from the set of *feasible* partitions $\Phi(D)$ such that $f(P)$ is minimized (or maximized), where f is the objective function. Feasible partitions are clusterings of the data set D that satisfy requirements imposed by the algorithm. Thus, $\Phi(D)$ serves to exclude invalid hypotheses, such as solutions with overlapping clusters. Each member of $\Phi(D)$ is a set of clusters $C_1, \ldots, C_k$ that satisfy the following properties:

$$\emptyset \subset C_i \subseteq D \tag{2.3}$$

$$C_i \bigcap C_j = \emptyset,\ i \neq j \tag{2.4}$$

$$\bigcup_i C_i = D \tag{2.5}$$

That is, the clusters C_i must be non-empty, disjoint, and collectively cover D. This notion can be extended so that $\Phi(D)$ also incorporates problem-specific constraints, thereby enabling the further exclusion of unpromising areas of the search space. We impose an additional requirement that all partitions in $\Phi(D)$ satisfy all specified constraints Con:

$$C_i \models Con \tag{2.6}$$

We will formalize our definition of Con, the set of constraints, in Section 3.1.1. For now we will use the more general notion that Con contains any information restricting which partitions are considered valid possibilities by the algorithm. This view of constraints encompasses a wide range of related work, as the rest of this chapter will indicate. In fact, the parameters specified as inputs to a clustering algorithm could also be considered constraints. For k-means clustering, the value k is a constraint on the output: only partitions with exactly k clusters are permitted. However, we follow the approach of Tung et al. (2001) in excluding these parameters from our examination of constrained clustering, since they are built into the algorithm itself.

Next, we explore existing work on combining clustering algorithms with domain knowledge. We have identified relevant work that uses each of four different kinds of domain

[1] The problem statement presented here is adapted from Batagelj and Ferligoj (1998).

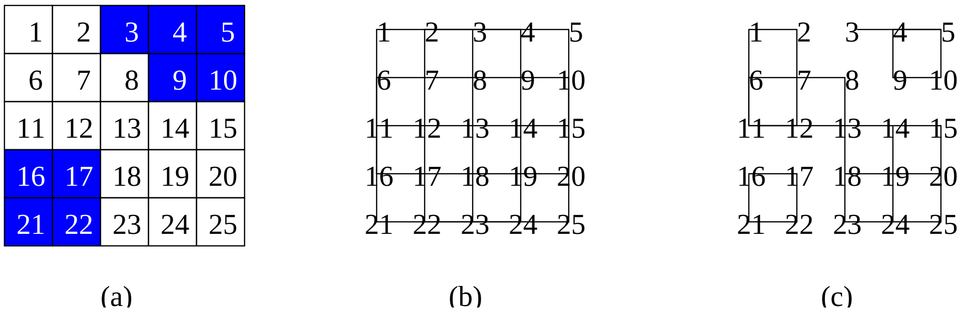

Figure 2.1: Image segmentation. (a) the image with pixels numbered; (b) the corresponding neighborhood (contiguity) relation; (c) a three-cluster result after clustering the pixels subject to relation $N(p)$

knowledge. After discussing each of these categories, we describe which kinds of knowledge our methods can handle.

2.3 Global Constraints

The supplemental knowledge source Con may contain information that applies to the data set as a whole. We refer to this kind of information as a *global* constraint. It may take the form of a neighborhood relation or some more general relation between items. In this section, we review methods for incorporating both kinds of global constraints.

2.3.1 Neighborhood Information

Clustering algorithms are often applied to problems where the data items are arranged according to *structural* or *neighborhood* information. For example, images consist of individual pixels that are related by their two-dimensional positions. Figure 2.1 illustrates a simple example. The image in part (a) is composed of 25 pixels and has three distinct regions.

Spatial contiguity. When attempting to *segment* this image into a number of regions via clustering, it is a common hypothesis that two adjacent pixels are more likely to belong to the same class than two widely separated pixels are. More precisely, we seek to enhance the *contiguity* of the clusters. It is therefore often useful to define, for each pixel, a set of *neighbors*. This set is usually either the four immediately adjacent pixels (i.e., the north, east, south, and west neighbors) or the full set of eight possible neighbors (i.e., the diagonal neighbors as well). For a given pixel p, the neighborhood, $N(p)$, can be expressed as one of

$$N(p) = \{p' \mid d_S(p, p') \leq 1\} \tag{2.7}$$

$$N(p) = \{p' \mid d_S(p, p') < 2\} \tag{2.8}$$

where d_S is the Euclidean distance according to the (spatial) image layout. Note that this distance computation is completely independent of $d(x, y)$, which is the distance that is computed while actually clustering. The distance measure $d(x, y)$ relies on the *features* of each pixel, such as its color or intensity, while d_S only examines the spatial position of each pixel. Equation 2.7 describes the four-member neighborhood, and Equation 2.8 describes the eight-member neighborhood. We can express $N(p)$ in terms of a relation: $\{(d_i, d_j) \mid d_j \in N(d_i)\}$. Figure 2.1b shows the four-member neighborhood relation on the sample image, with neighbors connected. Specifying $N(p)$ for a data set is similar to viewing it as a Markov Random Field (Kinderman and Snell, 1980), where the classification of each item is dependent on the classifications of the neighboring items but not on any items farther away.

The neighborhood of p, $N(p)$, is a way to capture structural information about a data set. In our formulation, the corresponding relation forms Con. Neighborhoods are not restricted to pixels and two-dimensional images. For example, if each data item d represents a molecule in three-dimensional space, Con could be defined to indicate molecular bonds in a lattice. More generally, Con can represent any relational information between items in the data set. We will see an example of our constrained clustering approach applied to the problem of Mars spectral analysis in Chapter 6. It is an image segmentation problem, and as we will see, our methods can easily incorporate spatial contiguity constraints.

We now discuss five methods that have been proposed for incorporating neighborhood information in clustering algorithms and compare our approach to each.

Adding Positional Features. The most obvious way to incorporate a notion of spatial location in a data set is to simply add additional features that encode an item's position to the data set. For a two-dimensional image, these features might be the *row* and *column* of each pixel in the image. The distance function $d(x, y)$ would then naturally assign a higher similarity value to pixels that are close together and lower values to those that are far apart. An advantage of this method is that it is very simple; it requires no modification to the algorithm being used. However, while this approach adds a *preference* for spatially contiguous clusters, it does not *enforce* a spatial requirement. Our work with image analysis in Chapter 6 allows for the specification of a constraint strength, so that the analyst running the clustering algorithm can control the degree to which contiguity is enforced. In addition, simply adding positional features is a solution that is restricted to the contiguity issue alone. We allow for the specification of knowledge from a variety of sources.

Duplicating Neighbors as Features. Another commonly proposed (but often impractical) method is to augment each item in the data set with *several* sets of additional features: one set per neighbor in the data set (Jain and Farrokhnia, 1991; Masson and Pieczynski, 1993; Roberts et al., 1996). Essentially, data item d_i receives duplicates of each of its neighbors' features. It is easy to see that this approach causes an immense increase in the data set's dimensionality, which will affect virtually every clustering algorithm's performance. This is important for work with real problems; the data set we experiment with in Chapter 6 has up to 1024 features per pixel. Like the previous approach, neighbor duplication is simple to implement but does not actually enforce a spatial requirement. We will compare our approach to this method in more detail in Chapter 6.

Modifying the Distance Calculation. Each of the preceding methods makes modifications to the data set D. It is also possible to incorporate spatial information by changing how the distance between two items is calculated, without modifying the data set directly (Oliver and Webster, 1989). This approach folds the observation and background knowledge together into one information source, which has the benefit of enabling the use of regular clustering methods. However, a disadvantage is that the two sources of information are mixed, and interpreting the results becomes more difficult. It may not be possible to determine whether items were clustered together due to intrinsic similarity in their observations or because of their spatial proximity.

In addition, tinkering with the distance metric is undesirable because it reduces the algorithm's applicability. The algorithm becomes so specialized that it can only be applied to problems that have access to the expected domain knowledge, since the distance calculation requires it. Our goal has been to create algorithms that are capable of taking advantage of the information that is present without making assumptions about what domain knowledge will be present. Our algorithms remain general but are able to specialize using whatever knowledge is provided to them.

Modifying the Objective Function. The fourth category of methods that incorporate neighborhood information when clustering are those which modify the objective function itself. Usually, this means replacing the objective function with a weighted sum of the original objective function and the supplemental neighborhood (contiguity) information. Murtagh (1985) notes that, in general, attempting to optimize both *compactness* (variance) and *contiguity* requires that the user specify the relative importance of the two objectives via a weighting parameter. For example, Theiler and Gisler (1997) combine variance and contiguity via a user-specified parameter $\lambda \in [0, 1]$. We will compare our soft constrained clustering approach to their contig-k-means algorithm in more detail in Chapter 6. The most significant difference between their work and ours is that our algorithms allow for the specification of a weighting factor (strength) for *each* constraint. Although contiguity is important, it does not apply equally to the entire image. If it did, the best solution would be one that assigns every pixel to the same cluster. In fact, certain regions should have more contiguity than others; these correspond to the final clusters in the image. If some initial knowledge about regions in the image is available, we have the ability to specify that the contiguity constraint is stronger for the coherent regions than for other places in the image. Finally, we do not restrict ourselves to contiguity constraints alone. We can accommodate multiple sources of constraining information using a single representation.

Ambroise et al. (1997) introduce an EM-based method for incorporating contiguity constraints. Their approach penalizes a potential solution according to how discontiguous the solution is. Once again, there is only a single weighting factor that can be specified, and contiguity information is the only form of domain knowledge that can be accommodated.

Directly Restricting $\Phi(D)$. The final method for incorporating neighborhood information in clustering is to impose explicit constraints on $\Phi(D)$, the set of allowable partitions. This is usually referred to as *contiguity-constrained clustering* (Gordon, 1973; Murtagh, 1985; Gordon, 1996; Bachar and Lerman, 1998), *relational clustering* (Ferligoj and Batagelj, 1982, 1983), or *conditional clustering* (Lefkovitch, 1980).

The output clusters, in this formulation, must preserve the structure given in an input relation R (i.e., $R = Con$). The set $\Phi(D)$ is thus restricted to solutions that satisfy R. For each partition $P \in \Phi(D)$, each $C \in P$ must be a valid subgraph such that $(V, E) = (C,\ R \cap (C \times C))$. This is accomplished in agglomerative algorithms (e.g., single-link and complete-link) by restricting which cluster pairs can be considered for merging at each step. Batagelj and Ferligoj (1998) describe a method for satisfying relation R in an iterative relocation algorithm.

Relation R is treated as a set of *soft* constraints; not all links in R must be preserved in the final result. However, the absence of a link between a pair in $C \times C$ is a *hard* constraint. If R in the image segmentation example (Figure 2.1b) included a link between pixels 21 and 5, a two-cluster solution suddenly would be possible. Otherwise, the two dark regions cannot be merged without also merging the intervening white pixels.

Another method is to construct the minimum spanning tree over the data set (Murtagh, 1985), where nodes are the data points and edges indicate that two points are contiguous. The edges are weighted by the distance (in terms of their feature vectors) between the two items. This method differs from the general MST methods mentioned above (Zahn, 1971; Urquhart, 1982). Those methods include edges between all pairs of items; Murtagh creates edges between neighbors only. Therefore, the set of possible solutions is automatically restricted to partitions with contiguous clusters.

Contiguity-constrained clustering requires that the relation R be complete, i.e., the induced graph is connected. There are two common ways to handle the disconnected case. If contiguity is defined in terms of a maximum threshold (beyond which two items are not considered contiguous), one can iteratively grow this threshold until a solution is obtained that is fully connected (Perruchet, 1983). Another option is to separately analyze each connected component (Ferligoj and Batagelj, 1983; Murtagh, 1985), since as above, it is assumed that the lack of a link between two components prevents them from being clustered together. While this may in some cases be a valid assumption (as in the neighborhood relation for image segmentation), for general relations it will not be. Incomplete knowledge of the relation R is one possibility. In addition, some relations may only be defined over a subset of the instances. Also, clusters *must* be contiguous, which means that they cannot be composed of two separate contiguous regions. This is reasonable when segmenting an image into distinct objects, but problematic for other spatial applications, such as a geological analysis of a satellite image, where the same minerals may occur at different places in the image.

In contrast, our approach does not require that Con be complete nor that the resulting clusters be contiguous. We can handle an incomplete subset of the possible constraints without difficulty. Our methods will enforce the contiguity constraint at the strength that is specified.

2.3.2 General Relations

Contiguity-constrained clustering has been heavily investigated because of the volume of image analysis work being done (e.g., image segmentation, satellite image analysis, remote sensing of the Earth and other planets). However, neighborhood relations capture only one kind of domain knowledge for clustering. There are other kinds of global relational information that are important for clustering.

Ferligoj and Batagelj (1983) extend the notion of relational constraints to apply to asymmetric relations. However, their method appears to apply only to transitive relations since it computes a transitive closure of R. More importantly, they restrict their attention to relations that indicate "sameness." For example, their method fails if $(x, y) \in R$ means that x and y *should not* be clustered together; this is a valid relation, but one that does not indicate "sameness" and is not transitive.

The incorporation of general relations into clustering algorithms has not been well investigated. Our work in future chapters will show that real-world problems require richer relations to express relevant domain knowledge. Problems often require the encoding of relational information that only applies to a subset of the instances or is inherently intransitive. Consider the hypothetical problem of assigning diners to tables at a wedding reception. If phrased as a clustering problem, the diners are the data items and the table groups are the clusters. The event organizers may know that certain individuals do not want to be seated at the same table. This relation is incomplete in that it (presumably) applies only to a subset of the diners. It is also intransitive; the fact that person A does not want to sit with person B and person B does not want to sit with person C does not imply anything about the degree of friendship between A and C. This problem cannot be handled by the previously described constrained clustering methods; they are designed with other constraints in mind. However, our methods can accommodate this kind of relational, incomplete, intransitive information and enforce the specified constraints.

2.4 Cluster-level Constraints: Capacity Constraints

Domain knowledge in Con may also take the form of information that applies to individual clusters. *Cluster-level* constraints impose requirements on the shape, size, orientation, or other features of the clusters themselves. The most common type of cluster-level constraint is a constraint on the minimum or maximum cluster capacity. Cluster-level constraints do not fit well into our constraint formulation, and we will not devote much space to discussing them. Methods that can accommodate capacity constraints are included in this section because they represent an interesting set of methods that are complementary to ours.

Minimum capacity constraints. Bradley et al. (2000) developed a method which makes use of cluster-level constraints to avoid solutions with empty clusters, which often occurs when using a k-means algorithm, especially in a high-dimensional space when k is large. This approach imposes constraints on cluster structure. They allow the specification of a minimum number of items for each cluster. In addition to avoiding solutions with empty clusters, the authors also demonstrate that the modified algorithm is less prone to local minima than the traditional k-means algorithm.

In cases where the problem domain dictates a minimum value for cluster size, this is a valuable approach. However, although they allow mathematically for a different minimum value for each cluster, it is not clear how the assignment of different values would be done.

Initial cluster centers (which determine what the identity of each cluster will be) are generally chosen randomly.

Tung et al. (2001) refer to minimum capacity constraints as *existential* constraints and incorporate them into an iterative relocation algorithm by restricting the allowable item movements. This prevents the existential constraint from ever being violated. In addition, as noted above, sequential *region growing* algorithms naturally accommodate capacity constraints.

Maximum capacity constraints. Also, some methods make use of a *maximum* cluster capacity. Murtagh (1985) suggests that this kind of constraint can be satisfied by using a hierarchical clustering algorithm and then selecting "suitable" clusters from the hierarchy. However, a single hierarchy will not include every possible cluster, and there may not be one available of the appropriate capacity. This kind of constraint also appears often in research on the *facility location problem* (Shmoys et al., 1997), where k facilities must be selected to service a set of customers but each facility can handle a maximum of c customers.

2.5 Feature-level Constraints: Heuristic Rules

Domain knowledge may also be information that depends on the features of the data set. *Feature-level* constraints provide guidance for the placement of items based on their values for a given feature.

A constrained batch hierarchical algorithm that incorporates feature-level constraints was developed by Béjar and Cortés (1998). This algorithm takes in a set of rules that test for specific feature values and predict the appropriate class. Items that have the same predicted class are assigned to the same cluster. Talavera and Béjar (1999) developed a similar algorithm which expands on the work of Béjar and Cortés by allowing a set of instances to be grouped together even when the appropriate class label is not known.

This system relies on user input to operate: the user selects which partition in the hierarchy to use as the derived partition. Thus, the user determines, perhaps inadvertently, which constraints are actually reflected in the result. If the partition is chosen below the point where a set of constrained instances are merged together, then this constraint will not be satisfied. However, in our formulation, the set of constraints that must be satisfied is explicitly encoded in Con, so unsatisfied constraints are not a concern. Further, we support knowledge that indicates when items should not be grouped together, whereas these constrained methods do not.

2.6 Instance-level Constraints

The final form of domain knowledge that we will examine is at the most specific level. *Instance-level* constraints place restrictions on individual pairs of items with regards to their relative cluster membership. They may take the form of partial labels on the data set, user feedback in interactive systems, or direct constraints on the relative placement of item pairs. The existing work in this category is the most relevant to the our algorithms, because we have focused on enabling the use of instance-level constraints.

2.6.1 Partial Labels

Although labeling an entire data set is often infeasible, it is often quite practical to request that a small subset be labeled. If this information is available, it functions as a valuable source of information. When processing a *partially labeled* data set, our methods convert the existing labels into a set of Con relationships.

Majority vote based on labeled members. There is a lot of work currently being done to investigate the best ways to make use of combined labeled and unlabeled data. For example, the Neural Information Processing Systems conference has recently held two Unlabeled Data Supervised Learning Competitions, in 2000 and 2001. These competitions provided data sets and an evaluation system to enable the direct comparison of a variety of methods that are able to combine labeled and unlabeled data. Although the emphasis was on supervised methods that are able to take advantage of supplemental unlabeled data sets, there are also complementary unsupervised methods that can make use of the labeled data.

One approach that makes use of both kinds of data with a clustering algorithm is described by Demiriz et al. (1999). They cluster both data sets together using a modified k-means algorithm, and each cluster is assigned a label by majority vote based on its labeled members. The algorithm then tries to minimize, over a range of k values, both *dispersion* (e.g., mean square error) and *impurity* (e.g., Gini index). Minimizing dispersion leads to a preference for fewer (larger) clusters, while minimizing impurity leads to a preference for a larger number of more specific (smaller) clusters. The labels that are available on some of the instances act as constraints on the clustering process via the impurity measure. This approach combines the strengths of supervised and unsupervised learning, but some details are unclear. For example, how is a cluster label assigned if none of its members are labeled? What proportion of instances should be labeled for this method to be effective? Rather than attempting to assign a label to each cluster, we simply enforce the specified labels so that items with identical labels end up in the same cluster, and items with different labels are placed into different clusters. Unlabeled items are clustered normally.

Seeding clusters based on labels. Basu et al. (2002) suggest a different use of data labels. Rather than using the labels to determine the identity of a cluster, *seeded clustering* methods use them to intelligently select the initial cluster centers for the k-means algorithm. For each group of items with the same label, they select the mean of that group as one of the initial cluster centers. Each item in the labeled subset of data is assigned to the cluster center that matches its label. Basu et al. propose two modified k-means algorithms. One algorithm uses the cluster centers as an initial starting point only and is free to modify the cluster assignments for each item in later iterations. The other algorithm enforces the initial assignment of every labeled item throughout the clustering process, updating the unlabeled items only.

Seeded clustering is similar to the methods we propose. The most significant difference is that the methods of Basu et al. are restricted to domain knowledge that can successfully be encoded as individual data labels, while our methods encompass a far wider range of information. For example, assume that we know that the items in a subset A of the data set are similar enough that they should all obtain the same label. Also, assume that we have the same knowledge about a second group of items B, but that we do not know how the two groups are related. That is, we do not know whether they are two distinct groups or whether they should be merged into one larger group. In the formulation of Basu et al., we cannot abstain from that decision. Their encoding requires that we either assign every item in A one label and every item in B a second label, in which case they will be treated as distinct clusters, or assign them all the same label, in which case they will all be merged together. In contrast, our methods do allow for the specification that certain items should be grouped together but that the relationships between others are not known. This is possible because our constraints are expressed as pairwise relationships rather than labels. Conversely, if the information available is actually encoded as labels, we can easily convert this into pairwise constraints. Therefore, our constraint formulation is more general and can accommodate a wider range of domain knowledge than that of Basu et al. We will report on an empirical comparison of our methods to theirs in Section 3.6.

2.6.2 User Feedback

Information about individual items in the data set may come in other forms than class labels. Interactive clustering systems adopt an iterative approach, where the system produces a partition of the data and then presents it to the user for evaluation. The user can indicate where the system has made mistakes, and that information can be used on the next iteration of the algorithm. One example of such an interactive clustering system is developed by Cohn et al. (2000). Like *active learning* approaches (Cohn et al., 1994), where an algorithm requests labels for the items it deems most informative, this system is able to receive more precise guidance towards the desired solution. This can reduce the amount of information that must be specified, since any information the algorithm is able to deduce on its own need not be explicitly encoded. Beyond simply enforcing the information specified by the user, the interactive clustering system of Cohn et al. also attempts to induce a better distance metric over the feature space. The system can automatically stretch or shrink the apparent distances between items according to the constraints provided by the user. This ability can be very helpful for users who cannot or do not wish to develop their own, domain-specific distance measure.

However, this kind of distance metric induction may not be appropriate for all domains. We will discuss this issue further in the next section. In addition, it may be easier for a user to specify up front all of the information they wish the system to use while clustering, rather than interacting with the system after each iteration. The relative tradeoffs are best evaluated on a per-problem basis. Finally, scaling is an issue: the larger the data set is, the more difficult it may be for the user to identify and correct clustering mistakes.

2.6.3 Pairwise Relationships

We now describe our chosen representation for domain knowledge and explain why we believe it to be the most general and flexible representation available. We will also contrast our method to the most relevant alternatives.

We believe that the most general way to represent relational information over the data set is as a collection of *pairwise links* between items. This representation of domain knowledge covers several of the other knowledge sources we have already discussed. Global constraints that define a neighborhood relation are easily encoded as a set of pairwise links between items. Relations that are incomplete or intransitive are also naturally expressed in this format. Feature-level constraints often indicate whether two items should or should not be clustered together, which allows us to encode them as pairwise links as well. The only kind of constraint that is not easily represented as pairwise links is the cluster-level constraint. In this dissertation, we will reserve an investigation of cluster-level constraints for future work.

Due to the generality it provides, we focus on the instance-level pairwise expression of domain knowledge. Commonly, two kinds of constraints are used: must-link and cannot-link. This formulation has been explored in two main ways. The first is to interpret each pairwise constraint as an indicator that the entire feature space should be stretched (compressed) to accommodate a cannot-link (must-link) constraint (Zhu et al., 2001; Klein et al., 2002). The second option is to treat each constraint as an isolated statement about the two items involved (Kleinberg and Tardos, 1999; Wagstaff and Cardie, 2000; Wagstaff et al., 2001). Deciding which approach to use depends on what kind of knowledge the constraints encode. The specified constraints can be viewed as *hard* or *soft* constraints. We next discuss work with both kinds of constraints.

Hard constraints. Hard constraints are declarative statements that must be satisfied by the output of the clustering algorithm. A must-link constraint on a pair of items indicates that the output partition *must* place those items in the same cluster. A cannot-link constraint indicates that the output partition *cannot* place those items in the same cluster.

In the context of hard constraints, Klein et al. (2002) compare our constrained k-means intelligent clustering algorithm to a new approach: rather than modifying the clustering algorithm to accommodate constraints, they pre-process the data set to modify the pairwise distances between items. The distance between items that must-link is set to zero, and the distance between items that cannot-link is set to the maximum pairwise distance for that data set plus one. Constraint information thus propagates throughout the data set. They use a complete-link agglomerative method on the transformed pairwise distances to cluster the items. They demonstrate that, for problems where clusters have shapes that are difficult for the regular k-means algorithm to detect, the complete-link algorithm operating on the transformed distances can achieve higher performance then our constrained k-means algorithm. They also achieve much higher performance than our methods on several real-world data sets. In addition, for some data sets, their approach requires less than half of the number of constraints that COP-KMEANS does, for the same level of accuracy. Because they extend constraint information through the feature space, less information must be explicitly encoded in the constraint set.

There is a subtle, but important, distinction between our methods. Klein et al. created an approach that takes in a handful of constraints and deduces the feature-level implications of those constraints. They have shown the benefits of this approach on several data sets, where unconstrained items benefit from constraints placed on very similar items. However, what happens when items that appear very similar are in fact from different classes? In such a case, the constraint propagation may unwittingly make mistakes by forcing those items into the same class.

At first glance, this situation seems unlikely. It indicates that the features being used to represent items are not very consistent with the true classes of the items. In inductive learning, the goal is usually to use features that are relevant to the problem and whose values correlate well with the true class labels. Unfortunately, it is not always possible to determine which features are the best ones to use or even whether a given feature is truly relevant to the target categorization. Alternatively, perfectly correlated features may be too expensive to compute or simply not available at all. If feature selection could be done perfectly, there would be no need for additional domain knowledge; the correct clustering would fall out immediately from the feature value distribution. Therefore, in practice, we may often face situations where the feature values are not reliable indicators of class membership.

This observation motivates our model of domain knowledge. Our encoding and use of constraint information can accommodate knowledge that may not be consistent with the feature values used to represent the data. Like Abu-Mostafa (1995), we specifically seek to accommodate information that cannot be deduced from the feature values alone. The tic-tac-toe data set, which we will present in Chapter 3, is one such example; its features do not correlate well with the class labels. Despite its difficulty, we are able to achieve large increases in clustering accuracy on this problem. The results of running Klein et al.'s algorithm on this data set are not yet available, but we believe that further insights could be drawn from this comparison.

Soft constraints. Soft constraints, or *preferences*, are statements that may or may not be satisfied by the clustering algorithm. We can be precise about how likely the constraints are to be satisfied by indicating a strength for each constraint, or we can treat them all as equally flexible preferences.

We have adopted the first approach. We posit that pairwise constraints can be made more expressive by augmenting each one with a *strength* that indicates how binding the constraint is. This strength varies from -1 to 1, where 1 indicates a hard must-link constraint, -1 indicates a hard cannot-link constraint, and 0 is a "don't care" statement. Strength values between the extremes are helpful to express heuristic or approximate domain knowledge; these constraints function as *preferences* on the clustering outcome. We refer to a constraint with a strength greater than 0 as a *positive preference* and a constraint with a strength less than 0 as a *negative preference*. We will present our handling of soft constraints in more detail in Chapter 6.

Kleinberg and Tardos (1999) also allow for the specification of an individual strength for each constraint. They are concerned with the *metric labeling problem*, which is equivalent to our above description of the constrained clustering problem. Their formulation allows the specification of instance-level constraints between pairs of items. Each constraint indicates that the two items should receive the same label (be placed into the same cluster) according to the constraint's *weight*. They incorporate the constraints by calculating a penalty proportional to the weights of the violated constraints. By phrasing the label assignment problem as a linear programming problem, they are able to obtain a solution that is guaranteed to be within a factor of two of the optimal solution.

Our work differs from that of Kleinberg and Tardos both in the kind of constraints that are permitted and in the methods that are used to obtain a solution. We can accommodate both positive and negative preferences, while their work is restricted to positive preferences only. Their constraints must have nonnegative weights; there is no facility for including constraints that indicate when two items should not receive the same label. It would be instructive to determine whether the linear programming approach could be extended to include constraints of this nature. In addition, the linear programming method they use to obtain a solution differs greatly from the two constrained clustering methods that we will present. Their approach is valuable since it provides an approximation guarantee, which k-means and COBWEB (the two algorithms we have modified) do not. Our methods are useful because they are much less computationally expensive than the linear programming approach.

Zhu et al. (2001) adopt a different approach that treats all of the constraints as soft constraints that may or may not be satisfied by the output of the clustering algorithm. They allow for three kinds of constraints: must-link, cannot-link, and "this pair is closer together than that pair." Like the interactive clustering approach of Cohn et al. (2000), their goal is to induce an appropriate distance measure on the data set. The constraints are not annotated with a strength or weight; instead, they are given to an iterative method that determines appropriate weights, one per feature, to (linearly) transform the data set into D'. Once again, this process stretches or compresses the feature space appropriately, so that must-link pairs are brought closer together and cannot-link pairs are pushed farther apart. They then make use of a standard clustering algorithm to cluster D', which is more likely to enforce the specified preferences than clustering with the original D would be. Their approach is very similar to that of Klein et al.; the major difference is that Zhu et al. do not *require* that the specified constraints be satisfied, so they are treated as preferences rather than constraints. In addition, the warping of the feature space performed by Klein et al. is not necessarily a linear transformation from the original feature space.

In contrast to the method used by Zhu et al., our methods will ensure that the soft constraints are enforced, to the degree of their individual strengths. This gives us much more precise control over how the constraints are interpreted and applied by the clustering algorithm. Information we are very confident about can be specified with a high strength, while information that is less reliable can be specified with a low strength. Our soft constraint formulation is therefore more expressive and more exact than that used by Zhu et al. In addition, because we do not modify the feature values for the items in the data set, the resulting clusters are more easily interpreted, since they exist in the same feature space as the original items.

Another important difference between our work and that of Zhu et al. also arose in our discussion of Klein et al.'s work. When modifying the feature space, a single pairwise constraint between two items will also affect every other pair of items. Our work enforces very local warpings of the feature space, by requiring that pairwise constraints be enforced; the constraints do not propagate to other items unless there is a transitive link in the constraint relation. In contrast, both Zhu et al. and Klein et al. extend each pairwise constraint (in different ways) through the feature space. In some cases this may be appropriate, given the interpretation of the knowledge encoded in the constraint. It is our opinion that it is preferable to adopt a more conservative approach and to assume that each specified constraint applies only to the two items explicitly mentioned. If it is known that similarity in the feature space correlates well with similarity in terms of data labels, then one of the distance-warping methods may be of most benefit, since they will automatically propagate constraint information.

2.7 Summary

This chapter has presented an overview of both general clustering algorithms and the work that has been done to enhance those algorithms with additional domain knowledge. This knowledge functions as a set of constraints on the clustering process. We identified four main sources of knowledge, including global constraints, cluster-level constraints, feature-level constraints, and instance-level constraints. We discussed the advantages and disadvantages of relevant research in each category.

Our focus on instance-level constraints is due to the flexibility that encoding provides: we can encode all types of constraints, except cluster-level constraints, in this fashion. It also allows us to express information about incomplete, intransitive, or otherwise "difficult" relations among items in the data set. In the next chapter, we will discuss in more detail our proposed method and how we represent hard constraints at the instance level. Soft constraints will be explored more fully in Chapter 6.