INTELLIGENT CLUSTERING WITH INSTANCE-LEVEL CONSTRAINTS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Kiri Lou Wagstaff

August 2002

INTELLIGENT CLUSTERING WITH INSTANCE-LEVEL CONSTRAINTS

Kiri Lou Wagstaff, Ph.D.
Cornell University 2002

One goal of research in artificial intelligence is to automate tasks that currently require human expertise; this automation is important because it saves time and brings problems that were previously too large to be solved into the feasible domain. Data analysis, or the ability to identify meaningful patterns and trends in large volumes of data, is an important task that falls into this category. Clustering algorithms are a particularly useful group of data analysis tools. These methods are used, for example, to analyze satellite images of the Earth to identify and categorize different land and foliage types or to analyze telescopic observations to determine what distinct types of astronomical bodies exist and to categorize each observation. However, most existing clustering methods apply general similarity techniques rather than making use of problem-specific information.

This dissertation first presents a novel method for converting existing clustering algorithms into *constrained clustering* algorithms. The resulting methods are able to accept domain-specific information in the form of constraints on the output clusters. At the most general level, each constraint is an instance-level statement about a pair of items in the data set that indicates a preference for being placed into the same cluster, or, alternatively, into different clusters. The constrained clustering algorithms developed and presented in this dissertation enforce each constraint according to the strength of that preference.

The second major contribution of this dissertation is the application of constrained clustering algorithms to diverse, significant, challenging real-world problems. We observe that the additional domain knowledge, when combined with the algorithms' ability to enforce that knowledge, produces improvements on a variety of tasks. The problem domains include automated map refinement, natural language processing, and automated data analysis of Hubble Space Telescope observations.

BIOGRAPHICAL SKETCH

Kiri Lou Wagstaff grew up in the small community of Castle Valley, Utah, amid the red rocks, desert sage, and wild freedom from the press of humanity. Her first computer was a Commodore Plus/4, a magical toy that utterly transformed her 11-year-old existence. After graduating from Grand County High School in 1993, Kiri spent four years in the Computer Science department at the University of Utah, earning her B.S. Along the way she developed an intense interest in artificial intelligence. The best of the AI topics, in her view, had to be machine learning - AI programs that could actually get better over time!

Kiri arrived at Cornell University in 1997 and continued in her pursuit of knowledge, getting heavily involved in natural language processing and ballroom dancing. In the summer of 1999, she worked as an intern at the Jet Propulsion Laboratory in Pasadena, California, and the following summer was spent at DaimlerChrysler's Research and Technology Center in Palo Alto, California. She spent her fourth and most productive year research-wise in absentia in Berkeley, California. In the spring of 2002, she put aside all of her extra-curricular interests and settled down to write the manuscript you now have before you. In August, she will start work as a researcher at the Johns Hopkins University Applied Physics Lab, in the Space Department. She looks forward eagerly to the chance to apply her research ideas to support space missions, spacecraft, and the analysis of data collected by their instruments.

ACKNOWLEDGEMENTS

First and foremost, I thank my advisor, Dr. Claire Cardie, for all of her guidance, patience, and instruction in the ways of the research world. Claire is a truly gifted technical writer, and I continue to strive to live up to her high standards. I would like to particularly thank my committee members, Dr. Jim Bell, Dr. Marie desJardins, and Dr. Bart Selman, for seeing me through this long journey and for offering good advice at critical points along the way. Several others contributed to this work. Sugato Basu generously contributed the graphs in Section 3.6 and several useful email exchanges. Dr. Seth Rogers and Dr. Stefan Schroedl collaborated with me on the road map refinement problem. Dr. Jim Bell worked closely with me on the analysis of STIS Mars observations, supplying the data set, offering advice and encouragement, and interpreting results, despite his hectic schedule. I would also like to thank Jascha Sohl-Dickstein, Jason Soderblom, and Eldar Dobrea for their assistance with the STIS data sets and general good cheer.

My informal advisors and mentors have been invaluable. Dr. Pat Langley has been a reliable collaborator and friend since I asked him for a summer job in the spring of 2000. Dr. Dennis DeCoste was an excellent supervisor at JPL and quite exhilarating to work with. Dr. Elaine Weyuker has been my mentor and frequent email correspondant since 1998 and repeatedly offered me priceless advice. I would also like to thank Dr. Chris Johnson at the University of Utah for being an inspiration and a role model. Even more significantly, I thank my parents, David and Lois, for their lifelong guidance and steadfast support in my pursuit of the Ph.D.

Many of my friends have contributed in significant ways to the completion of this work. Dr. Amy McGovern was my thesis buddy and offered a slew of critical insights on the initial drafts of each chapter. This dissertation would truly suffer in the absence of her input. Although she beat me to the title, she won't beat me to Mars! David Kempe shed brilliant light on the formal analysis of my algorithms and tactfully pointed out my errors. Lengthy discussions with Dan Klein about the nature of constrained clustering made me confront several hard questions and to emerge the better for it. Jeff Vinocur presented me with a very thorough proofreading of a nearly-final draft, and his suggestions helped add the final polish it needed. Andy Sfekas and Dr. Kim Cleland were the best friends I could hope for during my final year. They cheered me on in my job interviews and at my defense and didn't complain (too much) when I couldn't come out to play because I had to write. Westley Weimer, friend and aite, was there from the beginning. His critiques and suggestions added significantly to my research progress and the quality of the implementation. His friendship and sense of humor carried me through several difficult times.

There are several institutions that have contributed financially to the research that produced this dissertation. I would like to thank the National Science Foundation for three years of support as a Graduate Research Fellow. I would also like to thank NSF Grant IRI–9624639 and DARPA TIDES Grant N66001-00-C-8009. I also thank the Jet Propulsion Laboratory and DaimlerChrysler's Research and Technology Center for exceedingly productive summer internships.

Finally, I would like to thank astra, stardust, schneller, zinger, and the CFS linux cluster for their tireless dedication to helping me develop code and to running my clustering experiments. In addition, many thanks go to nova, astra's faithful printer sidekick.

ad astra

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Unsupervised learning algorithms have had some impressive successes. Data mining algorithms, for example, are regularly used in the corporate world to extract useful information from large customer databases. Rather than being restricted to the realm of academic research, algorithms from machine learning are now showing up in commercial software, including the PolyAnalyst software offered by Megaputer and Genio Miner from Hummingbird. In addition to being used as commercial tools, learning methods have been used to extend the body of scientific knowledge. Clustering algorithms, which automatically divide a data set into meaningful sub-groups, have been especially successful in both areas. For example, the Autoclass clustering program analyzed a large body of infrared spectral data and discovered a sub-class of stars previously unknown to astronomers (Goebel et al., 1989).

However, the majority of clustering algorithms are limited in what they can achieve. They can detect general trends and patterns in data, but they cannot make use of additional knowledge specific to the problem at hand, as a human expert can. This dissertation develops and evaluates a new kind of clustering algorithm that can take advantage of problem-specific information to become a temporary expert in the domain at hand. In particular, these "intelligent clustering" methods make use of information expressed as constraints between two items in the data set. These algorithms, although general in nature, are able to greatly enhance their performance on individual tasks.

In this chapter, we first discuss unsupervised learning and describe how it is used. We then contrast it with supervised learning methods and identify some important limitations of both approaches. Next we outline our proposed method of intelligent clustering, which falls somewhere in between these two extremes and seeks to address their respective limitations. After describing several scenarios that stand to benefit from our method, we conclude by summarizing the key contributions of our work.

1.1 Unsupervised Learning

We often think of learning as a product of instruction. A student learns by observing a teacher or by carefully examining correct answers. We refer to this kind of situation as *supervised* learning. In contrast, *unsupervised* learners have neither an instructor nor a set of correct answers. Unsupervised learning is exploratory and experimental, seeking knowledge through discovery.

Clustering algorithms are unsupervised in that they do not have access to any specific information about what they should be seeking. A clustering algorithm uses the same principles to perform data analysis whether the objects under scrutiny are stars, or apples, or human beings. Success hinges on how well a given data set conforms to the assumptions encoded in those generic principles. Although currently in widespread use with many different kinds of data, clustering algorithms adopt no special provisions to account for particular characteristics of the data set or problem domain.

Exploratory learning via built-in bias. All unsupervised learning is exploratory in nature; the learner is presented with a set of observations and then encouraged to discover patterns and meaningful structure independently. This is akin to how we learn to interpret visual images or make summaries of long text passages: we do not receive specific guidance that tells us "this is a human face" or "this newspaper article should be summarized with the following three sentences," but we are guided by general notions of pattern matching and simplification, respectively. Similarly, a computer can be instructed to detect patterns in data and to group items by similarity. Despite the lack of specific guidance, unsupervised approaches cannot be said to be completely ignorant. Just as a human makes use of general principles to enable learning, a computer makes use of a *learning bias* of some sort, which encompasses any assumptions it makes about what is to be learned (Mitchell, 1997). For a clustering algorithm, this bias might be a preference for identifying groups of items that are compact and well-separated from other groups. However, we claim that, for many situations, the built-in bias of clustering algorithms is either insufficiently specific or actively misleading.

Clustering as a data mining tool. This limitation is significant because clustering algorithms are currently being used for a variety of purposes. Unsupervised data analysis, or *data mining*, is becoming more and more prevalent as organizations continue to create and maintain vast databases. Corporations stockpile information about their customers; scientists collect gigabytes of observational data. In both cases, "mining" these databases for recurring patterns or other interesting structure is very important. Clustering is even being used to detect patterns in reported computer crimes, so that sites likely to be future targets can be warned (Brown and Gunderson, 2001). Researchers at Stanford University estimate that the worldwide production of data in electronic format is between 500,000 and 1,700,000 *terabytes* per year (Lyman and Varian, 2001). This implies several thousand new terabytes of data every day. The human data analyst is overwhelmed by the sheer volume of information, yet extracting trends and classifications from the data is critical. Currently, clustering algorithms for data mining are successfully culling useful information from these large databases, but as above, standard approaches disregard any domain-specific information.

Clustering as a data compression tool. An additional application of clustering algorithms is *data compression*, where massive data sets are reduced to more manageable sizes by replacing a group of similar items with a single representative data point. This alleviates the intense demand for data storage and transmission resources. For example, probes and rovers operating remotely on other planets or in deep space are capable of collecting large amounts of data, but are limited by size and bandwidth in terms of how much they can store or transmit back. If we can only obtain a fraction of the full data set, it is preferable to receive information about a carefully selected subset of the most interesting items, rather than simply the first ones that are lined up in the queue (Mjolsness and DeCoste, 2001). Clustering algorithms can perform this kind of filtering to make better use of limited resources such as bandwidth or storage space. *Intelligent* clustering algorithms can use domain knowledge to make better decisions about which items are redundant (i.e., should be grouped together) and which are significantly different.

Disadvantages of unsupervised learning. Despite the many applications of clustering algorithms, their very generality also limits their performance on any specific task. Because they make use of very general notions to identify patterns and interesting structure in data, they cannot specialize very well for a specific problem. Further, because unsupervised methods are unguided, they may report patterns or trends that, while present in the data, are completely uninteresting (e.g., a pattern caused by a systematic error in the data gathering process). The traditional alternative to unsupervised learning has been supervised learning, which adapts to the problem at hand by not only exploiting but *requiring* problem-specific knowledge.

1.2 Supervised Learning

Supervised algorithms differ from unsupervised algorithms in that they depend on having a set of labeled examples that demonstrate what kind of distinctions should be made. The labels indicate the correct answer for each of a set of sample problems. Testing what has been learned then involves presenting the learner with additional, unlabeled, problems and comparing the learner's responses to the correct answers. This is not so different from how teaching and testing of humans often occur in our classrooms; we seem to be very good at learning by example.

Disadvantages of supervised learning. Despite the proven success of supervised approaches to learning, they too have certain limitations and drawbacks. A supervised algorithm performs best when a large body of labeled examples is available for the algorithm to learn from. The algorithm is able to train itself to generate the correct responses, according to the labeled examples, and (hopefully) to also generalize to unseen data. But as the adage "garbage in, garbage out" reminds us, supervised algorithms are limited by the quality of the labeled examples they are given. In addition, they may be susceptible to over-fitting the training data: an algorithm that focuses too closely on getting all of the given examples right may then be incapable of performing well on new, different examples.

The most important drawback to supervised learning is that an expert must manually create a sufficiently large set of labeled examples. Supervised learning is an all-or-nothing paradigm; it cannot make use of any unlabeled data. Unfortunately, individual labeling is a time-consuming and tedious process. Further, in some cases it may not be possible to provide labels at all. For some problems, the true answers are not known, and we would like the computer to automatically discover interesting patterns and classifications on its own. Supervised methods cannot provide solutions to these problems.

Finally, it is not possible to select one learning paradigm as the universal best approach. Supervised and unsupervised methods serve different learning goals. Supervised methods are appropriate when the goal is to build a predictive model (e.g., neural network, decision tree, support vector machine) for the problem domain. Unsupervised approaches are used when the goal is to analyze the data to discover interesting structure (e.g., clusters or association rules). Our focus is on the latter type of problem.

Table 1.1: Census data represented in feature-vector format

Name	SSN	Age	Gender	Occupation	Income
Diana	111-11-1111	30	F	Professional Dancer	$50,000
Gary	444-44-4444	30	M	Professional Dancer	$50,000
Lyra	222-22-2222	29	F	Rocket Scientist	$90,000
Tim	777-77-7777	28	M	Assistant Professor	$70,000
...	...	...	...	...	...

1.3 Intelligent Clustering

We are faced with two extremes. Supervised learning requires that every item be assigned a label. Unsupervised learning ignores any such guidance. What if we have a data set where some, but not all, items are labeled? What if we have additional information about the data set that cannot be expressed as individual labels? In the absence of other alternatives, we are forced to use an unsupervised approach and to discard all of this extra information.

However, deliberately ignoring information is contrary to what we see as the basic philosophy of learning. Successful learning is opportunistic, and it attempts to exploit information from as many sources as possible. Some algorithms, referred to as *weakly supervised* or *semi-supervised* methods, seek to limit the amount of manual labeling that must be done yet still make use of domain-specific information when it is available. We extend this body of hybrid algorithms by proposing the use of "intelligent clustering" algorithms. In this section, we identify four situations where these algorithms, which can take advantage of partially labeled data and information that transcends the label format, are particularly useful.

1.3.1 Knowledge Not Expressed as Labels

The usual division between supervised and unsupervised learning recognizes only one source of information: individual data labels. However, as hinted at above, there are many other forms that knowledge can take. It may be presented as structural or spatial information about the problem, heuristic guides, relational information between items, or some combination of these elements. In recognition of this variety, the intelligent clustering algorithms we propose are able to exploit any such information that can be expressed as a set of constraints between items in the data set.

Information that describes a relationship between items in a data set has proven particularly challenging for machine learning applications because it cannot be represented in the usual data format. The most common way to represent items in a data set is with a *feature vector*. Each item possesses a value for each of a given set of features, and those values can be strung together to create a feature vector. For example, consider a database of census records, as shown in Table 1.1. Each person is represented by a variety of features, including social security number, age, gender, occupation, and annual income.

However, although it is possible to describe an item with a series of features, it is not possible to use those features to express a relationship between two items. For example, we cannot use a feature to capture the fact that Gary and Diana are married to each other, because this requires a way to express relationship information *between* items (people). This representational issue is a problem because a learning algorithm cannot make use of information in formats it does not understand.

We propose a method for representing instance-level relationships and for naturally integrating those relationships into clustering algorithms. In particular, we focus on relationships that are consistent with the desired clustering output, and describe how these can be expressed as a set of instance-level constraints. To continue the example, we may want clusters of people to represent households or neighborhoods, and therefore desire that clusters preserve the marriage relationship. We will also show how this formulation is a convenient way to express useful background knowledge about a variety of real-world problems.

1.3.2 Partially Labeled Data Sets

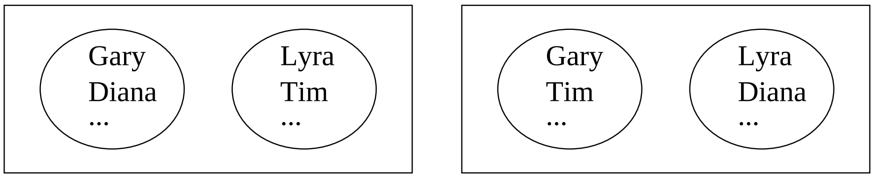
Section 1.2 described supervised learning and hinted at some reasons why it might not be possible (or feasible) to provide individual labels for every item in a data set. This becomes more and more common as researchers encounter data sets with thousands or millions of elements; in such cases, manual labeling is simply unrealistic. While supervised methods can restrict themselves to the subset of data that does possess labels, the unlabeled items also contain information that can contribute to better system performance. Significant work has been done in modifying supervised algorithms to enable them to operate on data that is not fully labeled (e.g., Blum and Mitchell, 1998; Nigam et al., 1998) However, less work has been done in enabling unsupervised algorithms to exploit partially labeled data sets.

We will show how to take existing labels, convert them into a set of instance-level constraints, and then ensure that they are preserved in the final output of a clustering algorithm. Our approach can therefore integrate labels, traditionally only useful for supervised learning, into unsupervised methods.

1.3.3 Intelligent Exploratory Data Analysis

In contrast to situations where informative labels are lacking for practical reasons, there are also domains where supervisory information simply is not known. This is the case for many data mining applications where the goal is to detect novel patterns in data. Examples of such situations include market basket analysis, spectral analysis of astronomy data, and automated image segmentation. Although specific labels for individual items are not known, the user may have specific requirements for the output partitions. Ideally, we would like user to be able to indicate those requirements to the clustering algorithm.

In our census data example, we may wish to analyze the data to detect interesting subgroup structure, without having labeled examples of the kinds of distinctions we wish to discover. To this end, we could apply any traditional clustering algorithm, which would produce one possible partition of the data set. Figure 1.1a shows a default partition of the data from Table 1.1. Gary and Diana have been placed in the same cluster, and Lyra and Tim appear in a different cluster. However, if we have a more sophisticated clustering goal, such as finding clusters that are consistent with the gender feature, the partition shown in Figure 1.1b is a better solution. We would like to require that individuals of different genders end up in different clusters, but regular clustering algorithms are inadequate for such tasks.



(a) Default partition (b) Consistent with gender

Figure 1.1: Different partitions of the census data shown in Table 1.1

We propose extending current methods for exploratory data analysis by enabling them to make use of feature-based clustering heuristics, such as the above restriction with respect to gender, thereby producing more "intelligent" results.

1.3.4 Structured Data Sets

Our final example involves data sets that possess an inherent structure. For example, clustering is often applied to images, where each pixel represents an item in the data set. One application is in *image segmentation*, where the goal is to automatically divide an image into homogeneous sub-regions or clusters. Although it is possible to apply a regular clustering algorithm to this problem, and cluster the pixels based on their color or intensity values, this kind of approach abandons the information inherent in the two-dimensional structure of the image. The fact that certain pixels are adjacent is important when attempting to segment an image, and that structure should be preserved as much as possible in the output.

We propose methods that will integrate information about the structure of a data set and preserve that structure in the final output, to the degree desired by the user.

As the previous examples demonstrate, there are many cases where the experimenter possesses useful information about what assumptions can be made about potential solutions, even if individual labels for every item in the data set cannot be made (or would be prohibitively expensive to make). These situations include, among others, knowledge expressed in non-label forms, partially labeled data sets, intelligent exploratory data analysis, and structured data sets.

1.4 Contributions

In this dissertation, we investigate methods for enabling clustering algorithms to take advantage of such problem-specific information in an automatic, efficient fashion. This research creates a bridge between powerful data analysis techniques and expert knowledge. In summary, this dissertation presents the following contributions:

A flexible instance-level constraint formulation. We propose the use of instance-level constraints for encoding domain knowledge. We will use both hard and soft constraints. *Hard* constraints encode restrictions that must be satisfied in the final output of the algorithm.

Soft constraints express preferences about the final partition generated by the algorithm and therefore offer additional flexibility and expressiveness. By accommodating both kinds of constraints, we provide a rich language for encoding information that is important for clustering algorithms.

A method for intelligent clustering. We describe a general method for augmenting clustering algorithms to enable them to make use of background knowledge about the problem domain. In particular, we demonstrate how to incorporate knowledge in the form of instance-level constraints. This method is not specific to any single clustering algorithm.

Three intelligent clustering algorithms. We first apply this general technique by modifying two widely-used clustering algorithms, k-means (MacQueen, 1967) and COBWEB (Fisher, 1987), so that they can accommodate hard instance-level constraints. These algorithms, though fundamentally unsupervised, are able to take advantage of problem-specific knowledge to improve their clustering accuracy. Later, we present a second modified version of k-means that can also accommodate soft constraints.

Empirical demonstration of scalability and improvements in clustering accuracy on real-world problems. We evaluate our new clustering algorithms using artificially-generated constraints and on real problems, where constraints are generated from heuristic knowledge about the problem domain. Specifically, we focus on the problems of a) automatically identifying road lanes from GPS data, b) determining noun phrase coreference relationships, and c) analyzing infra-red spectral observations of Mars. In applying our methods to such diverse problems, we are able to show that general constrained clustering algorithms can become adept experts on particular problems when making use of domain information.

1.5 Roadmap

In the next chapter, we will examine a large body of relevant work and discuss the advantages of our approach. Chapter 3 describes the details of our method of integrating background knowledge with clustering algorithms and presents two modified clustering algorithms, based on k-means and COBWEB. We then examine the application of constrained clustering to several real-world problems in Chapters 4, 5, and 6. In addition, Chapter 6 presents a third intelligent clustering algorithm that can make use of soft constraints.

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 $\mathcal{S}_n^{(k)}$:

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

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).

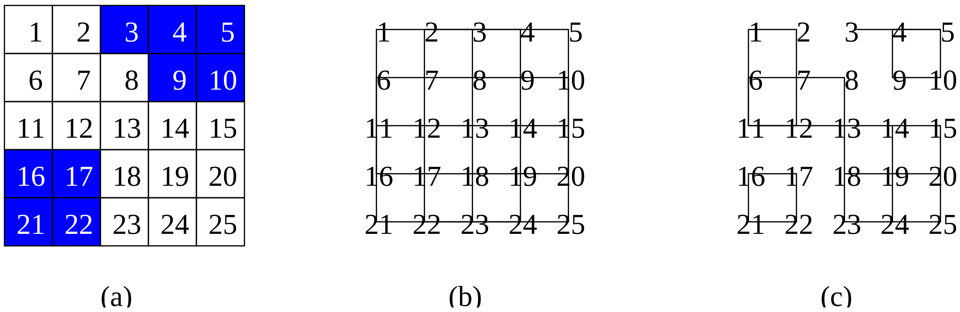


Figure 2.1: Image segmentation. (a) the image with pixels numbered; (b) the corresponding neighborhood (contiguity) relation; (c) a three-cluster result after clustering the pixels subject to relation $N(p)$

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.

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.

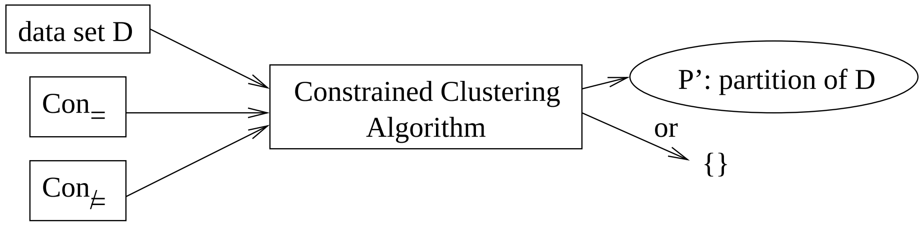


Figure 3.1: Constrained clustering architecture

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,

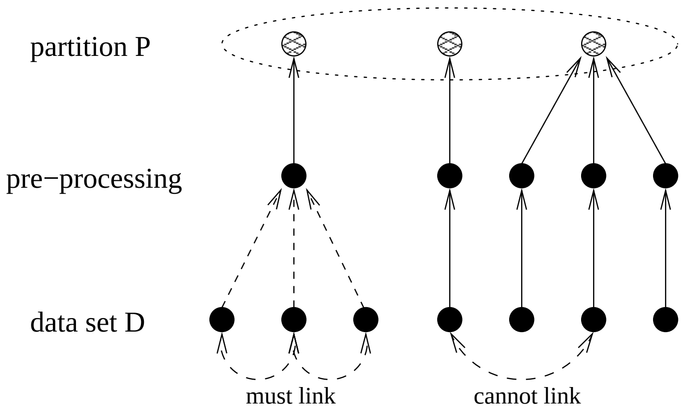


Figure 3.2: Constrained clustering process with a batch algorithm

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 $\mathsf{married}$ 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, $\mathsf{married}$, 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 $\{\ \mathsf{Gary} =_m \mathsf{Diana},\ \mathsf{Tim} =_m \mathsf{Lyra},\ \ldots\}$. 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, $\ldots$}. 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:

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

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:

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

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 $Ct_1 \ldots Ct_k$ 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 $\{C_1, \ldots, C_k\}$.

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

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

[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:

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

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

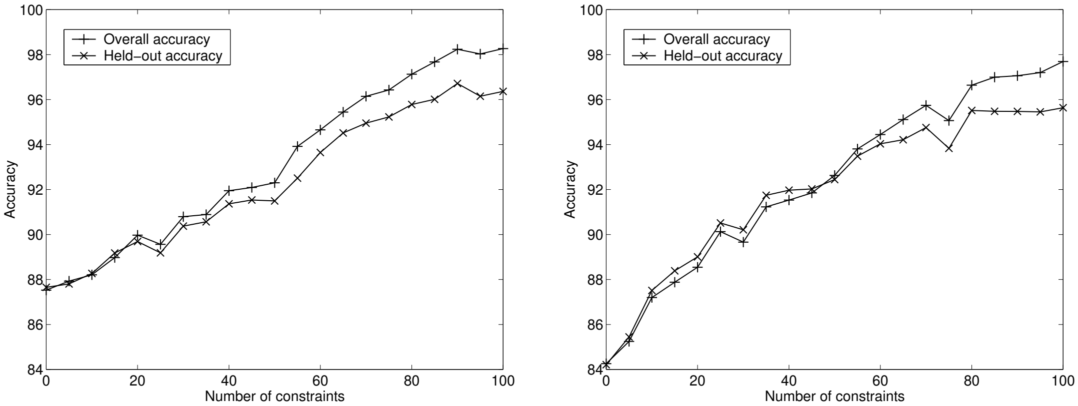
For evaluation, we compared the output partitions produced by COP-KMEANS and COP-COBWEB with the provided data labels. To calculate agreement between our results and the correct labels, we used the Rand index (Rand, 1971). This metric is commonly used to compute the similarity of two partitions, P_1 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)

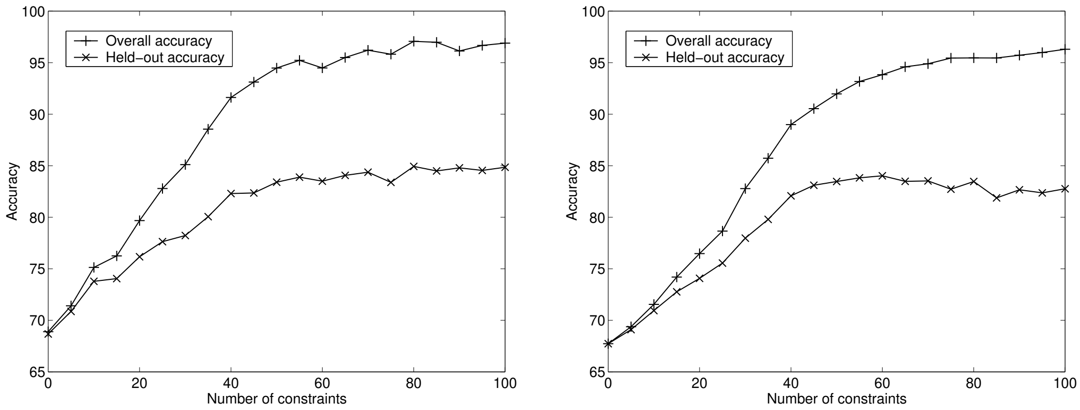
We conducted 50 trials on each data set, randomly re-ordering the data set each time, and averaged the results. Each trial is a 10-fold cross-validation run.

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



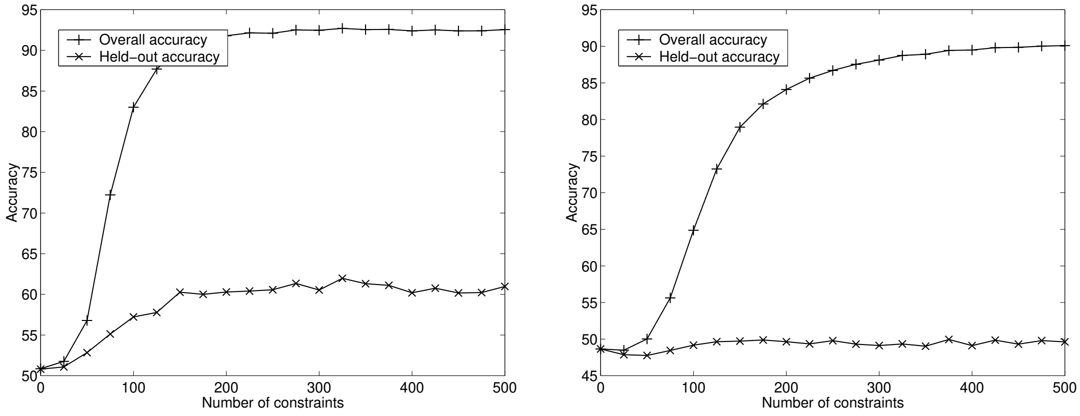
(a) COP-KMEANS (b) COP-COBWEB

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

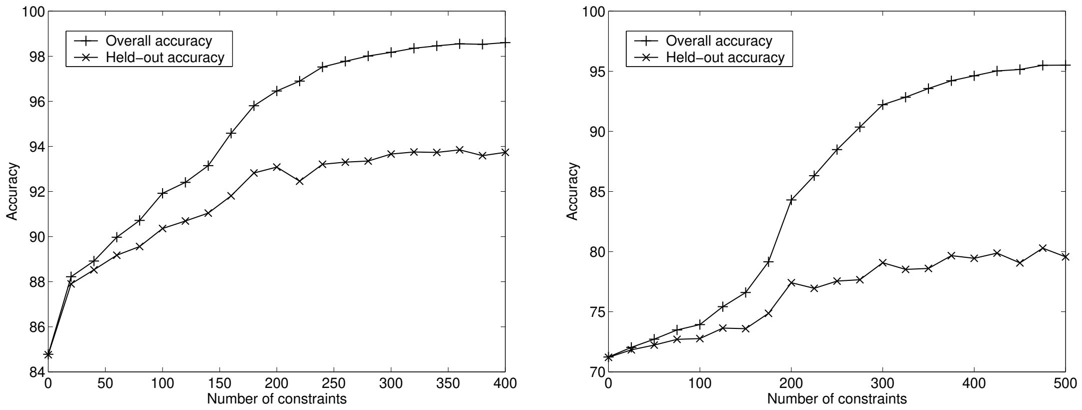
Figure 3.5: Clustering accuracy with artificially generated constraints: the tic-tac-toe data set (100 instances in two classes)

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.

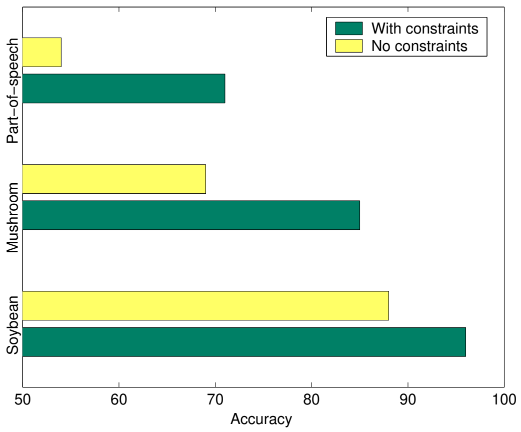
Discussion of results. Our experiments on six different data sets have shown that both of the constrained clustering algorithms exhibit significant accuracy improvements when using constraints. Figure 3.7 summarizes the improvements in held-out accuracy that we observed for certain numbers of constraints. 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.

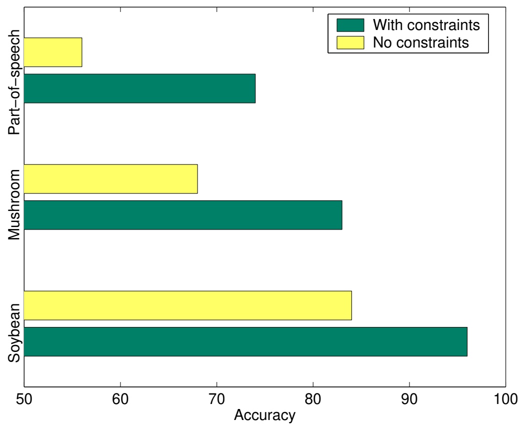
These experiments also demonstrate how our algorithms can easily handle constraint relations that apply to only a subset of the data. This contrasts with the work that has been done with contiguity constraints (see Chapter 2), where the constraint relation must encompass all data items. In addition, 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

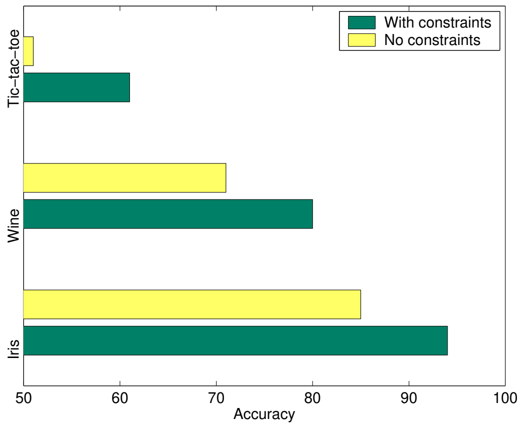
In addition to the accuracy improvements we have just presented, we also observed several other interesting results in our experiments with artificial constraints. First, we find that the use of constraints can aid COP-COBWEB in selecting an appropriate value for k, the number of clusters present in 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

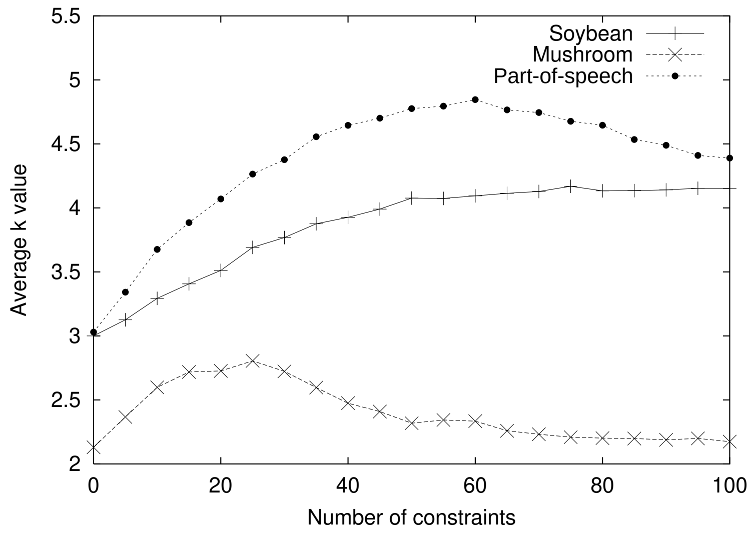
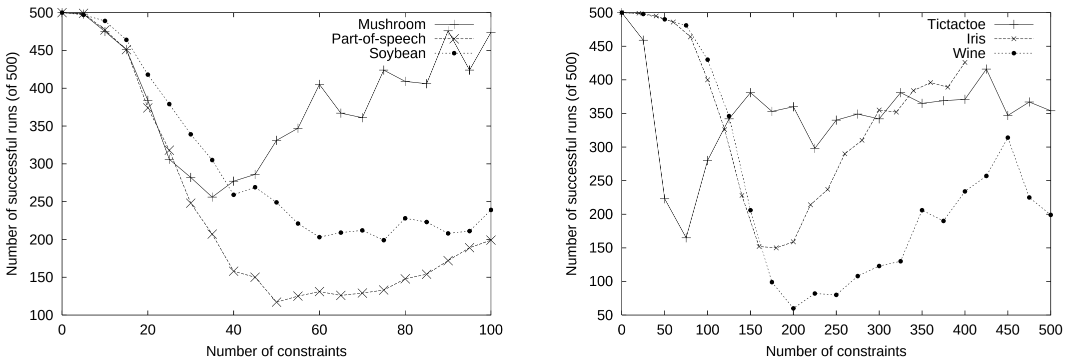


Figure 3.8: COP-COBWEB: Selected value for k, averaged over 500 runs

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

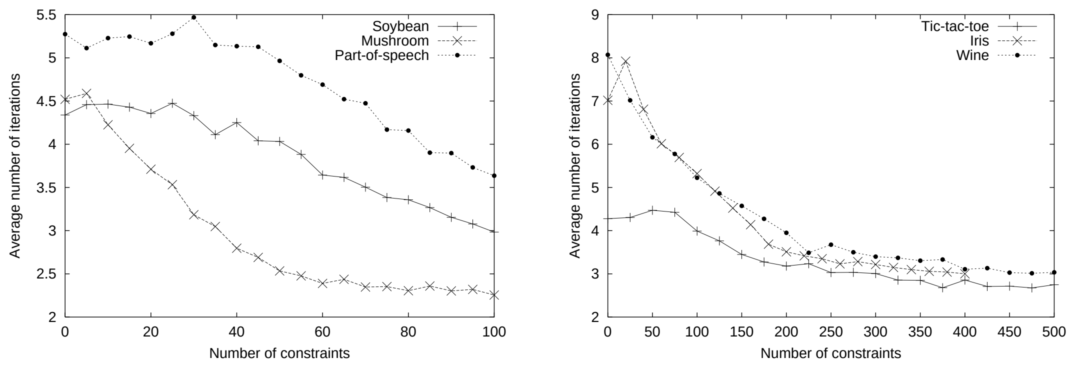
tended to be those either with low complexity or those with high complexity. The hardest problems were the intermediate ones.

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.

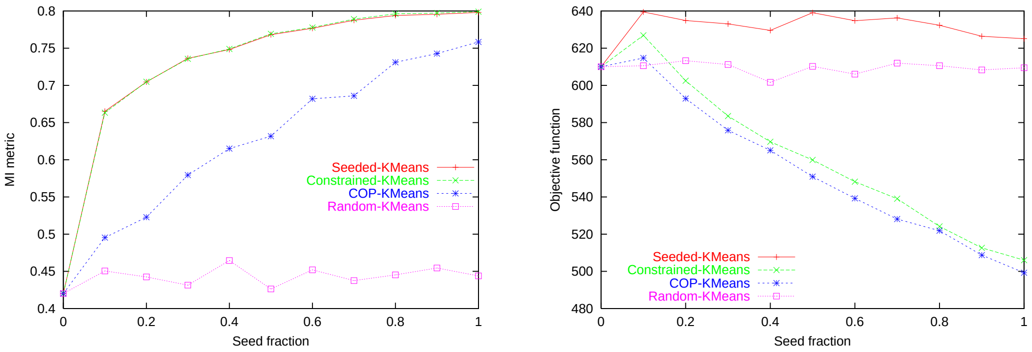
Basu et al. propose two algorithms based on this method. The first is Seeded-KMeans, which uses the labeled data to identify the initial cluster groups but then is free to modify the data assignments (i.e., break constraints) as clustering progresses. The second algorithm is Constrained-KMeans, which 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.

CHAPTER 4
APPLICATION 1: ROAD MAP REFINEMENT

The experiments presented in Chapter 3 all involved artificial problems; we used standard data sets in conjunction with constraints that were randomly generated from the true data labels. Those constraints were therefore completely reliable, in terms of guiding the clustering algorithm towards a result that was consistent with the labels used for evaluation.

However, we are also very interested in testing these methods in the context of real-world problems. Can real (and potentially noisy) domain knowledge be successfully represented as instance-level constraints, and will those constraints enable significant improvements in clustering accuracy? In this chapter, we first explore the application of intelligent clustering to the problem of automatic road map refinement (Section 4.1). Section 4.2 examines the performance of the regular k-means algorithm on this problem. In Section 4.3, we describe how to encode heuristic domain knowledge about this problem as a set of instance-level hard constraints. As we will show, the unconstrained k-means algorithm performs very poorly compared to COP-KMEANS, which has access to additional domain knowledge about the problem (Section 4.4). In the two chapters that follow, we will demonstrate the utility of our methods on two other real-world problems.

4.1 Digital Road Maps

Digital road maps currently exist that are used in several applications, such as generating personalized driving directions. However, these maps contain only coarse information about the location of a road. Map accuracy, in terms of how close the map points are to the true location of the road, is low. For example, Navigation Technologies, Inc., (NavTech) offers maps with individual points that are up to 15 meters off from the true road location (Navigation Technologies, 1996). This level of accuracy, while sufficient for generating directions for a human driver to follow, is inadequate for more sophisticated applications.

Our goal in this application is to refine digital maps by improving their accuracy and enhancing the fine details. We will produce maps that are annotated not only with the location of the road, but also with the location of individual road lanes. Maps of this nature enable the automatic generation of more precise driving directions as well as a host of more advanced applications, such as alerting a driver who inadvertently drifts from the current lane.

4.1.1 Using GPS Data to Automatically Refine Maps

One method for producing enhanced maps would be to manually re-survey each road segment on the map and to update the road maps by hand to include the new information. This would not only be extremely tedious and expensive but would also take a long time to complete. Instead, we propose using an automated method to analyze actual automobile driving patterns and extract information about the location of road lanes. This approach is low-cost, fast, and allows for easy updates to the road maps. Further, if a road lane is closed due to construction or some other obstruction, our analysis methods can detect that that lane is no longer in use and update the current status of the maps accordingly.

Our approach to this problem is based on the observation that drivers tend to drive within lane boundaries (rather than, for example, straddling two lanes). Over time, lanes should correspond to "densely traveled" regions, in contrast to the lane boundaries, which should be "sparsely traveled." Consequently, we hypothesized that it would be possible to collect data about the location of cars as they drive along a given road, and then cluster that data to automatically determine where the individual lanes are located.

4.1.2 Experimental Methodology

The data set we will be using consists of observations of several cars driving on the same network of roads. Each car's position was sampled approximately once per second using differential GPS (Global Positioning System) receivers affixed to the top of the vehicle. The DGPS receiver recorded the longitude, latitude, and altitude of the vehicle, as well as some information about how reliable each sample was. All data was taken from I-280 in the Palo Alto, CA, region. We define a *traverse* as a single pass by the same car from one end of a road segment to the other end. Consequently, each vehicle may have made more than one traverse of a given road.

Road segmentation. To facilitate processing, we segmented the data generated by each driver by matching it to NavTech's coarse road map segments. A road segment is a continuous section of a road that is terminated on each end where it intersects with other roads. Very long stretches of road may also be subdivided into separate segments by the NavTech map. Although NavTech's data points are not good enough to pinpoint the exact location of a road, they are good enough to allow us to segment the data streams into individual stretches of road.

Data format. We then converted the data into the following format. Each data point is represented by two features: its distance along the road segment and its perpendicular offset from the road centerline.[1] For evaluation purposes, we asked the drivers to indicate which lane they occupied and any lane changes. This allowed us to label each data point with its correct lane.

4.2 K-means Performance on Lane Finding

The k-means clustering algorithm (MacQueen, 1967) is a standard choice for clustering in Euclidean space. K-means has demonstrated good performance on a variety of problems, from image segmentation (Marroquin and Girosi, 1993) to speech recognition (Fontaine et al., 1994), and is known to converge to a local optimum (Selim and Ismail, 1984). It is therefore a sensible first choice for clustering our GPS data.

However, a straightforward application of k-means yields poor results on this problem. For example, Figure 4.1 shows the data set collected from several traverses of the same road segment. The horizontal axis is the distance along the road (in meters) and the vertical axis is the centerline offset. Figure 4.2 shows the output of the regular k-means algorithm on that road segment. We provided the correct value of k (number of lanes, 4) as input to k-means.

[1] The centerline parallels the road but is not necessarily located in the middle of the road. Schroedl et al. (2000) provide full details on the centerline computation.

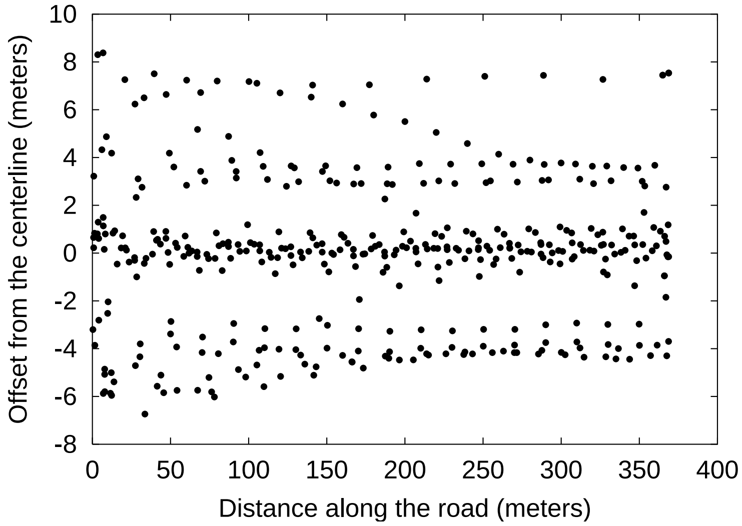


Figure 4.1: Raw data for a sample road segment; the x-axis is the distance along the road (in meters) and the y-axis is offset from the road centerline

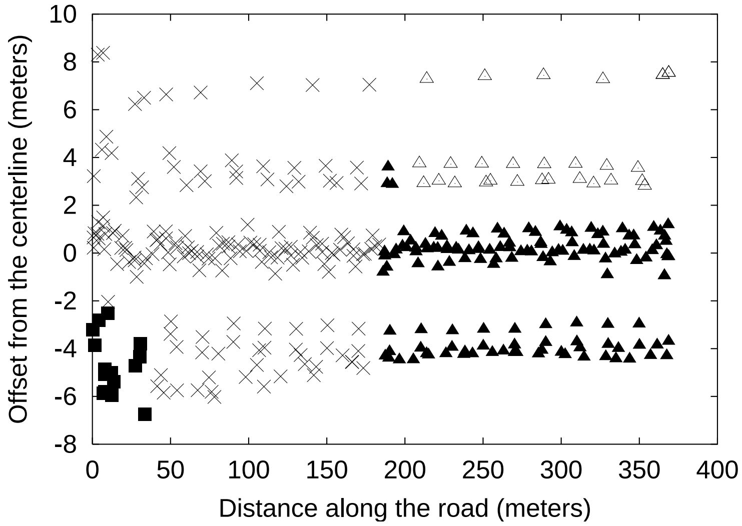


Figure 4.2: K-means output partition for the sample road segment in Figure 4.1

The points assigned to each of the resulting four lane clusters are represented by different symbols. This algorithm does a poor job of identifying the four lanes, and the mistakes it makes are representative of its general behavior on this problem. K-means by default seeks clusters that are spherical in the feature space, which results in clusters that span multiple lanes in the y direction and do not span the entire length of the road in the x direction.

To alleviate this problem, we could manually modify the distance calculation so that distance in the y direction is more heavily weighted than distance in the x direction is. Alternatively, we could re-scale all of the feature values to place more emphasis on the y distance than on the x distance. From the viewpoint of the k-means algorithm, either method would have the effect of compressing the feature space horizontally. However, the first approach would have us tweaking the distance calculation differently for every problem the k-means algorithm encounters. The second approach also has a major drawback: it requires an intimate knowledge of the clustering algorithm's bias so that the data can be pre-processed in ways that will improve performance. In contrast to both of these approaches, our method allows the algorithm to remain problem-independent. We do not need to make any internal changes to the algorithm to fit it to a specific problem, and we do not need to pre-process the data to make it fit the algorithm. Instead, we specify all problem-specific knowledge in a separate, isolated component (Con), and allow the algorithm to combine it automatically with the data.

With respect to the lane finding problem, we observe that the built-in bias of the k-means algorithm (towards spherical clusters) is inappropriate. However, this is because the k-means algorithm sees the data set only as a set of generic data points. It does not know that it is seeking road lanes, which have a specific characteristic shape. We know that road lanes tend to be elongated (along the road segment) and that they tend to be parallel to each other. In the following section, we will show how to convert our domain knowledge about the problem into useful instance-level constraints and report on the results of using those constraints with COP-KMEANS, the intelligent clustering version of k-means.

4.3 Generating Lane-finding Constraints

The constraint types defined in Chapter 3 provide us with a language to express our domain knowledge about the problem of lane finding. In this case, we focus on two domain-specific heuristics for generating constraints: trace contiguity and maximum separation.

Heuristic 1: *Trace contiguity* means that, in the absence of lane changes, all points a, b generated by the same vehicle in a single pass over a road segment should end up in the same lane. In other words, traces are contiguous in that all of the points from a single trace are linked. We convert this heuristic into a set of constraints in the following way:

$$\forall_{a,b}\ a.traverse = b.traverse \Rightarrow (a, b) \in Con_{=}$$

where each point is labeled according to its traverse of the segment. Again, the same car may have made more than one traverse of the same segment; in this case, points generated by the same car but on different passes will have different values for the $traverse$ feature.

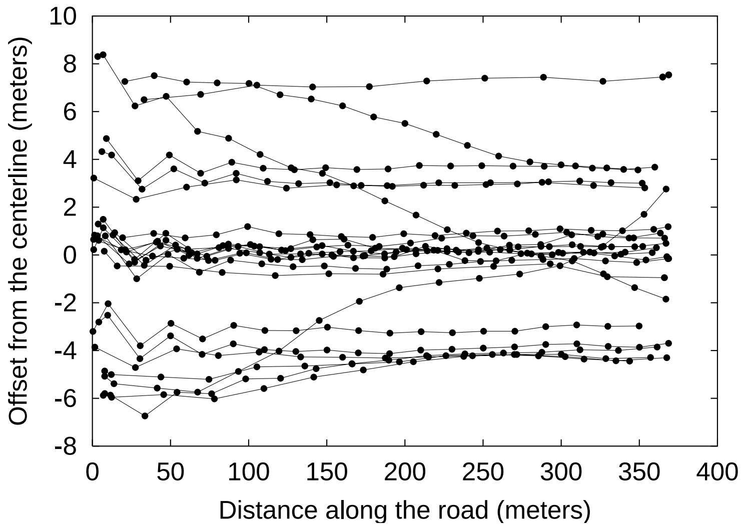


Figure 4.3: Domain knowledge encoded as pairwise links between points generated by the same traverse of the road segment

Figure 4.3 provides a conceptual representation of the relational trace contiguity information by linking each adjacent pair of points that came from the same traverse. We can see that two points may appear to be very far apart in this Euclidean space, yet still be generated by the same car. The transitivity of the must-link relation ensures that two such points will still end up in the same lane cluster.

Heuristic 2: *Maximum separation* refers to a limit on how far apart two points can be (perpendicular to the centerline) while still being in the same lane. If two points are within ten meters of each other in distance along the road segment, and their centerline offsets differ by at least four meters, then we generate a constraint that will prevent those two points from being placed in the same cluster:

$$\underset{a,b}{\forall} \ |a.x - b.x| < 10.0 \text{ and } |a.y - b.y| > 4.0 \Rightarrow (a, b) \in Con_{\neq}$$

This heuristic helps express the idea that lanes are parallel; the fact that two points are close together in the x direction has a different meaning than two points being close together in the y direction. Recall that the x and y axes do not correspond to a fixed set of coordinates in the real world. They simply represent "distance along the road" and "perpendicular offset from the centerline." The x axis follows the road centerline, which may be curving in the real Euclidean space.

We could also have included a set of links in Figure 4.3 between items deemed too far apart by the maximum separation heuristic, but this would make the figure difficult to interpret visually. The full set of constraints representing our domain knowledge about the problem, however, includes both $Con_{=}$ and $Con_{\neq}$.

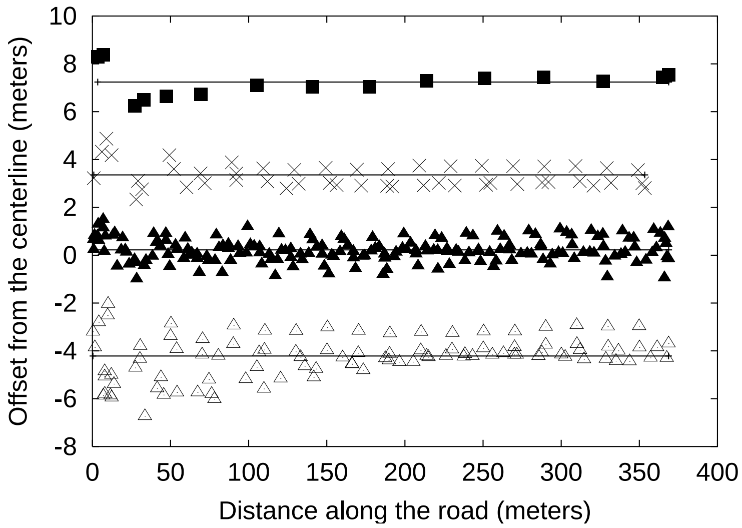


Figure 4.4: COP-KMEANS output for the sample road segment in Figure 4.1; lane centers are marked by solid lines

Figure 4.4 shows the correct partition of data points into lanes, according to the labels we obtained from the drivers. In addition, the final cluster (lane) centers appear as solid lines. Compare to Figure 4.2, which is the output generated by the k-means algorithm.

4.3.1 Lane Change Detection

The trace contiguity heuristic applies only in the absence of lane changes. If a driver changes lanes, we need to break that trace into a set of sub-traces so that no trace spans multiple lanes. This is something of a chicken-and-egg problem, since we do not yet know where the lane boundaries are, but we need to be able to detect lane changes! Since we are working with domain knowledge in the form of heuristics, however, we will content ourselves with an approximation method for detecting lane changes.

We define a *lane change* as a contiguous sequence of points $d_1, \ldots, d_m$, all generated on the same traverse, whose y feature values define a monotonic sequence such that $|d_1.y - d_m.y| >$ TOLSEP. TOLSEP is the maximum separation tolerance allowed for points within a single lane. This value is somewhat road-dependent; we used 2.0 meters for our experiments with the I-280 data.

The points in a lane change sequence $d_1, \ldots, d_m$ do not belong to a lane. They are transitional points signaling motion from one lane to another. Because it is not possible to unambiguously assign them to a lane, it is also not possible to assign them to a cluster. Therefore, after identifying a lane change sequence, we remove all of the points in that sequence from the data set for clustering purposes. This also severs the trace contiguity connection from the points that occur before the lane change with those that occur after the lane change (in terms of distance along the road segment). For example, Figures 4.2 and 4.4 have the lane change points removed.

4.3.2 Accuracy of Lane-finding Constraints

The constraints we used for our approach to this problem are, as previously noted, generated from heuristics. For example, the trace contiguity heuristic's accuracy is dependent on the accuracy of the lane change detection heuristic. We selected approximate values for TOLSEP, but those values may not always be precisely correct, due to noise in the GPS data and natural variations in road construction. Therefore, the constraints generated by these heuristics may not be completely reliable. This is important because we are treating each constraint as a hard restriction on the set of possible solutions. As we will see, the constraints are "reliable enough" to provide very high performance on this task. Nevertheless, it is important to evaluate the accuracy of the constraints, since that accuracy will affect the accuracy of our output. In the next section, we will accompany our discussion of experimental results with an analysis of the accuracy of the input constraints.

4.4 Experimental Results

This section describes the results we obtained when applying k-means and COP-KMEANS to the same set of 20 different road segments. In each case, the algorithms were required to select the best value for k. We first describe two important implementation details: how cluster centers are represented for this problem, and how to automatically select the best value for k. We then discuss our experiments and observe that COP-KMEANS, which has access to the heuristic domain knowledge described above, consistently performs much better than the regular k-means algorithm does. Finally, we analyze the errors made by COP-KMEANS and discuss the reliability of the heuristic constraint sets.

4.4.1 Cluster Center Representation

To better analyze performance in this domain, we modified the cluster center representation. The usual way to compute the center Ct_i of cluster C_i is to average all of its constituent points in the following way:

$$Ct_i = \frac{1}{|C_i|} \sum_{d \in C_i} d \tag{4.1}$$

where each d is a vector of feature values. Thus, the center of a cluster is itself a point; in this case, Ct_i would have two values and be represented by $(Ct_i.x, Ct_i.y)$. There are two significant drawbacks to using that representation for this problem. First, the center of a lane is a point halfway along its extent, which commonly means that points inside the lane but at the far ends of the road appear to be extremely far from the cluster center. Second, and more significantly for any practical use of this method, applications that make use of the clustering results need more than a single point to define a lane.

Instead, we represented the center of each lane cluster with a line segment parallel to the road centerline (see Figure 4.5). We express this lane representation with a tuple $\langle l, r, y \rangle$, which indicates the x coordinates of the left (l) and right (r) ends of the lane as well as the (constant) centerline offset of the lane, y:

$$Ct_i.l = \min_{d \in C_i} d.x \tag{4.2}$$

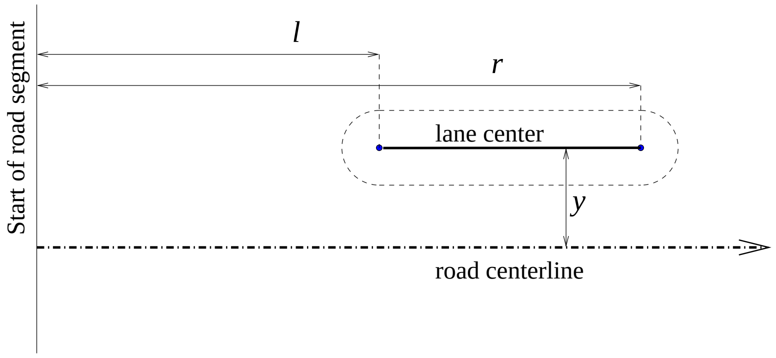


Figure 4.5: Representation of a cluster (lane) center. l and r are the x coordinates of the left and right ends of the lane, and y is the y offset of the lane from the centerline. Points that fall anywhere on the dashed line are considered equally distant from the center of the cluster.

$$Ct_i.r = \max_{d \in C_i} d.x \tag{4.3}$$

$$Ct_i.y = \frac{1}{|C_i|} \sum_{d \in C_i} d.y \tag{4.4}$$

Thus, the center of the cluster is a line segment from $(Ct_i.l, Ct_i.y)$ to $(Ct_i.r, Ct_i.y)$. This formulation more accurately models what we conceptualize as "the center of the lane," provides a better basis for measuring the distance from a point to its lane cluster center, and also provides useful output for other applications. To ensure a fair comparison, both the basic k-means algorithm and COP-KMEANS make use of this lane representation (for this problem).

K-means uses variance as its objective function, which relies on the ability to calculate the distance from a point to its assigned cluster. For two-dimensional data, the distance between a point d and a cluster C_i would be calculated as

$$dist(d, C_i) = \sqrt{(d.x - Ct_i.x)^2 + (d.y - Ct_i.y)^2}. \tag{4.5}$$

Due to our modification to the cluster center representation, we instead make use of the following method for calculating distance:

$$dist(d, C_i) = \begin{cases} \sqrt{(d.x - Ct_i.l)^2 + (d.y - Ct_i.y)^2} & d.x < Ct_i.l \\ \sqrt{(d.x - Ct_i.r)^2 + (d.y - Ct_i.y)^2} & d.x > Ct_i.r \\ |d.y - Ct_i.y| & \text{otherwise.} \end{cases} \tag{4.6}$$

This is a simple extension of the Euclidean distance to handle a line rather than a point. If d is to the left of the cluster's left endpoint, then we calculate the Euclidean distance from d to $(Ct_i.l, Ct_i.y)$. If d is to the right of the cluster's right endpoint, we calculate the distance from d to $(Ct_i.r, Ct_i.y)$. Otherwise, we calculate the vertical distance from $d.y$ to the cluster center $(Ct_i.y)$. For example, in Figure 4.5, any points that fall on the dashed line are considered equidistant from the cluster center.

It is interesting to note that even with this more expressive cluster center representation, the k-means algorithm continues to create very spherical clusters. In the absence of constraints, the k-means algorithm has no incentive to "grow" its clusters horizontally and to take advantage of this representation. Therefore, its greedy search never encounters solutions with very elongated clusters. In contrast, when clustering with constraints, the constraints provide exactly that kind of motivation.

4.4.2 Selecting k

In these experiments, both algorithms were required to automatically select the best value for the number of clusters, k. Each algorithm generated a partition for each value of k, from 1 to 5, and then selected the partition (and thus, the value of k) that it considered to be the best. In this problem domain, we know that there is a reasonable upper limit on k; in particular, the roads we were analyzing were all highway roads with a maximum of four lanes per road segment. Therefore, we can easily bound the possible range of values for k for this problem.

To enable the algorithms to select their best output partition, we used a second measure that trades off cluster cohesiveness against simplicity (in this case, measured as the number of clusters). More precisely, we define the *quality* of a partition by calculating the average squared distance from each point to its assigned cluster center and penalizing for the complexity of the solution:

$$quality(C_1, \ldots, C_k) = \frac{1}{n}\,(\sum_{d \in D} dist(d, C_d)^2) \cdot k^2. \tag{4.7}$$

The goal is to minimize this value. Note that this measure differs from the objective function used by k-means and COP-KMEANS while clustering (variance). In the language of Jain and Dubes (1988), quality is a relative criterion, while variance is an internal criterion. An internal criterion (objective function) allows an algorithm to select between different potential solutions generated when using the same set of input parameters. A relative criterion allows an algorithm to select between solutions generated on different runs, with different parameters (in this case, different values for k).

In the lane finding domain, the problem of selecting k is particularly challenging due to the large amount of noise in the GPS data. Each algorithm performed 30 randomly-initialized trials with each value of k (from 1 to 5). COP-KMEANS selected the correct value for k for all but one road segment, but k-means *never* chose the correct value for k (even though it was using the same method for selecting k). This is due to its strong bias towards spherical clusters; the most spherical cluster is obtained by assigning all points to a single cluster.

4.4.3 Results on I-280 Data

Table 4.1 presents accuracy results for both algorithms over 20 road segments. Accuracy was calculated using the Rand metric (see Section 3.5.1) and indicates the amount of agreement between the output partition (set of lanes) and the true lane labels. These results represent overall accuracy rather than held-out accuracy, since determining the right set of

Table 4.1: Lane finding performance (Rand index). Each algorithm was given the true value for k.

Segment (size)	K-means	COP-KMEANS	Constraints alone
1 (699)	49.8	100	36.8
2 (116)	47.2	100	31.5
3 (521)	56.5	100	44.2
4 (526)	49.4	100	47.1
5 (426)	50.2	100	29.6
6 (503)	75.0	100	56.3
7 (623)	73.5	100	57.8
8 (149)	74.7	100	53.6
9 (496)	58.6	100	46.8
10 (634)	50.2	100	63.4
11 (1160)	56.5	100	72.3
12 (427)	48.8	96.7	59.2
13 (587)	69.0	99.8	51.5
14 (678)	65.9	100	59.9
15 (400)	58.8	100	39.7
16 (115)	64.0	76.6	52.4
17 (383)	60.8	98.9	51.4
18 (786)	50.2	100	73.7
19 (880)	50.4	100	42.1
20 (570)	50.1	100	38.3
Average	**58.0**	**98.6**	**50.4**

constraints is part of the problem; the constraints are not artificially generated from the true labels. The number of data points for each road segment is also indicated. These data sets are much larger than the UCI data sets discussed in Section 3.5, providing a chance to test the ability of the algorithm to scale to larger problems.

As shown in Table 4.1, COP-KMEANS consistently outperformed the unconstrained k-means algorithm, attaining 100% accuracy for all but four data sets and averaging 98.6% overall. The unconstrained version performed much worse, averaging 58.0% accuracy. The clusters the latter algorithm produces often span multiple lanes and never cover the entire road segment lengthwise, even though it was given the correct value for k. As discussed above, lane clusters have a very specific shape: they are greatly elongated and usually oriented horizontally (with respect to the road centerline). Yet even when the cluster center is a line rather than a point, k-means seeks compact, usually spherical clusters. Consequently, it does a very poor job of locating the true lanes in the data.

Figure 4.2 showed the output of the regular k-means algorithm for road segment 15. The correct value of k was specified to obtain this output (without it, the algorithm selects $k = 1$). COP-KMEANS achieves 100% accuracy on this road segment (and it selects the proper value of k). The correct partition, which is identical to the COP-KMEANS output, was shown in Figure 4.4.

The final column in Table 4.1 is a measure of how much is known after generating the constraints and before doing any clustering. It shows that an average accuracy of 50.4% can be achieved using the background information alone. This demonstrates that neither general similarity information (k-means clustering) nor domain-specific information (constraints) alone perform very well, but that combining the two sources of information effectively (COP-KMEANS) can produce excellent results.

4.4.4 Analysis of COP-KMEANS Errors and Constraint Accuracy

An analysis of the errors made by COP-KMEANS on the lane-finding data sets shows that each mistake arose for a different reason. For road segment 12, the algorithm incorrectly included part of a trace from lane 4 in lane 3. This appears to have been caused by noise in the GPS points in question: they are significantly closer to lane 3 than lane 4. The same thing happened for a single point in road segment 13. Alternatively, the labels for this trace may themselves be noisy. On road segment 16, COP-KMEANS chose the wrong value for k (it decided on three lanes rather than four). This road segment contains relatively few data points (115, compared to the average of 534), which possibly contributed to the difficulty. Finally, for road segment 17, a small part of one trace was assigned to the wrong lane due to an incorrect lane-change detection. Since COP-KMEANS made so few errors on this data set, it is not possible to provide a more general characterization of their causes.

We mentioned above that quality of the output clusters obtained from COP-KMEANS will be sensitive to the reliability of the constraints it is given. In addition, since these constraints were generated heuristically, we may not expect them to be completely reliable. Consequently, we evaluated the accuracy of the constraints that were generated for each road segment by calculating how well they agreed with the true lane labels. We define two sets of *accurate* constraints, $Acc_=$ and $Acc_{\neq}$, which are subsets of $Con_=$ and $Con_{\neq}$ respectively, as follows:

$$Acc_= = \{(a,b) \in Con_= \mid a.lane = b.lane\} \tag{4.8}$$

$$Acc_{\neq} = \{(a,b) \in Con_{\neq} \mid a.lane \neq b.lane\} \tag{4.9}$$

where $a.lane$ is the true lane label for point a. We then calculate constraint accuracy:

$$\text{C-acc} = \frac{|Acc_=| + |Acc_{\neq}|}{|Con_=| + |Con_{\neq}|}. \tag{4.10}$$

For all but one of the road segments presented in Table 4.1, C-acc is 100.00%. The exception is segment 13, where the constraint set is 99.93% accurate. The overall constraint accuracy for the entire collection of road segments, which includes almost 2 million constraints, is 99.99%. In this case, our heuristic domain knowledge is very accurate; it is almost entirely consistent with the true data labels. In addition, for the single road segment with slightly inaccurate constraints, COP-KMEANS was still able to achieve 99.8% performance.

In fact, it is instructive to evaluate the sensitivity of COP-KMEANS to the accuracy of the input constraints. For example, if we modify the lane change detection heuristic to use TOLSEP $= 3.0$ meters, the accuracy of the constraints for segment 13 drops to 99.86%. Impact on COP-KMEANS performance is minimal, however; it drops from 99.797% to 99.795%.

4.4.5 Is K-means a Straw Man?

It might be argued that k-means is simply a poor choice of algorithm for this problem. For example, there are versions of k-means which are biased towards elliptical, rather than spherical, clusters, which might be expected to have better performance on this problem. However, selecting a different algorithm for each problem encountered has its drawbacks. First, we lose any advantages obtained by having a general solution to the clustering problem. For example, we must have access to a comprehensive suite of clustering algorithms, each with their own bias, and each of which must be maintained. Second, and more importantly, selecting the proper algorithm for a given problem requires specialized knowledge about clustering, each algorithm's bias, and the ability to decide what kind of bias is appropriate for the problem. Finally, a specialized algorithm with a suitable bias simply may not be available. We have access to a k-means algorithm that prefers ellipsoidal clusters, but we do not have one that prefers, for example, hyper-rectangular clusters. If that is the most appropriate bias for a given problem, then we must implement a new version of k-means to satisfy it. We therefore believe that a better alternative is to maintain a *general* clustering algorithm that can accommodate domain-specific knowledge input to *automatically specialize* by adapting or overriding its built-in bias.

In addition, the marked improvements we observed with COP-KMEANS suggest another advantage of this method: algorithm choice may be of smaller importance when constraints based on domain knowledge are available. For this task, even a poorly-performing algorithm can boost its performance to extremely high levels. In essence, it appears that domain knowledge can make performance less sensitive to which algorithm is chosen. This is good news for anyone wishing to make use of unsupervised learning methods without having the background to make informed decisions about which algorithm should be used.

4.4.6 Comparison to agglom

Rogers et al. (1999) previously experimented with a clustering approach that viewed lane finding as a one-dimensional problem. Their algorithm (agglom) only made use of the centerline offset of each point, ignoring the position of a point along the road segment. They used a hierarchical agglomerative clustering algorithm that terminated when the two closest clusters were more than a given distance apart (which represented the maximum width of a lane).

This approach is quite effective when there are no lanes that merge or split in a road segment, i.e., each lane continues horizontally from left to right with no interruptions. For the data sets listed in Table 4.1, their algorithm obtains an average accuracy of 99.4%, which is slightly higher than the results obtained with COP-KMEANS.[2] This is largely due to the impact of a single road segment (segment 16) for which COP-KMEANS chose the wrong value for k, as previously described.

However, all of these data sets were taken from a highway, where the number of lanes is constant over the entirety of each road segment. In cases where there are lane merges or splits, the one-dimensional approach may have difficulty representing the extent of a lane along the road segment. We expect that when processing data obtained from a larger variety of roads, including segments with lane merges and splits, COP-KMEANS and agglom may have performance that differs more greatly.

4.5 Summary

In this chapter, we applied the COP-KMEANS algorithm to a real-world problem. We demonstrated how to encode heuristic domain knowledge as a set of instance-level constraints and reported on the significant performance gains they provide. COP-KMEANS, which combines generic similarity information with problem-specific domain knowledge, is able to significantly outperform either source of information in isolation. We also analyzed the accuracy of these heuristic constraints and found them to be highly accurate, which contributes to the performance of COP-KMEANS. In the next chapter, we will see that COP-COBWEB also achieves performance increases when applied to a (different) real-world problem.

[2]A maximum merging distance of 2.5 meters was specified.

CHAPTER 5
APPLICATION 2: NOUN PHRASE COREFERENCE RESOLUTION

In Chapter 4, we saw how COP-KMEANS performed much better than the regular k-means algorithm on the problem of automatic map refinement. In this chapter, we explore the application of COP-COBWEB to a different real-world problem, that of noun phrase coreference resolution. The major result of this chapter is a demonstration that COP-COBWEB is able to effectively apply constraints to improve its performance on the coreference problem.

In Section 5.1, we define this problem and justify our approach to it. Next, we describe COBWEB's default performance on this problem and discuss how appropriate linguistic constraints can be generated (Section 5.2). Evaluation in this domain is particularly challenging, so we devote some time to a thorough discussion of it in Section 5.3. Finally, we present experimental results on news articles (Section 5.4) and compare COP-COBWEB's performance to that of other systems (Section 5.5).

5.1 Noun Phrase Coreference

The field of Natural Language Processing (NLP) is concerned with developing algorithms to process text written in human (natural) languages. There are many steps involved in "processing" (i.e., understanding) a text, including *parsing* the input text to discover the syntactic structure of the sentences, performing *word sense disambiguation* to match words with their contextual definitions, and higher-level analyses to discover deeper semantic meanings.

One of the steps in text processing is referred to as *noun phrase coreference resolution*. Noun phrases include proper names (e.g., "Frodo Baggins"), common nouns ("a hobbit"), and pronouns ("he"). The purpose of this step is to identify pairs of *coreferent* noun phrases, i.e., those that refer to the same entity. We perform this task automatically each time we encounter a pronoun in written or spoken language: we deduce which other noun phrase(s) it refers to. For example, "it" in the previous sentence refers to "a pronoun." You, as a human, probably deduced that without any conscious effort. Although humans are generally able to perform this resolution step with ease, it presents a formidable challenge to automated NLP systems. Noun phrase coreference is a complex linguistic phenomenon, and successful resolution relies on cues from syntactic structure, semantic constraints, and shared world experience on the part of the author and the reader.

As a concrete example, an excerpt from a news article about space exploration is shown in Figure 5.1 with each noun phrase enclosed in square brackets. Any reader of this text, whether a human or a computer, must be able to determine the coreference relationships to fully understand what the text means. For example, the fact that "Mars" and "the Red Planet" refer to the same entity is a crucial component in interpreting the passage. The figure also visually combines coreferent noun phrases by placing them into three separate groups. This document contains a total of 12 coreference groups; the nine remaining bracketed noun phrases do not participate in a coreference relationship and are considered singletons.

5.1.1 A Clustering Approach

Coreference is a binary relation over noun phrases. It is an equivalence relation that operates on pairs of noun phrases. Consequently, the task of identifying coreference classes can be naturally phrased as a partitioning, or clustering, problem. With a suitable set of

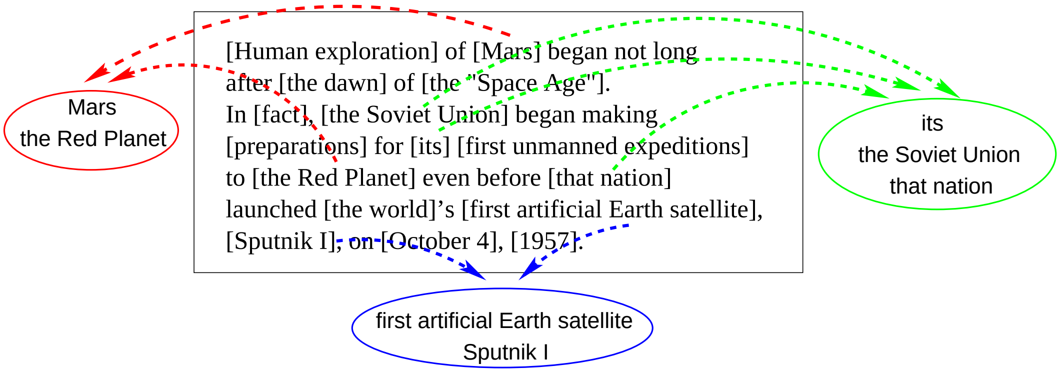


Figure 5.1: An excerpt from a news article with noun phrases bracketed and coreference classes indicated

features to describe each noun phrase and a method for measuring the distance between each pair of noun phrases, we can apply any standard clustering algorithm to this task. Each of the resulting clusters corresponds to what we referred to earlier as a "group."

In previous work on this problem, we developed an approach to this problem that uses a clustering algorithm with a highly specialized, hand-tuned distance measure that incorporates constraint information (Cardie and Wagstaff, 1999). However, this algorithm is specific to the problem of noun phrase coreference and cannot be used for other tasks. In contrast, we will present the results of using the multi-purpose COP-COBWEB algorithm, which takes in the data set of noun phrases separately from the set of constraints.

In this section, we will describe in more detail how noun phrase coreference resolution can be cast in terms of a partitioning problem. We will also discuss the features we use to represent each noun phrase. In addition, we will justify our choice of COP-COBWEB for this problem. Finally, we will show an example of the output produced by the system in the absence of constraints.

5.1.2 Coreference Resolution as Partitioning

As discussed above, we can view the coreference resolution problem as a partitioning task. More precisely, the data set $D = \{NP_i\}$ consists of all of the noun phrases in an input text. We have selected six features to represent each noun phrase: $NP_i = < F_1, \ldots, F_6 >$. We seek a partition $P = \{C_1, \ldots, C_k\}$ such that each cluster C_j represents an equivalence class of noun phrases with respect to coreference.

Generation of noun phrase features. To create the input data set for a given document, we used the Empire noun phrase finder (Cardie and Pierce, 1998) to locate all noun phrases in the text. Note that Empire identifies only *base noun phrases*, i.e., simple noun phrases that contain no other smaller noun phrases within them. For example, "Human exploration of Mars" is too complex to be a base noun phrase. It contains two base noun phrases, "Human exploration" and "Mars," connected by a preposition.

Each noun phrase in the input text is represented by a set of six features, which are listed in Table 5.1. The choice of features was inspired by heuristics proposed by research

Table 5.1: Features used to represent noun phrases for coreference resolution

F_i	**Feature**	**Description**
1	POSITION	Position in the document (sequential)
2	HEAD NOUN	Rightmost noun in the noun phrase
3	SEMANTIC CLASS	General class from WordNet
4	NUMBER	singular or plural
5	GENDER	masculine, feminine, either, or neither
6	ANIMACY	animate, inanimate, or neither

on coreference in the field of computational linguistics (Mitkov, 1999). All feature values are automatically generated by simple heuristics and, therefore, are not always perfect. Because Empire is a partial parser, we do not have access to the full parse structure of the document. This additional information could potentially increase the accuracy of the noun phrase feature values, or allow the use of additional features, such as the grammatical *role* of the noun phrase in the sentence (e.g., subject or object).

As an example, the values for feature 4 (NUMBER) were determined by checking whether the noun phrase ended with an 's'. If so, it was labeled plural; otherwise, it was singular. Feature 3 (SEMANTIC CLASS) relies on the availability of a semantic network. We used WordNet (Fellbaum, 1998) to classify the head noun into one of several general categories: animal, city, human, time, object. If none of these applied, we used the noun phrase's immediate parent in WordNet's class hierarchy. Separate heuristics were used to identify a noun phrase as a number, money, or a company. If the system was unable to assign a specific feature value, the noun phrase under consideration received a value of unknown for that feature. Further details on our feature generation methods for this problem are discussed by Cardie and Wagstaff (1999).

Selection of COP-COBWEB. We selected the COP-COBWEB algorithm for this problem for two main reasons. First, COP-COBWEB automatically determines the number of clusters in the data set. Second, its incremental, order-sensitive nature is a good fit to our model of how humans process documents. Since documents are presumably written to facilitate human processing, we can expect that COP-COBWEB will benefit from this structure.

In contrast to our experiments with automatic map refinement, there is no easy way to estimate a good value for k, the number of noun phrase classes present in a document. As explained in Chapter 4, when clustering to find lanes in GPS data, it was reasonable to try a small set of different k values and select the best resulting partition. This works because roads rarely have more than four or five distinct lanes. However, with noun phrase coreference, the number of classes could vary from 1 to n, the number of noun phrases in the document. Documents may contain hundreds or even thousands of noun phrases. Therefore, COP-COBWEB, which automatically determines the number of clusters present, is a better choice for this problem.

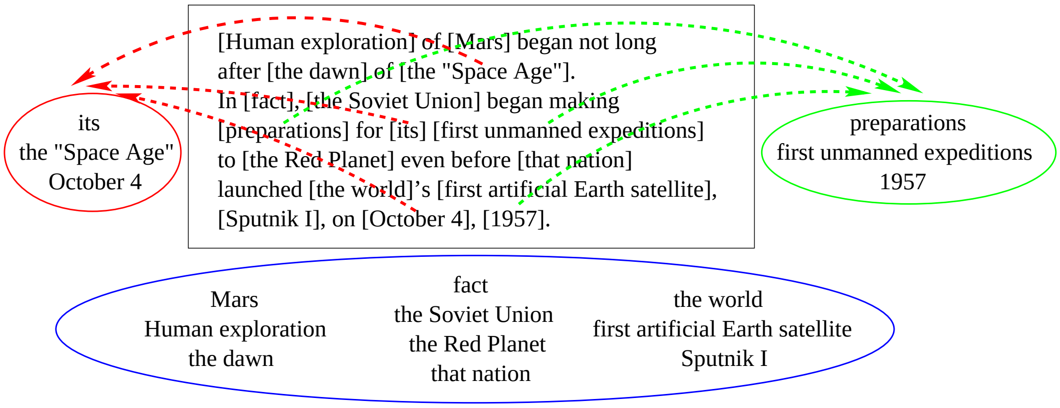


Figure 5.2: Coreference output for the text in Figure 5.1 when clustering without constraint information. Arrows from noun phrases to the largest cluster have been omitted.

Most documents (in particular, news articles) are written with the assumption that a human will be reading them. Humans generally read sequentially, from the beginning to the end of the document. We therefore process the text in an incremental fashion, continually building and refining our mental comprehension of the document as we progress. This means that our interpretation of what we encounter at any given point in the document is influenced by what we have already seen (but not by the remainder of the document).

COBWEB was originally developed based on observations about human conceptual processing (Fisher, 1987). It was intentionally designed to be incremental (and therefore sensitive to the order of the input observations) since that ability is essential in real-world situations. If a learner is limited to batch processing, then it cannot form abstractions or draw conclusions about its observations until it is entirely done processing them. Our modified version, COP-COBWEB, retains this incremental quality. With respect to the problem of text processing, incremental processing is very important. Because humans are the expected readers, documents written *by* humans are likely structured and optimized for an incremental reader. Our choice of COP-COBWEB exploits this inherent structure to enable better processing of the noun phrases in a document.

5.1.3 COBWEB Performance on Coreference Resolution

Let us return to the example text presented in Figure 5.1. Standard (unconstrained) clustering of the data using COBWEB produces the output shown in Figure 5.2 (for clarity, the arrows from noun phrases to the largest cluster have been omitted). In this case, COBWEB identified a total of three coreference clusters. We can see that the algorithm is overly generous in linking noun phrases together; the correct answer (according to Figure 5.1) contains 12 clusters. COBWEB is generally biased towards the creation of fewer clusters, since it seeks a partition of the data of minimal complexity. After discussing the kind of constraints that are useful for this problem, we will show the output of COP-COBWEB, using constraints, on this example in Section 5.2.1.

Table 5.2: Heuristics for generating constraints between NP_i and NP_j for coreference resolution

Compatibility		**Article**	
HC_1	$NP_i.\text{NUMBER} \neq NP_j.\text{NUMBER}$	HC_5	$i < j$ and $NP_j.\text{ARTICLE} =$ indefinite and $NP_i.\text{PROPER} =$ no and $NP_i.\text{PRONOUN} =$ none
HC_2	$NP_i.\text{ANIM} \neq NP_j.\text{ANIM}$		
HC_3	$NP_i.\text{GENDER} \neq$ either and $NP_j.\text{GENDER} \neq$ either and $NP_i.\text{GENDER} \neq NP_j.\text{GENDER}$	HC_6	$i < j$ and $NP_i.\text{ARTICLE} =$ none and $NP_j.\text{ARTICLE} =$ definite and $NP_i.\text{PROPER} =$ no and $NP_i.\text{PRONOUN} =$ none and $NP_i.\text{SEMCL} \neq NP_j.\text{SEMCL}$
HC_4	$NP_i.\text{GENDER} =$ either and $NP_j.\text{GENDER} =$ neither	HC_7	$i < j$ and $NP_i.\text{ARTICLE} \neq$ none and $NP_j.\text{ARTICLE} =$ none and $NP_j.\text{PROPER} =$ no and $NP_j.\text{PRONOUN} =$ no
Semantic class		**Appositive**	
HC_8	$NP_i.\text{SEMCL} \neq NP_j.\text{SEMCL}$ and ($NP_i.\text{SEMCL} =$ company or $NP_i.\text{SEMCL} =$ human)	HM_1	$NP_j.\text{APPOSITIVE} =$ yes and $i = j - 1$
HC_9	$NP_i.\text{SEMCL} \neq NP_j.\text{SEMCL}$ and $NP_{i,j}.\text{SEMCL} \neq$ company and $NP_{i,j}.\text{SEMCL} \neq$ human		

5.2 Generating Coreference Constraints

Once again, we make use of the constraint types defined in Chapter 3 to express our domain knowledge about noun phrase coreference. This problem has been studied extensively by linguists and has led to several heuristic rules that describe how coreference resolution can be done.

Table 5.2 lists the ten heuristics, HC_1 to HC_9 and HM_1, that we used to encode domain knowledge about coreference in the form of instance-level constraints.[1] These heuristics, like the feature set, were inspired by work in computational linguistics and by our own previous experience with noun phrase coreference (Cardie and Wagstaff, 1999). Each heuristic in the table takes as input two noun phrases, NP_i and NP_j, and generates either a cannot-link constraint (for HC_h) or a must-link constraint (for HM_1) if the heuristic's conditions are

[1]For simplicity, we here omit checks for unknown values in attributes; the heuristics apply only to feature values that are known. Also, we abbreviate ANIMACY to ANIM and SEMANTIC CLASS to SEMCL.

satisfied. Some of the heuristics compare feature values for NP_i and NP_j, while others use additional non-clustering features. A *non-clustering feature* is a feature that expresses useful information but is not used during the clustering process. In this case, the non-clustering features are useful for the constraint generation phase. We use three main kinds of heuristics to generate cannot-link constraints: *compatibility*, *article*, and *semantic class* heuristics.

Compatibility constraints. Heuristics HC_1 through HC_4 check for compatibility between NP_i and NP_j. A mismatch on NUMBER or ANIMACY is an immediate indication that a cannot-link constraint should be created between the two noun phrases. The heuristics for handling the GENDER feature are a little more complicated, since a value of either can match any of masculine, feminine, or either.

Article constraints. Heuristics HC_5 through HC_7 encode rules that examine the articles used in NP_i and NP_j. The ARTICLE present in a noun phrase (definite, indefinite, or none) is a non-clustering feature, because it cannot be profitably used in clustering. We do not want the similarity of two noun phrases to increase simply because they match on this feature; the fact that two noun phrases both have indefinite articles does not make them more likely to be coreferent. Two other non-clustering features are used by these heuristics: PRONOUN (nominative, accusative, possessive, demonstrative, ambiguous, or none) and PROPER NAME (yes or no). The constraints generated by these heuristics are truly complementary to the data set D, since they are based on information not available in D.

We will use the following two sentences to illustrate the article heuristics.

(1) [The cat] climbed [my tree].
(2) [A cat] crossed [the road].

HC_5 asserts that a noun phrase with an indefinite article cannot link backwards to a noun phrase that is not a proper name or pronoun. In general, an indefinite article signals the start of a new coreference class; "A cat" in sentence 2 cannot link to "The cat" in sentence 1. HC_6 claims that a noun phrase with a definite article cannot link backwards to a noun phrase with no article, unless it is a proper name or a pronoun or the head nouns match. This is because noun phrases with definite articles tend to link to other definite or indefinite noun phrases. For example, "the road" cannot link to "my tree." Finally, HC_7 claims that a noun phrase with no article cannot link backwards to a noun phrase with an article, unless it is a proper name or pronoun. Once a noun phrase has been introduced with an article, it is very unusual to refer to it without an article. For example, after encountering sentence 1, we would not expect to later see just "cat" or any non-pronominal noun phrase referring to the same cat (unless it were a proper name).

Semantic class constraints. Heuristics HC_8 and HC_9 provide special handling for noun phrases classified as human or company. A mismatch on semantic class usually indicates that NP_i and NP_j cannot be coreferent. However, because many proper names are not present in WordNet, we allow humans and companies to potentially link to noun phrases of unknown classification.

Appositive constraint. Heuristic HM_1 indicates that an appositive noun phrase must be linked to its immediate predecessor in the document. An *appositive noun phrase* is a noun phrase that explains or further defines the preceding noun phrase. In English, appositive noun phrases are usually enclosed by commas or parentheses. For example, in the sentence "I spoke with Mr. Brown, my landlord, yesterday," "my landlord" is an appositive noun phrase that refers to "Mr. Brown." In such cases, we generate a must-link constraint between the two noun phrases. Note that it is possible for the output of this heuristic to conflict with some of the HC_h heuristics. For example, in the sentence "My teacher, a famous biologist, gave an excellent talk today," "a famous biologist" is an appositive noun phrase that refers to "my teacher." However, HC_5 will claim that the two noun phrases cannot be linked, because "My teacher" is not a pronoun or a proper noun, and "a famous biologist" begins with an indefinite article. Consequently, we permit HM_1 to override all of the HC_h heuristics, because it is an extremely reliable heuristic. We will later present a comparative evaluation of the accuracy of each heuristic.

Constraint generation. We have defined ten constraint heuristics. To generate the constraint sets, $Con_=$ and $Con_{\neq}$, we apply each heuristic to every pair of noun phrases in a given document. Each application of a heuristic to a pair of noun phrases may result in a must-link constraint, a cannot-link constraint, or no constraint (if the heuristic's conditions are not met). We take the transitive closure of the resulting set of constraints ($Con_= \bigcup Con_{\neq}$), as explained in Section 3.1.1, and provide the closed versions ($Con'_=$ and $Con'_{\neq}$) to COP-COBWEB.

Other constraint heuristics are possible. It is important to note, however, that this domain knowledge is being represented by hard constraints. Language is infamous for its exceptions to rules, and these heuristic rules are no exception. If a rule is violated by the actual text, but encoded as a hard constraint, then the clustering algorithm is forced to make an incorrect decision. We will analyze the accuracy of the constraint sets these heuristics create by comparing their decisions with the true (human-generated) coreference annotations. We will see that good performance on the coreference resolution problem can be obtained, even with the inherent restriction imposed by expressing knowledge as hard constraints. However, we expect that performance will be even higher, and that a larger range of domain knowledge can be encoded, if we instead make use of soft constraints to represent this knowledge. Soft constraints will be investigated more fully in Chapter 6.

5.2.1 Constraints on a Real Example

Let us once again return to the example text presented in Figure 5.1. Using the heuristics described in the previous section, and taking the transitive closure of the resulting constraints, we obtain a set of 111 pairwise constraints. Five of these constraints are presented in Table 5.3.

These constraints allow COP-COBWEB to avoid the mistakes that it makes without access to the domain knowledge encoded in the constraints. If we run COP-COBWEB with the indicated set of constraints, we obtain the output partition shown in Figure 5.3 (for clarity, singleton classes are not shown). The system identified 12 coreference clusters. The only

Table 5.3: Sample constraints generated on the text in Figure 5.1

Constraint	**Heuristic**
"first unmanned expedition" cannot link to "preparations"	HC_1
"the dawn" cannot link to "preparations"	HC_7
"the dawn" cannot link to "the Red Planet"	HC_9
"the Soviet Union" cannot link to "Human exploration"	HC_6, HC_9
"Sputnik I" must link to "first artificial Earth satellite"	HM_1
...	...

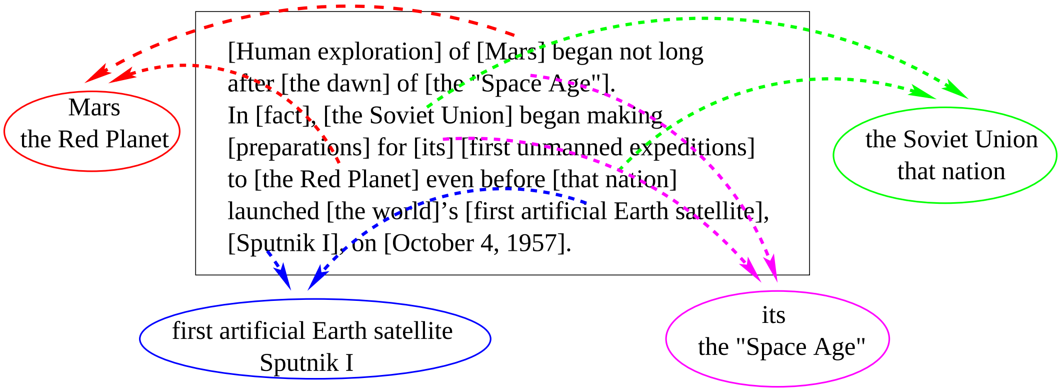


Figure 5.3: Coreference output for the text in Figure 5.1 when clustering with constraints

mistake it made was in deciding which noun phrase "its" refers to. Pronoun resolution is a notoriously difficult sub-problem in this task, especially for "it" (e.g., Dagan and Itai (1991) and others have done work exclusively on resolving "it"). Although the output does not perfectly correspond to the true coreference relationships in the text, it is a significant improvement over the output of the clustering algorithm without constraints. In the next section, we will see how to quantitatively evaluate this output.

5.3 Evaluation

To determine whether our encoding of linguistic knowledge and use of COP-COBWEB is an effective approach to the noun phrase coreference resolution problem, we need a method for evaluating the accuracy of the noun phrase partitions that we create. Unfortunately, there is little agreement in the NLP community about the best method for evaluating performance on this problem. We have identified eight different metrics that can be used, each of which assesses performance in a different way. A detailed analysis of the benefits and disadvantages of each metric is beyond the scope of this dissertation. However, we will describe all eight metrics briefly and present the results obtained using each metric. Although the numbers generated by the metrics differ, the general conclusions we can draw from them do not. In this section, we provide some terminology that will aid in interpreting our results. We introduce three common components in coreference evaluation metrics: *recall, precision,* and *f-measure.*

Recall and precision. Recall and precision are two criteria commonly used for evaluation in Information Extraction and Information Retrieval. In particular, several of the proposed noun phrase coreference metrics make use of them. *Recall* calculates how many of the desired items (in this case, coreference links) were correctly identified, and *precision* computes the percentage of items returned that are correct. In other words, recall is concerned with coverage, and precision is concerned with accuracy. Recall is defined as $\frac{N_c}{N_t}$ and precision is $\frac{N_c}{N_r}$, where N_c refers to the number of returned items that are correct, N_t is the total number of correct items that could be returned, and N_r is the number of (correct and incorrect) items that are returned.

F-measure. A trivial way to get 100% recall would be to put all of the noun phrases into one cluster, since this would definitely cover all of the correct coreference links (i.e., $N_c = N_t$). However, precision would be very poor since so many *incorrect* coreference links would be implied by that partition (in particular, precision would be $2 \cdot \frac{N_t}{n(n-1)}$). For evaluation purposes, these two numbers are usually combined into a single value called the *f-measure*:

$$\text{f-measure} = \frac{(\beta + 1.0)PR}{\beta P + R}. \tag{5.1}$$

where R and P refer to recall and precision, respectively. The parameter β varies between 0 and 1; it allows the evaluator to indicate the relative importance of the two values. We use $\beta = 1$, which gives equal weighting to recall and precision.

We calculate recall and precision by comparing two partitions of the input. The correct partition is referred to as the *key*, and the output being evaluated is the *response*. To evaluate the response, we must first decide what the "items" being returned are. One method is annotate coreference by linking each anaphoric noun phrase[2] to each of its coreferent noun phrases. If we designate the set of coreference links (over which a transitive closure has been taken) in the key as CL_k and the set of links in the response as CL_r, we then have

$$N_r = |CL_r| \tag{5.2}$$

$$N_t = |CL_k| \tag{5.3}$$

$$N_c = |CL_k \bigcap CL_r| \tag{5.4}$$

This intuitive notion of agreement corresponds exactly to the Jaccard method of comparing two partitions, which we will present in more detail shortly. First, we discuss the methods that are most commonly used by NLP researchers for evaluating coreference performance. Some methods tend to view coreference groups as *chains* instead of clusters (these views are equivalent, but they have led to different evaluation methods). Chains are specified by linking each noun phrase to its closest coreferent antecedent. Thus, a group of g mutually coreferent noun phrases will have $g - 1$ links under this scheme, while the same group would have $\frac{1}{2}g(g-1)$ links according to Equation 5.2.

[2] Anaphoric noun phrases are those that have a coreferent antecedent (not all noun phrases in a document do).

Coreference evaluation metrics. The most common method for evaluating coreference chains involves determining the minimum number of links that would need to be added (to eliminate recall errors) or removed (to eliminate precision errors) (Vilain et al., 1995). We refer to this approach as the **Vilain metric**. However, there are important drawbacks to this method which have been discussed by several authors. The first is that the Vilain metric only evaluates accuracy on noun phrases that are involved in some coreference relationship, so correctly determining that a noun phrase is *not* coreferent with any other noun phrase is unrewarded. Second, the Vilain metric can report inappropriately high performance numbers for some partitions, because it only calculates the minimum number of changes that would be needed to convert the response into the key. Bagga and Baldwin (1998) instead propose the **B-CUBED** evaluation method, which scores each noun phrase individually for recall and precision, including those not involved in a coreference relationship. Popescu-Belis and Robba (1998) suggest matching each noun phrase class in the response to a class in the key (**COR**). They count each noun phrase in the response that is not attached to any key class as a recall error and each noun phrase attached to an incorrect key class as a precision error. They also suggest a refinement to COR that identifies an exclusive (one-to-one) mapping from key classes to response classes (**XCOR**). Trouilleux et al. (2000) further expand on the ideas of Popescu-Belis and Robba by establishing different *specificity classes* of noun phrases (proper name, lexical head, and pronoun). These classes are used to improve the one-to-one mapping of key classes to response classes. We refer to their method as **TROU**.

Generic partition similarity metrics. Because we view the key and response as partitions of the same set of noun phrases, we can also make use of standard methods for computing the similarity of two partitions, P_1 and P_2. The most common of these is the **Rand** metric, which was described in Section 3.5.1. It views a partition as a set of $\frac{1}{2}n(n-1)$ pairwise decisions between items. We define values a, b, c, and d according to the following table:

		P_1	
		class(d_i) = class(d_j)	class$(d_i) \neq$ class(d_j)
P_2	class(d_i) = class(d_j)	a	b
	class$(d_i) \neq$ class(d_j)	c	d

The columns indicate decisions made by P_1 and the rows indicate decisions made by P_2. The diagonal entries represent situations where both partitions agree; off-diagonal entries represent decisions they disagree on. (This kind of table is also referred to as a *confusion matrix.*) Rand calculates agreement as

$$\text{Rand} = \frac{a+d}{a+b+c+d}. \tag{5.5}$$

In contrast, the **Jaccard** metric does not count decisions where both partitions placed the pair of items in distinct clusters (d):

$$\text{Jaccard} = \frac{a}{a+b+c}. \tag{5.6}$$

Table 5.4: COP-COBWEB noun phrase coreference results on the example text shown in Figure 5.1, as assessed by eight metrics. R stands for Recall, P stands for Precision, and F stands for f-measure.

	No constraints			**With constraints**		
Metric	(R	P)	F	(R	P)	F
Vilain	(75.0	23.1)	35.3	**(75.0**	**75.0)**	**75.0**
B-CUBED	(91.7	22.5)	36.1	**(91.7**	**93.8)**	**92.7**
COR	(75.0	7.7)	14.0	**(75.0**	**75.0)**	**75.0**
XCOR	(25.0	25.0)	25.0	**(93.8**	**93.8)**	**93.8**
TROU	(100.0	7.7)	14.3	**(75.0**	**75.0)**	**75.0**
Rand	58.3			**97.5**		
Jaccard	5.7			**50.0**		
FM	18.8			**67.1**		

Fowlkes and Mallows (1983) calculate a more complicated function of these values (**FM**):

$$\text{FM} = \frac{a}{\sqrt{(a+b)(a+c)}}. \tag{5.7}$$

This metric, like the Jaccard metric, ignores d (the number of pairwise decisions where both partitions placed the items in separate clusters). The sum $b + c$ is the total number of decisions that the two partitions disagree on. By dividing by $\sqrt{(a+b)(a+c)}$, the FM metric calculates a larger penalty for disagreements that are equally divided between b and c (i.e., the disagreements are spread out over the entire partition). Given a fixed number of disagreements $b + c$, FM is maximized when either b or c is zero (this means that a single cluster in one partition was split into multiple clusters in the other partition).

Using the example text in Figure 5.1, we evaluated the coreference system both with and without constraints using each of these eight metrics. The noun phrase bracketing and the feature generation were done by hand. The results are in Table 5.4. The first five metrics calculate recall and precision, while the latter three simply calculate agreement. Recall for the unconstrained output is generally fairly high, but precision is very low. In contrast, the use of constraints boosts precision to high levels, resulting in overall gains in f-measure. The three partition agreement metrics also show large improvements when constraints are used. Although each metric assesses performance in a slightly different way, they all agree that the output produced by COP-COBWEB when using constraints is superior to the output produced by regular clustering. This agrees with our informal impression of the output. The same eight metrics will be used in the next section to analyze the performance of COP-COBWEB on a larger body of evaluation texts.

Table 5.5: COP-COBWEB noun phrase coreference results on baseNP keys for the MUC-6 dry run evaluation, assessed by eight metrics

Metric	No constraints	Constraints	Pruned constraints
Vilain	50.1	**53.0**	53.9
B-CUBED	15.2	**74.3**	74.4
COR	22.3	**41.5**	42.0
XCOR	15.8	**69.9**	70.0
TROU	13.8	**28.9**	29.3
Rand	51.0	**96.9**	96.9
Jaccard	3.5	**20.3**	20.4
FM	15.1	**33.4**	33.5

5.4 Quantitative Evaluation of COP-COBWEB

To determine which of several coreference systems exhibits the best empirical performance, evaluation on a standard, common data set is crucial. The Message Understanding Conference (MUC) provides a common testing ground by putting together collections of manually annotated news articles. In addition to evaluating systems for their performance on other information extraction tasks, MUC offers a coreference resolution track. Each participant builds a coreference system, and then all systems are tested on the same data under the same conditions (MUC-6, 1995). This standardization makes it possible to run our system on the same data and to make precise comparisons.

Each MUC data set consists of a set of news articles. The noun phrases are not identified prior to processing, which means that noun phrase detection is an additional task that is required of each participant. The default scoring system uses the Vilain metric (Vilain et al., 1995) to compare the system's coreference output to the key document, which is annotated with the correct coreference information.

In our opinion, the lack of a standard set of input noun phrases is a big drawback to the MUC evaluations, because a coreference system may be unnecessarily penalized if the noun phrases it is working with differ from the noun phrases identified in the key document. To explicitly measure only a system's coreference ability, we would want to provide each system with the same input set of noun phrases. This has not been done, probably because it is difficult to separate a coreference resolution system from the parsing system it relies on to identify the noun phrases (and possibly their syntactic features). To better evaluate our system's coreference performance, we created a modified set of MUC key documents that use the same set of base noun phrases that our system receives as input. In the next section, our evaluations make use of these "baseNP keys." For quantitative comparisons with other systems (Section 5.5), we fall back on the default MUC keys.

5.4.1 Results on the MUC-6 Dry Run Data Set

The MUC-6 evaluation includes two data sets: the *dry run* data set, used for training and development, and the *formal* data set, used for a blind evaluation of the systems. Each

Table 5.6: Accuracy of heuristics as evaluated on the MUC-6 dry run data set

ID	**Heuristic**	**Accuracy**
HC_1	Number	98.95%
HC_2	Animacy	99.05%
HC_3 and HC_4	Gender	99.01%
HC_5, HC_6, and HC_7	Articles	99.57%
HC_8 and HC_9	Semantic class	99.20%
HM_1	Appositive	46.15%
All	All	98.96%
All but HM_1	No appositive	98.99%

data set contains 30 documents. The results of running our system on the MUC-6 dry run evaluation data sets are given in Table 5.5. We report f-measure for the first five metrics and agreement values for the others. By comparing the numbers in the second and third columns, we see that each of the metrics shows a significant improvement in performance when constraints are used. This improvement occurs despite the fact that the constraint heuristics only have access to shallow parsing information (base noun phrases). If our method were used in conjunction with a more sophisticated parser, we would be able to encode additional knowledge about coreference in the constraint sets, such as disallowing a coreference link between a subject and an object unless the object is a reflexive pronoun.

The smallest improvement occurs when the output partitions are assessed by the Vilain metric. It is known that this metric is often overly generous in scoring "bad" partitions (Popescu-Belis and Robba, 1998; Trouilleux et al., 2000). Popescu-Belis and Robba state that the Vilain metric is "too indulgent" because it calculates the *minimum* number of links that would need to be corrected to transform the response into the key. Visual inspection of the unconstrained output partitions suggests that this is probably happening. We will explain the contents of the fourth column shortly.

5.4.2 Accuracy of Coreference Constraints

We have noted that these hard constraints are generated by general heuristics and therefore are not guaranteed to be 100% accurate. It is important to get an idea of just how accurate the constraints are. We have calculated, for each heuristic, the percentage of constraints generated by that heuristic that agree with the true coreference annotations on the MUC-6 dry run data set. Table 5.6 presents the computed accuracy of each heuristic. The appositive heuristic is remarkably unreliable, which is a direct consequence of our poor ability to accurately detect appositive noun phrases. Consequently, we did not use this heuristic to generate constraints for the results in Table 5.5. When using all of the heuristics, the total accuracy is 98.96%; without heuristic HM_1, accuracy improves slightly to 98.99% (the effect of HM_1's inaccuracy is small because it is a heuristic that generates relatively few constraints).

We can see that the most reliable heuristics are conditioned upon the articles in the noun phrases. Surprisingly, the heuristic that prevents mismatches in NUMBER is the least reliable. This is probably due to our very simple assignment of singular and plural feature values to the noun phrases, because the heuristic itself is generally a very reliable one in English. As with the appositive heuristic, a more accurate classifier for this feature would improve the accuracy score for HC_1. Its accuracy is still high enough for it to be a useful source of constraints, so we will use it for our experiments.

5.4.3 Clustering with Pruned Constraints

Because the set of input constraints is not 100% accurate, it is difficult to determine whether errors made by COP-COBWEB are due to clustering mistakes or to incorrect information in the constraints. In general, we will not be able to automatically tell which of our heuristically-generated constraints are accurate and which are not. However, we are interested in determining how sensitive COP-COBWEB is to the accuracy of the input constraints. Because we have access to the true answer for each noun phrase, it is possible for us to prune the constraint set by removing each incorrect constraint. In addition to experiments with the heuristically-created $Con_=$ and $Con_{\neq}$, we also experimented with $PCon_=$ and $PCon_{\neq}$, their pruned counterparts:

$$PCon_= = \{(d, d') \in Con_= | d.label = d'.label\} \quad (5.8)$$

$$PCon_{\neq} = \{(d, d') \in Con_{\neq} | d.label \neq d'.label\} \quad (5.9)$$

The *label* attribute is the true class assignment of each noun phrase. The accuracy of $PCon_=$ and $PCon_{\neq}$ is guaranteed to be 100%. The fourth column in Table 5.5 presents the results we obtain when providing the pruned constraints as input to COP-COBWEB. In each case, clustering accuracy either stays the same or goes up slightly. We conclude that the 1% inaccuracy of the original constraint set, although definitely a factor when applying constrained clustering to this problem, should not have a large impact on future results.

5.4.4 Runtime Improvements

In Chapter 3, we mentioned that empirically, we have observed significant runtime savings when using COP-COBWEB with constraints. Let T_{const} be the time required to run COP-COBWEB with constraints and $T_{noconst}$ be the time required in the absence of constraints. In Figure 5.4, we plot $T_{const}/T_{noconst}$ for several documents from the MUC-6 dry run data set. We have only included the 12 documents for which $T_{const} > 1.0$; the remaining 18 documents were each processed in less than 1.0 second when using constraints. Runtime savings on these 12 documents range from 7 to 78%. Savings for the formal evaluation data set are even more dramatic; they range from 21 to 93%. These improvements in runtime occur because the constraints provided to COP-COBWEB exclude certain areas of the search space from consideration.

We have discussed COP-COBWEB in isolation by comparing its performance with and without constraints and demonstrating that the addition of pairwise constraints is a clear

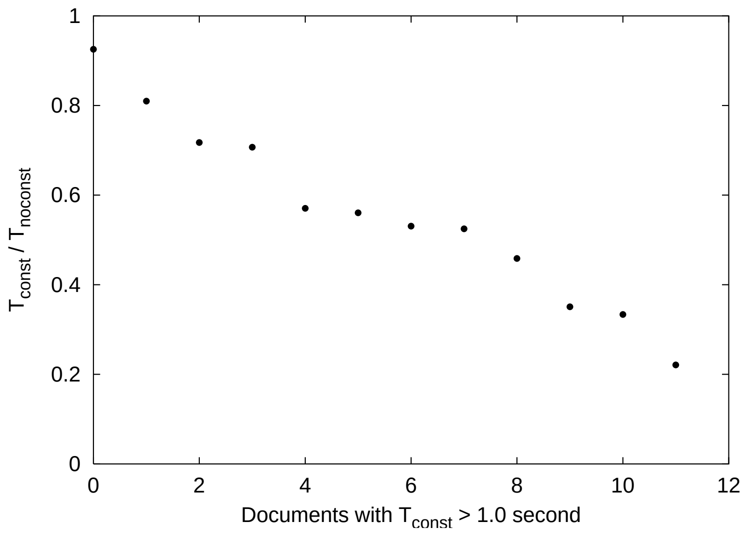


Figure 5.4: Runtime improvements observed when clustering with constraints on the MUC-6 dry run data set. T_{const} and $T_{noconst}$ are the runtimes for clustering with and without constraints, respectively. Only those documents with $T_{const} > 1.0$ second when using constraints are shown.

benefit to the algorithm, in terms of accuracy and efficiency. Next we will compare COP-COBWEB's performance to that of other state-of-the-art noun phrase coreference systems.

5.5 Quantitative Comparison to Other Systems

In this section, we compare our system's performance to other coreference resolution systems. We will continue to make use of the Vilain metric for quantitative comparisons. As mentioned above, several shortcomings of this metric have been identified (Popescu-Belis and Robba, 1998; Bagga and Baldwin, 1998; Trouilleux et al., 2000). However, the metric has been consistently used by participants in the MUC competitions, so it provides the only means for quantitative comparisons. It is important to keep in mind that, for these comparisons, the results do not just evaluate coreference accuracy. As discussed above, each system identified noun phrases in a different way, so errors in noun phrase identification are folded in with errors in coreference resolution in the output of any given system.

5.5.1 Results on the MUC-6 Formal Data Set

The formal MUC-6 evaluation included seven different systems. F-measures ranged from 40% to 65% (see Table 5.7). The best scoring system (Appelt et al., 1995) attained a recall of 59% and a precision of 72%, while the worst scoring system had a recall of 36% and a precision of 44%. Most coreference systems use a set of hand-coded filters or heuristics and algorithms that are highly specialized to the coreference task. The main exception is work that provides labeled training texts to a decision tree learner and outputs a classifier that can automatically label a pair of noun phrases as "coreferent" or "not-coreferent." We will discuss two simple baseline methods, summarize methods that do not use learning techniques,

Table 5.7: Summary of MUC-6 results and baseline performance on the MUC-6 formal data set, using the Vilain metric

Method	Recall	Precision	F-measure
Worst reported in MUC-6	36	44	40
Best reported in MUC-6	59	72	65
Baseline 1: All one class	80	34	47
Baseline 2: Match head noun	46	52	49

and then compare supervised and unsupervised coreference systems to our experiments with COP-COBWEB.

Baseline 1: All one class. The first baseline, "all one class," places all of the noun phrases into one large class; it claims that every noun phrase is coreferent with every other noun phrase in the document. This baseline is very simple, yet as shown in Table 5.7, its performance is surprisingly high according to the Vilain metric (this is one of the weaknesses of this metric). This baseline would attain a recall of 100% if we had access to the full set of noun phrases used by the human annotators on these documents. Since we are using our own set of noun phrases, the recall obtained by this baseline (80%) instead indicates the upper bound we can expect to obtain for recall. The remaining 20% of coreference links involve noun phrases that do not appear in our input data set. The missing noun phrases include nested noun phrases such as "Enron" in "the Enron executive," since Empire does not detect nested noun phrases, as well as noun phrases that are omitted due to parsing mistakes, such as incorrect part of speech tags. "All one class," as expected, attains a very low precision (34%).

Baseline 2: Match head noun. The "match head noun" baseline is slightly more sophisticated. It posits a coreference link between any two noun phrases that have the same head noun. This very simple rule also performs quite well, attaining 46% recall and 52% precision (49% f-measure).

Non-learning approaches. As mentioned above, most coreference resolution systems make use of hand-crafted filters and heuristics rather than techniques from machine learning. Azzam et al. (1998) evaluated their *focus-based* system, LaSIE, on the MUC-7 formal evaluation texts (MUC-7, 1998). LaSIE tracks the current conceptual *focus* at each point in the text as well as a list possible *actors*. Their system obtains an f-measure of 60.4%, but they also show that a very simple heuristic-based pronoun resolution method outperforms the focus-based method, achieving 62.3%.[3] The authors posit that this is because the focus-based method relies heavily on a sophisticated analysis of the text (to determine what the current focus is). This analysis is difficult to do well automatically, and errors in that

[3]These results do not appear in Table 5.7 because they are evaluated on a different data set.

Table 5.8: Quantitative comparison of noun phrase coreference systems that use machine learning techniques, on the MUC-6 formal data set using the Vilain metric

Method	Recall	Precision	F-measure
Supervised methods			
RESOLVE (decision tree) (McCarthy and Lehnert, 1995)	44	51	47
Decision tree, pruning non-coref examples (Soon et al., 2001)	59	67	63
Decision tree, pruning coref and non-coref (Ng and Cardie, 2002)	63	77	70
Unsupervised methods			
NP constrained clustering (Cardie and Wagstaff, 1999)	53	55	54
COP-COBWEB, no constraints	72	31	43
COP-COBWEB, with constraints	50	40	44

analysis cause errors in the coreference output. We have also observed that errors in our computed noun phrase features have an impact on coreference accuracy, but our system is likely to be less sensitive to this effect because we are computing simple features.

Supervised learning approaches. McCarthy and Lehnert (1995) and Aone and Bennett (1995) use supervised learning techniques to determine noun phrase coreference relations. Both systems train a C4.5 decision tree classifier (Quinlan, 1993). The resulting decision tree takes in two noun phrases and outputs whether or not they are coreferent. The decision tree trains on a data set consisting of several noun phrases *pairs* that are manually annotated as "coreferent" or "not coreferent." McCarthy and Lehnert create $\frac{1}{2}n(n-1)$ training instances by pairing each noun phrase with every other noun phrase. Aone and Bennett train only on noun phrase pairs where the second noun phrase is anaphoric. McCarthy and Lehnert participated in the MUC-6 evaluation, so we can compare our scores directly to theirs. On the formal evaluation corpus, their system, RESOLVE, obtained a recall of 44% and a precision of 51% (f-measure 47%). Table 5.8 presents these results as well as figures for the other learning methods we will discuss.

Although both systems demonstrate the ability to learn from the labeled examples, there are some drawbacks to this approach. The first is that coreference is a transitive relation, so making pairwise coreference decisions in isolation requires a later "coordination" step to resolve any conflicts. For example, if the system decides that NP_i and NP_j are coreferent, and NP_j and NP_k are coreferent, but NP_i and NP_k are *not* coreferent, the coordination step must decide how to resolve this so that transitivity is upheld. McCarthy and Lehnert mention this issue but do not discuss how they solved it. Others have suggested using a clustering algorithm as a post-processing step to ensure that the output is in fact a partition (Soon et al., 2001). Another problem with training on noun phrase pairs, identified by

Soon et al., is that the class distribution of the training set is highly skewed: the number of coreferent pairs is much smaller than the number of non-coreferent pairs. This situation is known to cause problems for supervised learners (Cardie and Howe, 1997). Soon et al. directly addressed this problem by intelligently pruning the training set to reduce the number of non-coreferent pairs. In evaluation on the MUC-6 formal data set, their method achieved a recall of 59% and a precision of 67% (f-measure 63%). Ng and Cardie (2002) further showed that intelligent pruning of the coreferent pairs as well can also improve performance, to an f-measure of 70% (see the second and third entries in Table 5.8).

Unsupervised learning approaches. In contrast to the above *supervised* approaches to noun phrase coreference resolution, we previously developed a custom clustering algorithm created specifically for this problem (Cardie and Wagstaff, 1999). This method takes in one parameter, a cluster *radius* threshold, and uses a custom distance measure to determine how close two noun phrases are. Linguistic constraints and preferences (soft constraints) are incorporated in the distance calculation; this system can therefore encompass a wider variety of domain knowledge than our hard constraints can. For the MUC-6 formal evaluation, this system obtained a recall of 53% and a precision of 55% (f-measure of 54%); we refer to this method as "NP constrained clustering" in Table 5.8. This system has the substantial benefit of not requiring any labeled training examples; it deduces coreference relationships solely on the "similarity" of the noun phrases under consideration. In addition, it out-performs the only other learning approach used in the MUC-6 evaluation (RESOLVE). However, it does not outperform the best-scoring system (as mentioned previously, this system had an f-measure of 65%). In addition, the decision tree methods previously discussed also perform significantly better.

Another approach that does not require human intervention is that of Dagan and Itai (1991). They focused exclusively on performing coreference for the pronoun "it," arguably the most difficult noun phrase to resolve. For each occurrence of "it," they collected co-occurrence statistics from a large body of texts that were parsed, but not labeled with coreference information. These statistics were computed for all subject-verb and verb-object pairs and used to filter out implausible antecedents. The approach looks promising, but unfortunately was only evaluated on 74 occurrences of "it" (not from the MUC texts), so it is difficult to compare to other work here. For those 74 examples, performance using their statistical filter improved from 64% to 74% (in terms of how many "it"s were assigned to their correct nearest antecedent).

5.5.2 Comparison of COP-COBWEB to Other Coreference Systems

For these data sets, and using the Vilain metric, COP-COBWEB achieves a recall of 72% and precision of 31% (43% f-measure) without any constraints. Performance improves with the inclusion of constraints to an f-measure of 44%. Because most of the constraints are cannot-link constraints, we expected precision to improve when constraints are used. It is also unsurprising that recall goes down, due to inaccuracies in the feature value computations and in the constraint heuristics themselves. We will explain why the Vilain scores obtained by COP-COBWEB appear low and then discuss the advantages of the COP-COBWEB approach to this problem.

Factors influencing COP-COBWEB's Vilain scores. There are several reasons that we do not observe higher f-measure values for COP-COBWEB, most of which arise from our experimental setup and are independent of the algorithm used. The most significant factor, which we have previously mentioned, is the mismatch between the set of noun phrases we obtain from Empire and the set of noun phrases used in the key documents provided by MUC. Since 20% of the coreference links in the keys involve noun phrases we do not have access to, there is a large impact on recall. Secondly, we have also noted the drawbacks of using the Vilain metric for evaluating noun phrase coreference. Although this metric reports only a small improvement in COP-COBWEB's performance compared to clustering without constraints, the other metrics we have used show much larger improvements (see Table 5.5). Finally, our feature values are probably not very accurate representations of the noun phrases. These three difficulties would arise regardless of which algorithm we used, so their contributions to the low f-measures reported do not tell us anything specifically about COP-COBWEB's abilities.

Other factors are more directly relevant to an assessment of COP-COBWEB. We have previously noted that the constraints used by the algorithm are imperfect, due to their heuristic generation. However, our experiments with pruned constraint sets have shown that this inaccuracy is unlikely to have a large impact in our results. It is also the case that supervised methods often out-perform unsupervised methods when applied to the same task. The supervised methods have specific examples of the desired classifications. In contrast, unsupervised methods simply seek a "good" organization of the data, which may or may not correspond to the specific concept encoded in the labels used for evaluation. COP-COBWEB is a hybrid algorithm; it has more guidance than a purely unsupervised method, in the form of constraints, but it does not have access to explicitly labeled examples of coreferent relationships.

Finally, we expect that our set of constraint heuristics is incomplete. Each heuristic generated constraints that improved the performance of COP-COBWEB, but it is likely that there are other applicable rules about coreference that could (and should) also be added to the set of heuristics. We have demonstrated that COP-COBWEB can effectively apply the constraints it is given; augmenting the constraint set with additional (reliable) information is likely to improve performance further.

Advantages of using COP-COBWEB for coreference resolution. Although the Vilain scores for COP-COBWEB do not exceed those of other algorithms, there is a significant advantage to using COP-COBWEB on this problem. Domain knowledge from a variety of sources can be easily encoded using our constraint formulation. For example, although the Vilain scores for both baselines are higher than those reported for COP-COBWEB, we could emulate the performance of either one by using a suitable set of constraints. Baseline 1, which places all of the noun phrases into a single class, can be duplicated by creating a must-link constraint between each pair of consecutive noun phrases in the document. Baseline 2, which connects each pair of noun phrases with identical head nouns, can be duplicated by creating a must-link constraint between noun phrases with matching head nouns and a cannot-link constraint between every other pair of noun phrases.

In addition, the use of soft constraints, rather than hard constraints, should further improve the performance of this algorithm. Because we could only encode knowledge as hard constraints, we were forced to omit information that functions as a "preference" rather than a "constraint." For example, HC_5 claims that a noun phrase with an indefinite article, such as "a cat," cannot link backwards to a noun phrase that is not a proper name or a pronoun. However, HC_5 is violated by a sentence such as "[Her cat] is [a cat] that literally climbs walls." Consequently, we would like to indicate that HC_5 is *usually* but not *always* reliable. In Chapter 6, we will see how soft constraints, which allow the specification of a *strength* for each constraint, can be formulated and integrated by a constrained clustering algorithm. We have reason to believe that this would lead to improved Vilain scores, due to our previous experience with a clustering algorithm specialized to the problem of coreference resolution (Cardie and Wagstaff, 1999). That algorithm used the same set of features that we have used for our experiments with COP-COBWEB, but it was able to accommodate preferences as well as constraints. The critical reader then asks why one might prefer COP-COBWEB to that algorithm, if we simply duplicate its performance on this problem. The answer highlights another significant advantage of COP-COBWEB: its generality. COP-COBWEB can be applied to any clustering problem, not just that of noun phrase coreference.

5.6 Summary

In this chapter, we applied the COP-COBWEB constrained clustering algorithm to the problem of noun phrase coreference resolution. This challenging problem is far from being considered "solved" by the NLP community, and a diverse array of techniques are being applied to it. We described how background linguistic knowledge about the coreference relation can be encoded as a set of instance-level constraints and provided as input to COP-COBWEB. The results of eight different metrics agree that this is an effective way to improve performance on this problem. We also compared COP-COBWEB's performance to other methods specifically designed with the coreference problem in mind. The best available coreference system, which uses supervised learning techniques and intelligent pruning of the training set, out-performs COP-COBWEB, which is not surprising. We do find, however, that the general COP-COBWEB algorithm, when specialized to this task through the use of constraints, provides an improvement over not using constraint information. In addition, COP-COBWEB allows for the encoding and use of a diverse set of constraints. We expect that applying COP-COBWEB to this problem with soft constraints will demonstrate further improvements.

CHAPTER 6
SOFT CONSTRAINTS AND APPLICATION 3: SPECTRAL ANALYSIS

This chapter presents a real-world[1] problem involving spectral analysis of telescopic observations of Mars. We have previously explored the use of clustering with hard constraints. This task is exploratory in nature, and because our knowledge of Mars is itself approximate and incomplete, encoding that knowledge using *soft* constraints is a necessity. The first major contribution presented in this chapter is the development of a soft constrained clustering algorithm. Our analysis of real Mars data produces the second major contribution: the preliminary conclusions we can draw based on constrained clustering of this data simultaneously confirm previous discoveries and suggest novel hypotheses about the planet. Our techniques are demonstrated to be useful for scientific discovery of new knowledge.

After describing the problem of spectral analysis (Section 6.1), we survey other approaches that use clustering on this problem (Section 6.2). We analyze the performance of the basic k-means clustering algorithm in Section 6.3. In Section 6.4, we present a soft constrained k-means clustering algorithm, and in Section 6.5 we discuss the soft constraints we used in our experiments. We conclude with a discussion of our experimental results in Section 6.6.

6.1 Spectral Analysis

Spectral analysis, or *spectroscopy*, attempts to infer the composition or mineralogy of an object based on measurements of how it interacts with light (electromagnetic radiation) (Hunt, 1980; Bell, 1997). In this case, we focus on reflectance spectroscopy, in which we analyze the difference between the light incident on the object and the light that it reflects. The ratio of reflected to incident light, after being corrected as much as possible for viewing geometry and other experimental factors, is referred to as *radiance* or *albedo*. Due to the arrangement of atoms and their electrons, different elements and compounds produce reflectance spectra with different features. These features include *absorption bands*, which are contiguous regions in the spectra with relatively low radiance, indicating that the object is absorbing incident light strongly at those wavelengths. Other interesting features include the slope of the radiance between specific wavelengths and the location of radiance peaks and valleys.

Once radiance observations have been collected, it is possible to compare an observed spectrum to that of several different minerals or other compounds, as measured in the laboratory, to determine the composition of the remote object. Of course, for real observations, a given region on a remotely observed object is unlikely to be composed purely of one element or compound. It is more common to encounter a mixture, and this means that characteristic spectral features may be muted or absent. Nevertheless, it is often still possible to identify major constituents of an object based on the observed spectrum by performing matching to laboratory spectra (Adams and McCord, 1969).

Infrared observations. We do not restrict our observations to the visible part of the electromagnetic spectrum. The window of visible wavelengths is quite narrow (300 – 700

[1] Real *other* world.

nm), and distinguishing characteristics often appear beyond this limited scope, especially in the near-infrared (700 – 5000 nm). For example, an Fe^{3+} absorption band for various minerals occurs between 850 and 950 nm. Therefore, observations in the near-infrared are very important for remote sensing and for identifying the chemical composition of a remote body (Bell, 1997).

Manual spectral analysis. Data collected by remote sensing is most commonly interpreted and analyzed manually by experts in the field. Generally, this is done by creating radiance ratio images and scatterplots and then analyzing the results visually to construct threshold values that will distinguish homogeneous groups from each other. A *radiance ratio image* is created by calculating the ratio of radiance at two specified wavelengths for each pixel and then scaling the resulting values to produce an output greyscale image. *Scatterplots* are created by plotting pixels in two dimensions, usually either the radiance values at two wavelengths or a radiance value versus a related ratio value. Often, these values will be binned and colored according to the number of pixels that fall into each bin, so that highly-populated areas will be more immediately apparent.

Both of these analysis tools have been instrumental in the discovery of new knowledge about Mars. McSween Jr. et al. (1999) selected "representative" spectra from rock observations obtained by the Mars Pathfinder Lander and used the results of both techniques to conclude that there are four major spectral classes of rocks present. Bell et al. (2000) performed a detailed manual analysis of the dust and soil found at the Pathfinder landing site and determined that there are two distinct "dust" and five "soil" classes. However, by themselves these tools are only an intermediate step in the analysis process. A human expert is called upon to interpret the ratio images and scatterplots.

Consequently, manual analysis has limited scalability. As data sets get larger and larger, there is a real need for automated methods to assist in reducing the number of individual items that a human must examine. Some researchers have already experimented with using techniques from statistics and machine learning to augment their manual analyses of astronomical data. As reported by Lillesand and Kiefer (1987), this work has involved both supervised and unsupervised approaches.

Supervised learning for spectral analysis. The most popular supervised learning technique for spectral analysis is the neural network, although decision trees have also been used (Bertin, 2000). Gulati et al. (1995) applied a neural network to the problem of stellar classification (into known star categories). They selected 55 spectra with known classifications for training and then applied the learned neural network to 158 test spectra. Merényi et al. (1996) used a neural network to analyze spectroscopic observations of Mars; they manually identified ten classes in the data, then labeled five to ten sample training spectra for each class. After training the neural network on these examples, they were able to apply it to the full data set and classify each pixel into one of the ten classes.

Supervised learning can be of help in processing large data sets because human intervention is needed only for the initial labeling of the training set. However, supervised learning does not completely solve the problem of scalability. Gulati et al. note that their approach would be substantially improved by having access to a larger training set, and ideally, one that is more consistent with the items in the test set. It is not easy in general to estimate the number (or identity) of items required for the training set to guarantee good performance on unseen items. More importantly, supervised methods are useful when the goal is to apply an existing categorization to new data. We are in a different situation; we do not know precisely what categorization of Mars observations will yield the most information.

To summarize, the motivating hypothesis behind this work is that an analysis of spectral observations can identify geologically or compositionally interesting regions on Mars. This task is by definition an exploratory one. The volume of data being collected makes manual analysis impractical. Therefore, we turn to automated analysis methods as our tool of choice. Supervised methods would require us to specify examples of what we want to find in the data, while unsupervised clustering algorithms instead adapt to patterns actually present in the data set and therefore are less "biased." In addition, clustering algorithms can rapidly process large data sets and produce summary output (clusters) that point the human analyst towards specific subsets of data that have interesting characteristics. In the next section, we will discuss previous clustering approaches to the problem of spectral analysis.

6.2 Other Clustering Approaches to Spectral Analysis

We begin with a review of various clustering analysis techniques that have been applied to spectral data. Next, we discuss methods that can incorporate additional information. These methods have largely been limited to working with image contiguity information (spatial relationships between pixels).

6.2.1 Clustering Spectral Information

Clustering has been demonstrated to be of use on a variety of astronomy problems. It has primarily been used to automatically classify observations into meaningful groups or to perform image segmentation.

Classification. Djorgovski et al. (2000) discuss the utility of clustering algorithms for analyzing large sky surveys. Yoo et al. (1996) used COBWEB/95, an enhanced version of COBWEB (Fisher, 1987) that can handle continuous attributes, to induce a clustering hierarchy on 33,021 sky objects from the Second Palomar Observatory Sky Survey. The algorithm's taxonomy of sky objects successfully separated stars from galaxies, even though no training data was provided and this particular goal was not specified to the algorithm. de Carvalho et al. (1995) applied Autoclass, a Bayesian clustering algorithm (Cheeseman et al., 1988), to the same data set and obtained a partition of the objects into four clusters. Manual post-analysis of the clusters determined that they represented meaningful categories: stars, galaxies with bright central cores, galaxies without bright cores, and stars with "fuzz" around them.

Image segmentation. For a data set that includes observations taken at d spectral bands, we can view each pixel as a data vector, containing d values. The distance between two pixels can be defined as the Euclidean distance between their respective d-valued spectra. We can then apply common clustering algorithms to organize the data set into subregions that are internally similar and distinct from each other.

A different approach is to cluster the pixels in an image based on a histogram rather than the individual points themselves (Narendra and Goldberg, 1977; Schowengerdt, 1983; Wharton, 1983). Their algorithm effectively discretizes the feature space into a set of cells, where each cell is annotated with a *frequency*, i.e., the number of data points that fall into that region. Clusters are identified by seeking cells with relatively high frequency counts. This approach is computationally efficient and does not need to make any assumptions about the probability distributions of the clusters (e.g., Gaussians). However, these benefits come at the expense of losing fine discriminability, and as with any discretizing operation, the quality of the results is dependent on the choice of number of cells and their placement in feature space.

6.2.2 Incorporating Spatial Information

Although spectral information is very important for remote sensing analysis, it is also desirable to include other relevant information that is available. By far, the largest body of work on spectral analysis that uses information beyond just the spectral features is that which attempts to incorporate the spatial information inherent in a two-dimensional image. Each pixel is described by observations at d spectral bands, but it is a common hypothesis that the relative position of the pixels is also important for classification purposes. In other words, the identity of neighboring pixels may also be important to the classification of a given pixel. This bias is supported by the fact that, for real remote sensing imaging systems, several experimental factors (e.g., diffraction and internal scattering) cause light from a given location in the scene to spread out into neighboring pixels.

K-means variants. One approach is to use a Bayesian update rule inside the iterative k-means algorithm, and to weight the priors so that a given pixel has a predisposition for receiving the same classification as its neighbors (Montolio et al., 1992). Alternatively, Theiler and Gisler (1997) modified the objective function used by k-means to take into account the overall contiguity of the partition. We will adopt this approach for our own development of a soft constrained clustering method, but we will allow the encoding of domain knowledge from a variety of sources, not just spatial contiguity.

Spatial information as additional features. Soh and Tsatsoulis (1999) used the regular COBWEB clustering algorithm to perform image segmentation of images taken at a single wavelength. To do this, they first performed an initial segmentation of the image into regions by automatically finding threshold values such that all pixels within a region have intensity values below the calculated threshold. They then calculated additional spatial and textural features for each such region by examining, for example, the region's spatial *variety* (synonymous with discontiguity). Next, they used the COBWEB clustering algorithm to condense these regions into large-scale clusters in the image.

The authors applied this technique to several Earth images, and the approach demonstrates apparently good results visually. However, they provide limited assessment, despite the fact that they report class labels for each of the clusters obtained from COBWEB.

Quantitative assessment is limited to a comparison of the size of the resulting clusters to the amount of coverage in each real class indicated by a human expert on the same images; we do not know how much these clusters agree (spatially or spectrally) with the expert classification. In addition, this approach has only been applied to images taken at a single wavelength. For multi-spectral data, the authors speculate that their technique could be applied to each wavelength separately, and the results could be fused together in an unspecified manner. In contrast, we will accept multi-spectral input directly into a single run of the algorithm and automatically produce unified output.

Neighbors encoded as additional features. Masson and Pieczynski (1993) developed a stochastic version of the EM clustering algorithm (Dempster et al., 1977) by constructing additional features for each pixel that incorporated a single neighbor's values. Roberts et al. (1996) compared clustering multi-spectral data on the basis of the spectral information alone to clustering the same data with additional spatial information. The original data set, a Landsat image, contains information from seven spectral bands. Spatial information was incorporated by augmenting each pixel with a set of seven additional features for each of its eight neighbors. The biggest disadvantage of this method is the dramatic increase in dimensionality of the data set; in this case, dimensionality rises from $d = 7$ to $d = 63$. The authors compensated by running a principal components analysis (PCA) to provide a projection of the data down into a two-dimensional space. The PCA and projection were done separately on the original data ($d = 7$) and on the spatio-spectral data ($d = 63$). This transformation of the data allowed the authors to apply a variety of clustering algorithms (to each set of two-dimensional data) without modifying the algorithms themselves. Evaluations of the contiguity of the resulting partitions show that the contiguity is higher when using the spatio-spectral data set.

One of the algorithms Roberts et al. used was the contig-k-means algorithm developed by Theiler and Gisler (1997), which incorporates contiguity information directly into the k-means algorithm as previously described. Thus, when this algorithm was applied to the spatio-spectral data, it was in a sense receiving the same information twice (in the data features and in the spatial relation, which it was able to directly observe). Not surprisingly, the difference in its performance on the two data sets (in terms of contiguity) was very small. However, the variance of the partition it produced on the spatial data was much higher than that of its partition of the spatio-spectral data, which indicates that the latter clusters were a "tighter" fit to the data. On the other hand, it is unclear whether these numbers can be directly compared, since each data set is using a different pair of (PCA-reduced) features to represent each pixel.

6.2.3 The Need for Constraints in Remote Sensing

Although automated tools are useful for summarizing large quantities of data, there is a need for such tools that can also incorporate additional domain knowledge into their analyses. As we have just shown, most previous work with specializing clustering algorithms for the problem of spectral analysis has focused solely on methods for incorporating spatial contiguity. We claim that there is a much richer class of relevant information that can, and should, be used when performing a clustering analysis of spectral data. For example, Pieters

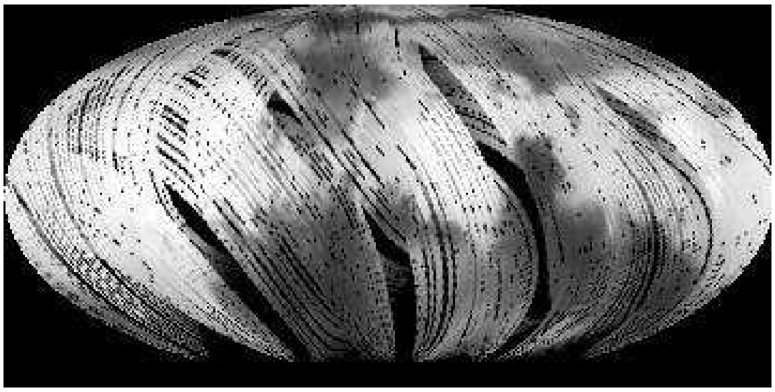


Figure 6.1: Mars observed by STIS at 907 nm

et al. (1996) warn of the dangers when blindly attempting to find the best numerical fit to data. Unsupervised application of their mixture modeling approach to determining mineralogical composition of remote objects often came up with physically implausible results. Therefore, they recommend the use of "additional constraints or assumptions" to restrict solutions "to be consistent with laboratory trends." Further, Djorgovski et al. (2000) warn that "a blind application of the commonly used clustering algorithms in such real-life cases could produce some seriously misleading or simply wrong results."

Exploratory analysis should be guided and constrained as much as possible, to ensure that the results it produces are reasonable from a scientific view. Therefore, our goal is to develop a method that can effectively encode and apply domain knowledge to improve its analysis of spectral data.

6.3 K-means Performance on Spectral Analysis

In the next section, we will show how the basic k-means algorithm can also be transformed to handle soft constraints and argue that this is a powerful and natural way to encompass domain knowledge from a variety of sources. However, we first describe the data set used in our experiments. In addition, we will discuss the performance of the regular k-means algorithm on this problem and show examples of its output.

6.3.1 Experimental Methodology

The data set under consideration is composed of multiple observations of the planet Mars by the Hubble Space Telescope (HST) in April and May, 1999. We will refer to it as the STIS[2] data set. The planet was observed at 1024 different wavelengths, resulting in 1024 images of Mars, each at a resolution of about 20×80 km per pixel. The images at each wavelength were then map-projected onto a 1200×600 pixel latitude/longitude grid to preserve this spatial resolution. To simplify the presentation of our analysis, we will show the results of working with a smaller version of this data set, which was subsampled to

[2]STIS stands for "Space Telescope Imaging Spectrograph," the HST instrument used to collect the data.

Table 6.1: Derived features for the STIS data set

Feature	**Description**	**Calculation**	**Min**	**Max**
FL	"flatness" of spectrum	$\frac{R_{528}}{R_{762}}$	0.38	1.02
BD_{670}	670 nm band depth (Fe^{3+})	$1 - \frac{R_{667}}{.58*R_{608}+.42*R_{747}}$	-0.061	0.24
BD_{860}	860 nm band depth (Fe^{3+})	$1 - \frac{R_{879}}{.68*R_{801}+.32*R_{1016}}$	-0.02	0.07
BD_{950}	950 nm band depth (Fe^{2+})	$1 - \frac{R_{947}}{.26*R_{747}+.74*R_{1016}}$	-0.07	0.06
SL	800 – 1000 nm slope	$R_{1016} - R_{801}$	-0.05	0.01

360 × 180 pixels. We also reduced the number of distinct wavelengths under consideration to 26, by performing a sliding Gaussian average on each full spectrum and recording every 40th value. Our final data set, then, contains 64,800 pixels, with a dimensionality of 26. The wavelengths span 528 to 1016 nm.

Figure 6.1 shows the entire planet when observed at a single wavelength (907 nm). The image is a Mollweide equal-area projection with the north pole at the top, south pole at the bottom, and 0^o longitude in the center of the map. Brighter areas indicate a higher observed radiance; this image has been contrast-enhanced to more clearly delineate the light and dark regions. Our data set is an image cube composed of 26 such images.

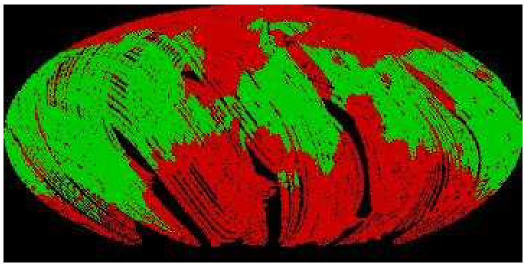
Derived features. In addition to the 26 wavelengths available in the downsampled data set, we computed five additional *derived* features. This choice of additional features was motivated by the features reported as most useful for classifying scenes obtained by the Mars Pathfinder Lander (Bell et al., 2000). The list of derived features is given in Table 6.1. R_λ refers to the radiance observed at wavelength λ.

The first feature of interest is a measure of how flat each pixel's spectrum is (FL), which is calculated by taking the ratio of the radiance at the lowest available wavelength to the radiance at the average "peak" of the STIS spectra. Most of Mars is dominated by areas with very strong absorption at lower wavelengths, which is what gives the planet its characteristic reddish color. The very red regions will have an FL value well below 1.0, while other areas will be distinguished by a FL value near 1.0. For this data set, FL varies from 0.38 to 1.02.

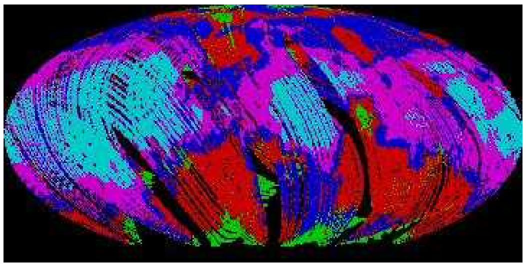
Since we are interested in determining the chemical composition of areas on Mars, we also include features that are sensitive to specific absorption bands. A band centered around 950 nm indicates the presence of Fe^{2+}, while bands near 670 and 860 nm indicate the presence of Fe^{3+} (see Adams, 1974; Morris et al., 1985). (The precise wavelength values differ slightly because we are working with a discrete set of 26 wavelengths, and in each case we selected the closest available value.) The band depth at wavelength λ, BD_λ, is calculated as:

$$BD_\lambda = 1 - ER_\lambda \tag{6.1}$$

where ER_λ is the "expected" radiance at λ in the absence of the band. We determine the "expected" radiance by calculating the continuum value: we compute a linear interpolation between the radiance values at two nearby wavelengths outside of the band.



(a) $k = 2$



(b) $k = 5$

Figure 6.2: Two classifications of Mars by the k-means algorithm

Finally, we also know that the spectral slope between 800 and 1000 nm is important for distinguishing Martian spectra (see McSween Jr. et al., 1999; Bell et al., 2000). We calculate this value by subtracting the radiance at 801 nm from the radiance at 1016 nm. Technically, this slope (SL) should be divided by the wavelength separation (215 nm), but since this value is constant for all spectra, we omitted that additional step.

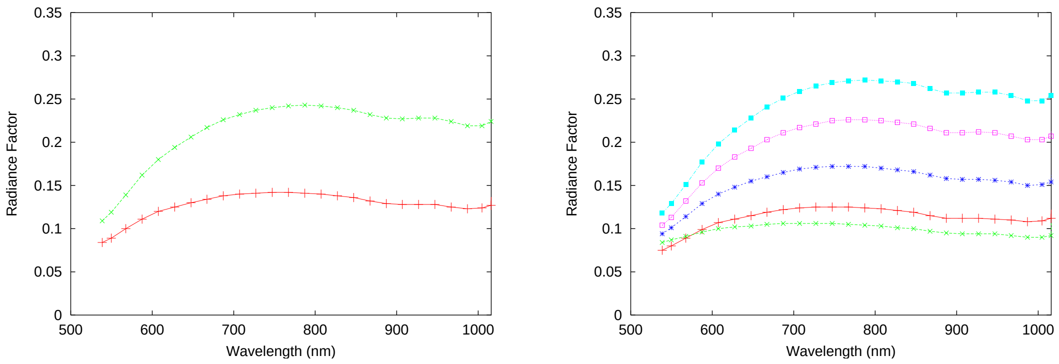
6.3.2 K-means Results

To provide a basis for later comparisons, we ran the regular k-means algorithm on the STIS data set. The output provided by this algorithm indicates what can be learned from an automated analysis of the data set without any specific domain knowledge. The input data includes the 26 spectral observations and five derived features for each pixel.

Before describing the results we obtained with k-means, we will first justify our choice of k-means for this problem. We do not claim that k-means is the only useful clustering algorithm for this problem. It may also be instructive to analyze the data using, for example, an algorithm which attempts to find a good value for k on its own, rather than having it specified as input. However, k-means is useful as an exploratory tool for spectral analysis precisely because we can control the number of clusters it generates. This allows us to examine Mars at different levels of generality. As we will show, the output at different k values provides us with different insights about the planet.

Figure 6.2 shows the output of the k-means algorithm for $k = 2$ and $k = 5$. In these images, each pixel is colored according to its classification by the clustering algorithm (the color values themselves are not meaningful). Black regions correspond to non-planet regions or to missing data. For $k = 2$, the planet is divided into two regions that correspond directly to its characteristic light and dark areas (cf. Figure 6.1). With larger values of k, such as $k = 5$, the planet splits into finer distinctions.

An examination of the average cluster spectra reveals that the cluster separations are dominated by simple albedo differences. Figure 6.3 plots, for each cluster identified by k-means, the average spectrum for that cluster. The x-axis is wavelength in nm, and the y-axis is the radiance factor, I/F. This value is defined as the observed radiance (I) divided by the radiance of sunlight reflected off a "perfect" diffuse reflector at the same distance as Mars (F). The colors used for each spectrum correspond to the colors used in Figure 6.2. Visually,



(a) $k = 2$ (b) $k = 5$

Figure 6.3: Average cluster spectra for the partitions shown in Figure 6.2

the average spectrum in each cluster is very similar to the other cluster spectra in shape, and brightness is the major distinguishing factor. This is not surprising; a principal components analysis (PCA) of this data set (and other Mars observations in general) confirms that the first principal component is albedo and that this component typically explains 90–95% of the variance in spectral observations of Mars (Bell, 1992). An exception to this general rule is the cluster identified by green x's in Figure 6.3b; its shape differs in important ways from the rest of the average cluster spectra. We will return to this interesting exception in Section 6.6.

6.4 Constrained Clustering With Soft Constraints

We have noted in previous chapters that some kinds of domain knowledge are perhaps better encoded as soft constraints (or *preferences*) rather than as hard constraints. This is particularly true for our spectral analysis of the Mars data. We do not yet know precisely what can be found in this data set, but we have some general knowledge about the problem domain that should be of use to a clustering algorithm. Therefore, before proceeding to describe the constraints that we used for this problem, we first describe our formulation for soft constraints and present an algorithm that can use them, SCOP-KMEANS.

6.4.1 Soft Constraints

Previously, we defined two kinds of hard constraints, must-link and cannot-link (see Section 3.1.1). Each constraint was specified as a relationship between two data items. For soft constraints (preferences), we will augment each relationship with an additional *strength* factor, s, that indicates how reliable the constraint is. This single constraint formulation ($\langle d_i, d_j, s\rangle$) subsumes both soft and hard constraints. The absolute value of s ranges from 0 to 1, with higher values indicating a stronger constraint. We also attach a sign to the strength; negative values indicate a preference against being grouped together, while positive values indicate a preference towards being grouped together. A constraint such as $\langle d_i, d_j, 1\rangle$ is equivalent to a must-link constraint, and $\langle d_i, d_j, -1\rangle$ is equivalent to a cannot-link constraint ($\langle d_i, d_j, 0\rangle$ is a "don't-care" statement).

Soft constraint closure. The use of soft constraints requires a modification to how we compute the constraint closure. The graph of the constraint relation is weighted; each edge (constraint) has a weight determined by its strength. In Section 3.1.1, we defined the closure of a set of hard constraints as follows:

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

Because the edges are now weighted, each derived constraint must also have a strength assigned to it. We calculate this *conservatively* as the minimum (absolute) strength of the two constraints that imply the new constraint. The closure for a set of soft constraints is defined in this way:

$\forall i, j, k$: given		produce
$\langle d_i, d_j, s_1\rangle$	$\langle d_j, d_k, s_2\rangle$	$\langle d_i, d_k, min(s_1, s_2)\rangle$
$\langle d_i, d_j, s_1\rangle$	$\langle d_j, d_k, -s_2\rangle$	$\langle d_i, d_k, -min(s_1, s_2)\rangle$
$\langle d_i, d_j, -s_1\rangle$	$\langle d_j, d_k, s_2\rangle$	$\langle d_i, d_k, -min(s_1, s_2)\rangle$

where s_1 and s_2 are the (positive) strength values for the corresponding constraints. If both constraints are positive (preference towards linking), then the derived constraint also has a positive strength, which is the minimum of s_1 and s_2. If one of the contributing constraints is negative, then the derived constraint also has a negative strength, which is the minimum of s_1 and s_2. As before, if both contributing constraints are negative, then we cannot conclude anything directly from them, so we do not create a derived constraint.

6.4.2 The SCOP-KMEANS Algorithm

We have chosen k-means as our prototype for the development of a soft constraint clustering algorithm. As discussed above, this is a useful algorithm for the spectral analysis problem, so it is very relevant to the goals of this chapter. A similar transformation is possible with the COBWEB clustering algorithm, but we will not present those details explicitly here.

We achieve soft constrained clustering with k-means by modifying its objective function to incorporate a real-valued penalty for violating constraints. This penalty is proportional to the strength of the violated constraint(s). SCOP-KMEANS[3] calculates CV, the maximum strength of the violated constraints, if any. See Table 6.2 for a full listing of the SCOP-KMEANS algorithm; once again the changes from the basic k-means algorithm are shown in bold. The objective function, f, combines CV with var (variance) in the following way:

$$f(C_1 \ldots C_k) = \frac{var}{1 - CV} \tag{6.2}$$

[3] "S" stands for "soft."

Table 6.2: SCOP-KMEANS algorithm

SCOP-KMEANS(number of clusters k, data set D, preferences $Pref\ \langle a,b,s\rangle$ where $(a,b) \subseteq D \times D$ and $s \in [-1,1]$)

1. Let $C_1 \ldots C_k$ be the initial cluster centers.
2. For each instance d in D, assign it to the cluster C_j which will minimize the following objective function:
$$f(C_1 \ldots C_k) = \frac{var}{1 - CV}$$
where var is the variance of the partition and CV is the constraint violation value; $CV :=$ CONSTVIOL$(d, C_1 \ldots C_k, Pref)$.
3. Update each cluster center C_i by averaging all of the points $d_j \in C_i$ that have been assigned to it.
4. Iterate between (3) and (4) until convergence.
5. Return the partition $\{C_1 \ldots C_k\}$.

CONSTVIOL(data point d, partition $C_1 \ldots C_k$, preferences $Pref$)

1. Let $CV_{max} := 0$, $nConst := 0$, $nViol := 0$.
2. For each $\langle d, d', s\rangle \in Pref$:
If $|s| > CV_{max}$,
If $s > 0$ and $d.class \neq d'.class$, then $CV_{max} := |s|$ and $nConst := nViol := 1$.
Else if $s < 0$ and $d.class = d'.class$, then $CV_{max} := |s|$ and $nConst = nViol := 1$.
Else if $|s| := CV_{max}$,
Increment $nConst$ by 1.
If $s > 0$ and $d.class \neq d'.class$, then increment $nViol$ by 1.
Else if $s < 0$ and $d.class = d'.class$, then increment $nViol$ by 1.
3. Return $CV_{max} * \frac{nViol}{nConst}$.

---

where CV is an estimation of the proportion of the maximum strength constraints that are violated, weighted by their strength. It ranges from 0 (no constraints violated) to 1 (all constraints with strength 1.0 violated). The penalty for constraint violation is non-linear. Since the goal is to minimize the objective function, partitions that violate constraints of greater strength will appear much "worse" to the algorithm than those that violate weaker constraints.

More precisely, the CONSTVIOL function examines each constraint that applies to data point d. CV_{max} is the maximum (absolute) strength of d's violated constraints. $nConst$ counts the number of constraints at strength CV_{max}, and $nViol$ tracks how many of those are violated. The function then returns CV_{max} times the fraction of maximum strength constraints that are violated. Constraints at a lower strength may or may not also be violated, but the maximum strength violations take precedence, as they should.

This algorithm is similar in motivation to the contig-k-means algorithm (Theiler and Gisler, 1997). However, they incorporate only one kind of soft constraint into the k-means objective function (contiguity), and they calculate a linear combination of contiguity and variance:

$$f(C_1 \ldots C_k) = \lambda \cdot D + (1 - \lambda) \cdot var \tag{6.3}$$

where D refers to the discontiguity of the partition $C_1 \ldots C_k$, and λ is a user-specified weighting factor that determines the relative importance of variance versus contiguity. In contrast, SCOP-KMEANS has the ability to incorporate a variety of different soft constraints, each with their own strength. In particular, our formulation can encode information indicating stronger contiguity in certain areas than in others. Further, our algorithm can accommodate negative constraints, which the contig-k-means algorithm cannot. Their constraint model does not allow negative constraints.

6.5 Generating Spectral Analysis Constraints

SCOP-KMEANS gives us the ability to encode approximate or heuristic domain knowledge in a form that the algorithm can use to guide its search through the space of possible partitions of the data. With respect to our exploratory analysis of HST observations of Mars, soft constraints are essential. Although some knowledge is encoded in our choice of features (and derived features), other information cannot be represented in that form. In contrast to previous approaches to clustering spectral data, we can use constraints to make such information available to the clustering algorithm.

There are two major kinds of domain knowledge that we selected for testing in conjunction with constrained clustering. The first, *spatial contiguity*, is commonly useful whenever processing images. The second, *slope relationships*, is specific to our current knowledge of the planet Mars. Both kinds of information cannot be effectively encoded either as simple features or as hard constraints. Although we will only report on two sources of domain knowledge for this problem, we do not claim that they are a comprehensive representation of everything currently known about Mars. A significant advantage of our clustering architecture is that new hypotheses about data relationships can be easily encoded and tested. We plan to investigate several additional sources of domain knowledge about this problem in a similar way in the future.

Spatial contiguity. The assumption that regions in an image should be spatially contiguous is a common one. When performing image segmentation to identify objects in a scene, contiguity is required.

For remote sensing, contiguity is important, but there is an important tradeoff between contiguity and the level of precise distinctions we wish to make. For example, we could classify the entire state of Utah as "desert," or we could identify multiple regions ("canyons," "desert," "forest," "water"). However, the "water" cluster will appear in several different isolated regions of the state, which greatly reduces the contiguity of this cluster (and the entire partition). Creating a separate cluster for each body of water ("Great Salt Lake," "Utah Lake," "Colorado River," "Lake Powell," etc.) would improve the contiguity of the partition, but this is only useful if the water in each of these bodies is different enough (in the end user's opinion) to merit being placed in different clusters.

Therefore, we express spatial contiguity as a soft constraint. It is a *global constraint*, but we can easily encode it as a set of instance-level soft constraints. More precisely, for each pair of pixels d, d' that are spatial neighbors in the image, we generate a constraint $\langle d, d', s_{contig} \rangle$. Formally,

$$\underset{d,d'}{\forall}\ d.neighbor(d') \Rightarrow \langle d, d', s_{contig} \rangle \in Pref.$$

We define the spatial neighbors of d as its eight directly adjacent pixels (border pixels will have fewer than eight neighbors). This formulation allows us to easily experiment with different values for s_{contig}. We will show the results obtained with a range of s_{contig} values in Section 6.6.

Slope relationships. The SL derived feature, which indicates the slope of a given spectrum between 800 and 1000 nm, is known to be very important for distinguishing Mars spectra (Bell et al., 2000). A negative slope can be indicative of a Fe^{2+} absorption band, hinting at the presence of olivine or pyroxene. However, a positive slope rules out the possibility of such a band. Therefore, because we are seeking mineralogically significant separations, we decided to test applying a negative soft constraint to pairs of spectra that have slopes of different signs. In this case, we are encoding a *feature-level constraint* as a set of instance-level constraints. More precisely,

$$\underset{d,d'}{\forall}\ d.SL > 0 \; and \; d'.SL < -0.03 \Rightarrow \langle d, d', s_{slope} \rangle \in Pref.$$

The range of SL on this data set is from -0.05 to 0.01. We restrict this constraint to items d' with a relatively strong negative value ($d'.SL < -0.03$). We will show the results of experimenting with various fixed values of s_{slope}. Another possible approach would be to let s_{slope} vary with the magnitude of the difference between $d.SL$ and $d'.SL$. Again, this level of fine control over soft constraints has not previously been possible.

6.6 Experimental Results

Our experiments with the STIS data and soft constraints have yielded a variety of interesting and exciting results. We have organized the presentation of these results into three

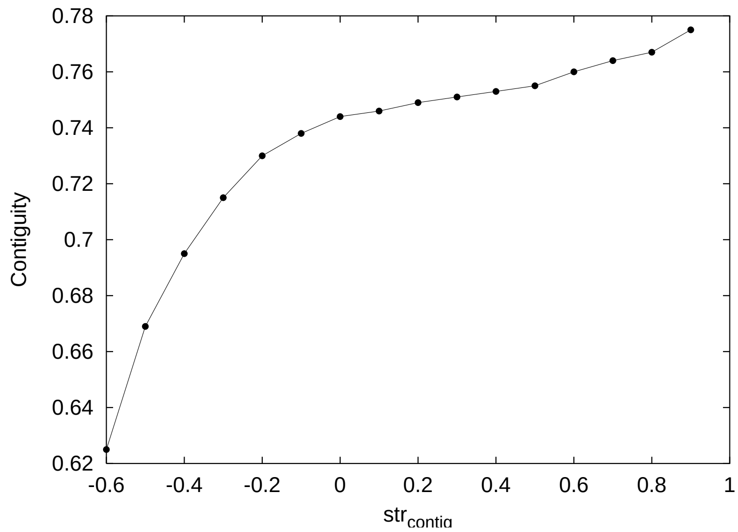


Figure 6.4: Contiguity obtained with k=4 for different s_{contig} values

categories. First, we will show experiments that confirm that our encoding and use of soft constraints is in fact an effective way to incorporate domain knowledge in clustering. Next, we will compare our analysis to results reported by other researchers about Mars. Finally, we will present a summary of several novel results that our experiments have produced. The results of these experiments, taken as a whole, firmly establish constrained clustering as a useful spectral analysis tool.

6.6.1 Effective Use of Soft Constraints

The spatial contiguity constraint incorporates the spatial relationships in an image into the clustering process. We can test whether this information has been effectively encoded and applied by the algorithm by looking at the contiguity of the resulting partition with and without constraints. This kind of constraint therefore provides an easy way to confirm the effectiveness of our soft constraint formulation and algorithm.

With respect to the STIS data, we created a set of contiguity constraints as described in Section 6.5 and experimented with different values for s_{contig}, the constraint strength parameter. In this case, we do not take a transitive closure over these constraints. This heuristic encodes a local notion of contiguity, which indicates that a pixel prefers to be classified similarly to its neighbors. A transitive closure of these constraints would require that *every* pixel be placed into the same class (with strength s_{contig}), which is not the preference we want to encode.

Figure 6.4 plots the per-pixel contiguity of the output partition for sixteen different runs of SCOP-KMEANS on the STIS data with $k = 4$. Each run used the same random initial cluster centers but had access to a different set of soft constraints; each constraint set used a different value for s_{contig}. Contiguity on the y-axis is specified as the average per-pixel contiguity. It is calculated in the following way:

$$Contig(P) = \frac{1}{n} \sum_{(d,d') \in N} \delta(d.class = d'.class). \tag{6.4}$$

The partition P contains all of the items $d \in D$, with each item annotated with its assigned cluster (class). N is a binary relation containing neighbor pairs (d, d'). δ is a function that returns 1 if its argument is true and 0 otherwise. A per-pixel contiguity of 0.75, for example, means that on average, 3/4 of a pixel's neighbors have the same class that the pixel does.

We experimented with s_{contig} ranging from -0.9 to 0.9 (we will explain what a negative value for this constraint means shortly). There are no solutions for $s_{contig} = 1.0$. The only way to find a valid solution for $s_{contig} = 1.0$ is to put all of the pixels into the same class, which is impossible for any $k \neq 1$. Also, the results obtained with $s_{contig} = [-0.7, -0.9]$ are degenerate; clusters are composed of uniformly scattered points in the image.

Figure 6.4 shows that as s_{contig} increases, so does the contiguity of the output partition, which demonstrates that the SCOP-KMEANS algorithm can accept these soft constraints and apply them successfully. Normally, we would use a positive value for s_{contig}, to indicate a preference towards spatial contiguity. However, we also tested what would happen when specifying a *negative* value for s_{contig}, indicating a preference against contiguity. SCOP-KMEANS was also successful at incorporating this kind of constraint: contiguity steadily decreases as s_{contig} becomes more negative.

The output at $s_{contig} = 0.0$ is exactly what is obtained when running without any constraints, as we would expect. This data set exhibits a fairly high amount of spatial contiguity even when clustering without constraints (0.744, with $k = 4$). We observe what seems to be an inflection point in the graph around $s_{contig} = 0.0$, due to our non-linear constraint violation penalty. If we instead used a linear penalty, we would expect this graph to exhibit linear behavior. Instead, the non-linear penalty allows us to interpret a $|s_{contig}|$ of 1.0 as a completely hard constraint that cannot be violated.

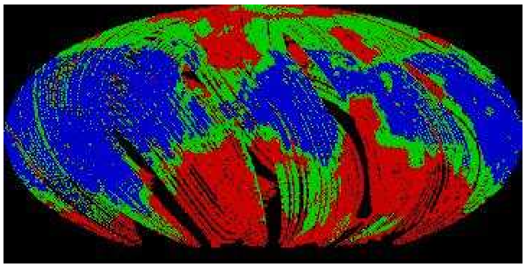
6.6.2 Comparison to Other Research on Mars

We have shown that SCOP-KMEANS can effectively make use of soft constraints by discussing our contiguity results. In addition, our analysis has also produced science results that largely agree with other investigations of Mars observations. Bell et al. (1997) collected HST observations of Mars in 1994 and 1995 at nine wavelengths (225–1042 nm) and analyzed the data using ratio images and scatterplots. They report three major findings:

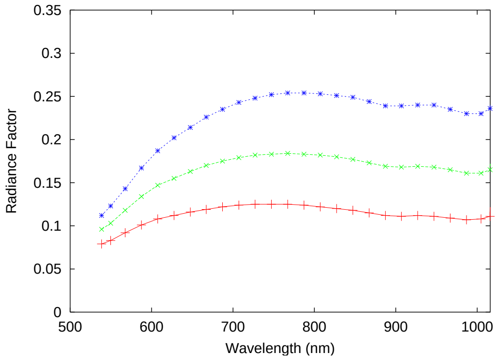
1. **B1:** There are two distinct bright regions, largely distinguished by their $\frac{R_{673}}{R_{410}}$ ratios (one is 20–25% higher than the other).

2. **B2:** Dark regions have a 3–5% deeper absorption band at 860 nm than do the bright regions.

3. **B3:** The 953-nm absorption band has a depth of 0–5% for bright regions and a depth of 7–15% for dark regions.



(a) Classification



(b) Average cluster spectra

Figure 6.5: K-means output for $k = 3$

Comparison to B1. Our observations do not include wavelengths as low as 410 nm, but we can examine the average values for FL for each cluster (this is the $\frac{R_{528}}{R_{762}}$ ratio). Running with $k = 3$ (and no constraints) produces clusters that correspond to one cluster covering the "dark" areas and two clusters collectively covering the "bright" areas (see Figure 6.5a). In the absence of constraints, we find that the two bright clusters have FL values of 0.523 and 0.446, which indicates that one has a ratio that is 17% higher than the other. The average cluster spectra are shown in Figure 6.5b. This agrees well with **B1**. It is important to note that these three clusters were identified based on their homogeneity; we did not identify "bright" and "dark" regions explicitly for the algorithm to analyze.

When performing the same analysis but incorporating a spatial contiguity constraint with strength 0.5, this result is unchanged; we still find two bright clusters with the same relationship. However, the clusters are slightly more contiguous (per-pixel contiguity rises from 0.802 to 0.814).

Comparison to B2. We observe less of a distinction between the bright and dark regions in terms of BD_{860} than was reported by Bell et al. Rather than the dark areas having a 3–5% deeper absorption band, we instead see a difference of only about 2%. However, the trend is the same.

Comparison to B3. Interestingly, our results directly contradict **B3**. Recall that the deepest band depth at 950 nm in our data set is only 6%. We observe a very minimal average 950-nm absorption band for bright regions and an absorption of 1–2% for dark regions.

There are two possible reasons for the discrepancy between our results and **B3**. First, we are using a very coarse division of the planet into two or three regions and labeling those as "bright" and "dark." Although the averages show very small absorption features, the constituent pixels in each cluster possess a variety of band depths. Pixels in the bright cluster have a BD_{950} that ranges from 4% to -7% (a negative band depth indicates a local reflectivity *maximum* rather than an absorption feature). Pixels in the dark cluster range from having a BD_{950} of 6% to -6%.

Second, we are probably not calculating the continuum ("expected") radiance value at 950 nm in the same way that Bell et al. did. This is important, because it directly affects the band depth calculation. This may explain why the maximum band depth we observed differs so greatly from their observations.

Regardless, we are encouraged by the degree of agreement we find between our independent analyses of the same planet. We find agreement even though the observations are separated by five years and some of the derived features have not been calculated in exactly the same way. In terms of observing conditions, Bell et al. noted that at the time they were collecting their data, Mars was experiencing very cloudy weather. In contrast, our data set was created during extraordinarily clear Martian conditions. This allows us to see more of the planetary mineralogy, due to the reduction in obscuring ice and clouds. As we will see in the next section, our automated analysis was also able to detect and isolate areas where ice and clouds are present.

6.6.3 Novel Scientific Findings

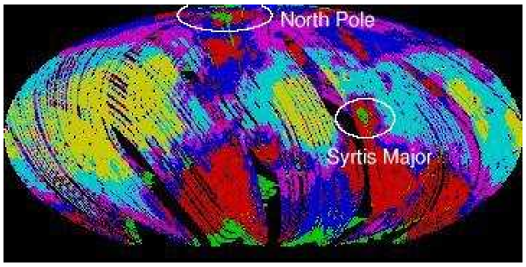
In addition to results that confirm or suggest alternatives to existing knowledge about Mars, our analysis also suggest several other interesting features of our particular data set. These observations add to the sum of our knowledge about the planet and are therefore valuable from a scientific viewpoint. In this section, we detail five interesting observations that came out of our automated analysis of the data set.

950-nm band depth. We found that this absorption band was a very distinct feature for regions on Mars. When clustering with $k = 2$, we find that the bright cluster has an average band depth of 0.3%. This is minimal, but it clearly contrasts with the average result for the dark clusters. Rather than finding an absorption feature at 950 nm, we find that the average value for BD_{950} is -2% (a local reflectivity maximum). This result suggests that the bright regions may be more likely to have a Fe^{3+} absorption band at this location.

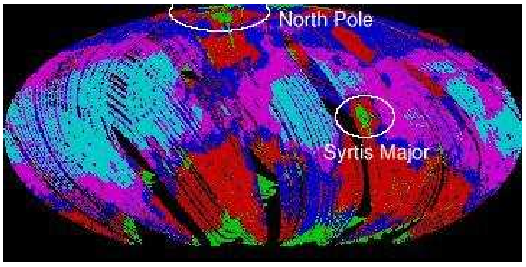
800 to 1000 nm slope. This feature also turns out to be important for distinguishing regions on Mars, as we hypothesized. The overall average slope for this data set is negative from 800 to 1000 nm. Our results indicate that bright regions, on average, have a slope that is twice as strongly negative as the dark regions.

Unusually dark low-red regions. There are two regions on Mars that show up as having abnormally *dark and flat* spectra, in comparison with the rest of the planet. Most of Mars is covered with spectra that have low FL values; the average for our data set is 0.530. However, these two regions have an unusually high average FL value of 0.820 (at $k = 6$). In fact, every pixel in the cluster has an above-average FL value (from 0.642 to 1.018). These spectra, because they are so relatively flat, can also be considered "low-red" regions; they will appear much less red than most of the planet does.

Besides being flat, these spectra are also very dark, with very low overall radiance values. The spatial location of these pixels is relevant; they occur in two places, Syrtis Major and



(a) With contiguity constraints, $k = 6$, $s_{contig} = 0.5$



(b) Without constraints, $k = 5$

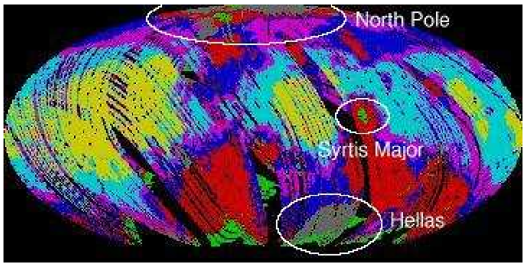
Figure 6.6: Dark low-red regions on Mars

two patches just southwest of the north polar cap. These are two places previously known to be very dark (McCord et al., 1971). Figure 6.6a shows the output of SCOP-KMEANS when using contiguity constraints with $k = 6$. The two regions of interest are labeled. (There are other pixels also assigned to this cluster, near the south edge of the planet. However, they are mostly artifacts of the data gathering process; they were observed under very poor conditions due to the viewing geometry between the Earth and Mars.)

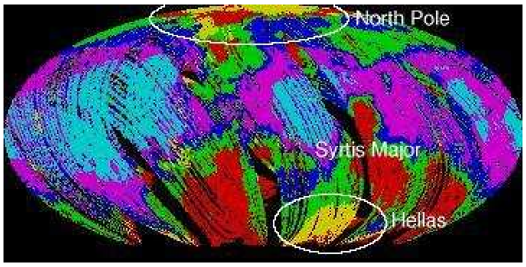
These results were obtained when clustering with contiguity constraints. If we do not use it, a similar cluster appears at $k = 5$, but it is much less distinguished; its average FL value is only 0.799. This, too, is above average, but it is less spatially (and spectrally) coherent. The variance for the cluster obtained when using the regular k-means algorithm is 0.068, while the variance of the corresponding cluster when using COP-KMEANSis only 0.057. Lower variance means that the average cluster spectrum is a better fit to the items in the cluster, spectrally. The average spectrum for the unconstrained cluster is shown in Figure 6.3b. It is the flattest cluster in the graph, clearly distinguished from the other clusters.

Ice/cloud regions. There are also two regions on Mars that are distinguished by *bright and flat* spectra. They appear in the north polar region and over the Hellas Basin. If we increase k to 7, this cluster appears with a FL of 0.745, which is higher than average yet distinguished from the dark, flat regions above. In fact, the higher value of k also permits a tighter fit to the dark flat cluster; it now has an average FL of 0.825. Because these regions are significantly brighter than dark flat cluster, and because of their spatial locations, we hypothesize that they represent ice (water ice or CO_2 ice) present at the pole and thin clouds present over the Hellas Basin. Both of these hypotheses are physically plausible and in accord with what we believe to be possible on Mars.

Figure 6.7a shows the output of SCOP-KMEANS when using contiguity constraints with $k = 7$. The north polar region is almost entirely covered by the ice/cloud cluster, as is the Hellas basin. The dark flat cluster is still visible (cf. Figure 6.6a). Once again, these are the results obtained when clustering with contiguity constraints. Without that information, the results are again less satisfactory. At $k = 6$, a cluster similar to the ice/cloud cluster appears (Figure 6.7b), but its FL value is at 0.751. We also observe that the ice/cloud



(a) With contiguity constraints, $k = 7$, $s_{contig} = 0.5$



(b) Without constraints, $k = 6$

Figure 6.7: Ice/cloud regions on Mars

cluster obtained when using contiguity constraints is a better fit to the data; that cluster's average variance is only 0.149, compared to an average variance of 0.178 obtained when clustering without constraints. In addition, without contiguity constraints the dark flat cluster disappears; some of its pixels appear to have been subsumed by the ice/cloud cluster, which explains why that cluster's FL value is higher.

Slope constraint has little effect. We experimented with different values for s_{slope} and different ways of calculating these constraints, but found that the constraints have little effect on the final results. This appears to be because the clusters found in the absence of constraints are already fairly well separated with respect to this feature; darker areas tend to have lower slope values than brighter regions, as mentioned above. It is possible that calculating this feature as a relative value rather than an absolute one might yield values that correlate less with albedo. We have not tested this hypothesis yet. Although the slope constraint did not significantly alter the results in this case, it is still a useful piece of knowledge to include. For exploratory analyses in particular, blind analysis of the data can produce misleading results. By including additional information such as the slope constraint, we can *guarantee* that the output will be reasonable in terms of this particular domain knowledge. Without such constraints, we cannot be certain that the results will be physically meaningful.

6.7 Summary

In this chapter, we have achieved two major goals. First, we have developed a formulation for soft constraints and a clustering algorithm that can make effective use of those constraints. Second, we applied this algorithm to a very large database of Mars observations and found that the results are of value scientifically both for confirming the results of independent analyses of the planet and for highlighting points of disagreement. These results indicate that constrained clustering can be a useful tool for scientific discovery. We believe that scientists from a variety of disciplines can benefit from adding constrained clustering to their set of frequently used analysis tools.

CHAPTER 7
CONCLUSIONS

This dissertation has presented four major contributions: an expressive way to encode knowledge as constraints, a general method for creating constrained clustering algorithms, three new constrained clustering algorithms, and a demonstration of the robustness of the approach by applying it to three significant real-world problems. This chapter summarizes each contribution and then discusses several extensions to this work that we envision.

7.1 Major Contributions

Expressive constraint formulation. We have developed an expressive constraint formulation for encoding domain knowledge about a problem. This encoding encompasses soft as well as hard constraints. We have discussed problem domains for which each type of constraint is most appropriate. We demonstrated that this encoding of knowledge effectively transmits information to constrained clustering algorithms and that the constraint formulation is rich enough to be of use on a diverse array of problems. Further, our encoding is novel in that it accommodates *negative* constraints as well as positive ones.

General method for creating constrained clustering algorithms. We have also developed a general method for transforming a regular partitioning clustering algorithm into a constrained clustering algorithm. The resulting algorithm can apply knowledge encoded as instance-level pairwise constraints to guide its selection of the best partition of the data. This transformation is not particular to any single algorithm, and its general applicability is one of its strengths.

Three new constrained clustering algorithms. We have applied this transformation to two common clustering algorithms, k-means and COBWEB, and developed constrained versions of each that are general enough to be applied to a variety of problems. Two of the algorithms, COP-KMEANS and COP-COBWEB, are designed to function with hard constraints, which must be satisfied by the final output partition. The third algorithm, SCOP-KMEANS, accommodates information encoded as soft constraints, which express preferences of varying strengths about the placement of items. All three algorithms represent novel contributions to the body of algorithmic tools available for data analysis.

Results on real-world problems. We have demonstrated the robustness of our approach by applying constrained clustering algorithms to three significant real-world problems and reporting on the improvements obtained when using constraints. First, we used COP-KMEANS to automatically improve road maps, refining them to the resolution of individual lanes. Second, we applied COP-COBWEB to the problem of noun phrase coreference resolution. Finally, we used SCOP-KMEANS to analyze spectral observations of the planet Mars; our results both support existing hypotheses about Mars and uncover interesting new aspects of the data set. The diversity of problems we experimented with is a testament to the general applicability of our techniques.

7.2 Extensions to Constrained Clustering

We have demonstrated the strengths of constrained clustering using several algorithms and applying our techniques to several real-world problems. There are several extensions to this work that we envision as the next logical steps. First, we plan to develop a soft constrained version of COBWEB. We expect that the method we used to develop SCOP-KMEANS can also be applied to develop a SCOP-COBWEB algorithm. We will apply this algorithm to the coreference problem, which will also allow us to encode a broader collection of information, as soft constraints. A comparison between the results of SCOP-COBWEB and the results we have already obtained with COP-COBWEB will further our understanding of the relative benefits of each type of constraint.

In addition, our work has been focused exclusively on partitioning algorithms. There are cases where hierarchical algorithms, which construct an organization of the data at several different levels of generality, are most useful. We anticipate the development of intelligent constrained hierarchical methods modeled after our constrained partitioning work, and we believe that the key lies in the use of soft constraints.

Finally, we have only scratched the surface in terms of interesting information to be discovered in the STIS Mars data set. We plan further experiments, using additional features and constraints, to assist us in further exploration of the planet. For example, we plan to experiment with the contiguity constraints that vary in strength across the planet. We will assign regions known to be fairly homogeneous a stronger contiguity constraint than "borderline" regions (such as the transition to the polar caps). In the process of these experiments, we will demonstrate the power of constrained clustering as a scientific tool. If this tool continues to be effective at confirming existing knowledge about Mars as well as discovering novel information, we hope to encourage its broader use in other scientific endeavors.

BIBLIOGRAPHY

Abu-Mostafa, Y. S. (1995). Hints. *Neural Computation*, 7:639–671.

Abu-Mostafa, Y. S. (2001). Financial model calibration using consistency hints. *IEEE Transactions on Neural Networks*, 12(4):791–808.

Adams, J. B. (1974). Visible and near-infrared diffuse reflectance spectra of pyroxenes as applied to remote sensing of solid objects in the solar system. *Journal of Geophysical Research*, 79:4829–4836.

Adams, J. B. and McCord, T. B. (1969). Mars: Interpretation of spectral reflectivity of light and dark regions. *Journal of Geophysical Research*, 74:4851–4856.

Ambroise, C., Dang, M., and Govaert, G. (1997). Clustering of spatial data by the EM algorithm. In A. Soares et al., editor, *geoENV I-Geostatistics for Environmental Applications*, pages 493–504. Kluwer Academic Publishers.

Anderberg, M. (1973). *Cluster Analysis for Applications.* Academic Press.

Aone, C. and Bennett, W. (1995). Evaluating automated and manual acquisition of anaphora resolution strategies. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 122–129. Association for Computational Linguistics.

Appelt, D., Hobbs, J., Bear, J., Israel, D., Kameyama, M., Kehler, A., Martin, D., Meyers, K., and Tyson, M. (1995). SRI International FASTUS system: MUC-6 test results and analysis. In *Proceedings of the Sixth Message Understanding Conference*, pages 237–248, San Francisco, CA. Morgan Kaufmann.

Azzam, S., Humphreys, K., and Gaizauskas, R. (1998). Evaluating a focus-based approach to anaphora resolution. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and COLING-98*, pages 74–78. Association for Computational Linguistics.

Bachar, K. and Lerman, I. (1998). Statistical conditions for a linear complexity for an algorithm of hierarchical classification under constraint of contiguity. In Rizzi, A., Vichi, M., and Bock, H., editors, *Advances in Data Science and Classification*, pages 131–136. Springer-Verlag.

Bagga, A. and Baldwin, B. (1998). Algorithms for scoring coreference chains. In *Proceedings of the LREC 1998 Workshop on Linguistic Coreference*, pages 563–566.

Basu, S., Bannerjee, A., and Mooney, R. (2002). Semi-supervised clustering by seeding. In *Proceedings of the Nineteenth International Conference on Machine Learning.*

Batagelj, V. and Ferligoj, A. (1998). Constrained clustering problems. In Rizzi, A., Vichi, M., and Bock, H., editors, *Advances in Data Science and Classification*, pages 137–144. Springer-Verlag.

Béjar, J. and Cortés, U. (1998). Experiments with domain knowledge in unsupervised learning: using and revising theories. *Computación y Sistemas*, 1(3):136–143.

Bell, J. F. (1992). Charge-coupled device imaging spectroscopy of Mars. *Icarus*, 100:575–597.

Bell, J. F. (1997). Visible and near-infrared spectroscopy. In Shirley, J. H. and Fairbridge, R. W., editors, *Encyclopedia of Planetary Sciences*, pages 911–915. Chapman and Hall.

Bell, J. F., McSween Jr., H. Y., Crisp, J. A., Morris, R. V., Murchie, S. L., Bridges, N. T., Johnson, J. R., Britt, D. T., Golombek, M. P., Moore, H. J., Ghosh, A., Bishop, J. L., Anderson, R. C., Bruckner, J., Economou, T., Greenwood, J. P., Gunnlaugsson, H. P., Hargraves, R. M., Hviid, S., Knudsen, J. M., Madsen, M. B., Reid, R., Rieder, R., and Soderblom, L. (2000). Mineralogic and compositional properties of Martian soil and dust: Results from Mars Pathfinder. *Journal of Geophysical Research*, 105(E1):1721–1755.

Bell, J. F., Wolff, M. J., James, P. B., Clancy, R. T., Lee, S. W., and Martin, L. J. (1997). Mars surface mineralogy from Hubble Space Telescope imaging during 1994-1995: Observations, calibration, and initial results. *Journal of Geophysical Research*, 102(E4):9109–9123.

Bertin, E. (2000). Mining pixels: The extraction and classification of astronomical sources. In Banday, A. J., Zaroubi, S., and Bartelmann, M., editors, *Mining the Sky: Proceedings of the MPA/ESO/MPE Workshop*, pages 353–371. European Southern Observatory, Springer.

Blake, C. L. and Merz, C. J. (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html.

Blum, A. and Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 92–100.

Bottou, L. and Bengio, Y. (1995). Convergence properties of the k-means algorithm. *Advances in Neural Information Processing Systems*, 7:585–592.

Bradley, P. S., Bennett, K. P., and Demiriz, A. (2000). Constrained k-means clustering. Technical Report MSR-TR-2000-65, Microsoft Research, Redmond, WA.

Bradley, P. S. and Fayyad, U. M. (1998). Refining initial points for k-means clustering. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 91–99, San Francisco, CA. Morgan Kaufmann.

Brown, D. E. and Gunderson, L. F. (2001). Using clustering to discover the preferences of computer criminals. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 31(4):311–318.

Cardie, C. (1993). A case-based approach to knowledge acquisition for domain-specific sentence analysis. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 798–803, Washington, DC. AAAI Press / MIT Press.

Cardie, C. and Howe, N. (1997). Improving minority class prediction using case-specific feature weights. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 57–65, Vanderbilt University, Memphis, TN. Morgan Kaufmann.

Cardie, C. and Pierce, D. (1998). Error-driven pruning of treebank grammars for base noun phrase identification. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and COLING-98*, pages 218–224, University of Montreal, Montreal, Canada. Association for Computational Linguistics.

Cardie, C. and Wagstaff, K. (1999). Noun phrase coreference as clustering. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 82–89, University of Maryland, MD. Association for Computational Linguistics.

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., and Freeman, D. (1988). Autoclass: A Bayesian classification system. In *Proceedings of the Fifth International Workshop on Machine Learning*, pages 54–64, Ann Arbor, MI. Morgan Kaufmann.

Cohn, D., Atlas, L., and Ladner, R. (1994). Improving generalization with active learning. *Machine Learning*, 15(2):201–221.

Cohn, D., Caruana, R., and McCallum, A. (2000). Semi-supervised clustering with user feedback. Unpublished manuscript.

Dagan, I. and Itai, A. (1991). A statistical filter for resolving pronoun references. In Feldman, Y. A. and Bruckstein, A., editors, *Artificial Intelligence and Computer Vision*, pages 125–135. Elsevier Science Publishers, North Holland.

de Carvalho, R. R., Djorgovski, S. G., Weir, N., Fayyad, U., Cherkauer, K., Roden, J., and Gray, A. (1995). Clustering analysis algorithms and their applications to digital POSS-II catalogs. In Shaw, R. A., Payne, H. E., and Hayes, J. J. E., editors, *Astronomical Data Analysis Software and Systems IV*, volume 77 of *ASP Conference Series*, pages 272–275.

Demiriz, A., Bennett, K. P., and Embrechts, M. J. (1999). Semi-supervised clustering using genetic algorithms. In *Proceedings of Artificial Neural Networks in Engineering '99*.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38.

Djorgovski, S. G., Brunner, R. J., Mahabal, A. A., Odewahn, S. C., de Carvalho, R. R., Gal, R. R., Stolorz, P., Granat, R., Curkendall, D., Jacaob, J., and Castro, S. (2000). Exploration of large digital sky surveys. In Banday, A. J., Zaroubi, S., and Bartelmann, M., editors, *Mining the Sky: Proceedings of the MPA/ESO/MPE Workshop*, pages 305–322. European Southern Observatory, Springer.

Fellbaum, C. (1998). *WordNet: An Electronical Lexical Database.* The MIT Press, Cambridge, MA.

Ferligoj, A. and Batagelj, V. (1982). Clustering with relational constraint. *Psychometrika*, 47(4):413–426.

Ferligoj, A. and Batagelj, V. (1983). Some types of clustering with relational constraints. *Psychometrika*, 48(4):541–552.

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

Fontaine, V., Leich, H., and Hennebert, J. (1994). Influence of vector quantization on isolated word recognition. In Holt, M. J. J., Cowan, C. F. N., Grant, P. M., and Sandham, W. A., editors, *Signal Processing VII, Theories and Applications. Proceedings of EUSIPCO-94. Seventh European Signal Processing Conference*, volume 1, pages 115–118, Lausanne, Switzerland. Eur. Assoc. Signal Process.

Forgy, E. W. (1965). Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768–769.

Fowlkes, E. B. and Mallows, C. L. (1983). A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383):553–569.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.

Gluck, M. A. and Corter, J. E. (1985). Information, uncertainty, and the utility of categories. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pages 283–287, Hillsdale, NJ. Lawrence Erlbaum.

Goebel, J., Volk, K., Walker, H., Gerbault, F., Cheeseman, P., Self, M., Stutz, J., and Taylor, W. (1989). A Bayesian classification of the IRAS LRS Atlas. *Astronomy and Astrophysics*, 222:L5–L8.

Gordon, A. D. (1973). Classification in the presence of constraints. *Biometrics*, 29:821–827.

Gordon, A. D. (1981). *Classification*. Chapman and Hall, New York, NY.

Gordon, A. D. (1996). A survey of constrained classification. *Computational Statistics & Data Analysis*, 21:17–29.

Guha, S., Rastogi, R., and Shim, K. (1998). CURE: An efficient algorithm for clustering large databases. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pages 73–84, Seattle, WA.

Gulati, R. K., Gupta, R., Gothoskar, P., and Khobragade, S. (1995). Automated classification of a large database of stellar spectra. In Shaw, R. A., Payne, H. E., and Hayes, J. J. E., editors, *Astronomical Data Analysis Software and Systems IV*, volume 77 of *ASP Conference Series*, pages 253–256.

Hartigan, J. A. (1975). *Clustering Algorithms.* Wiley, New York, NY.

Hunt, G. R. (1980). Electromagnetic radiation: The communication link in remote sensing. In Siegel, B. S. and Gillespie, A. R., editors, *Remote Sensing in Geology*, chapter 2, pages 5–45. Wiley, New York, NY.

Jain, A. and Farrokhnia, F. (1991). Unsupevised texture segmentation using Gabor filters. *Pattern Recognition*, 24(12):1167–1186.

Jain, A. K. and Dubes, R. C. (1988). *Algorithms for Clustering Data.* Prentice Hall.

Jancey, R. C. (1966). Multidimensional group analysis. *Australian Journal of Botany*, 14(1):127–130.

Kinderman, R. and Snell, J. L. (1980). *Markov Random Fields and Their Applications.* American Math Society, Providence, RI.

Klein, D., Kamvar, S. D., and Manning, C. D. (2002). From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In *Proceedings of the Nineteenth International Conference on Machine Learning.*

Kleinberg, J. and Tardos, E. (1999). Approximation algorithms for classification problems with pairwise relationships: Metric labeling and Markov random fields. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 14–23.

Krishna, K. and Murty, M. N. (1999). Genetic k-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 29(3):433–439.

Křivánek, M. (1991). *Algorithmic and Geometric Aspects of Cluster Analysis.* Academia Nakladatelství Československé, Praha.

Lefkovitch, L. P. (1980). Conditional clustering. *Biometrics*, 36:43–58.

Lillesand, T. M. and Kiefer, R. W. (1987). *Remote Sensing and Image Interpretation*, chapter 10 (Digital Image Processing). Wiley, 2nd edition.

Lyman, P. and Varian, H. R. (2001). How much information. Retrieved from http://www.sims.berkeley.edu/how-much-info on November 29.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Symposium on Math, Statistics, and Probability*, volume 1, pages 281–297, Berkeley, CA. University of California Press.

Marroquin, J. and Girosi, F. (1993). Some extensions of the k-means algorithm for image segmentation and pattern recognition. AI Memo 1390, Massachusetts Institute of Technology, Cambridge, MA.

Masson, P. and Pieczynski, W. (1993). SEM algorithm and unsupervised statistical segmentation of satellite images. *IEEE Transactions on Geoscience and Remote Sensing*, 31(3):618–633.

McCarthy, J. and Lehnert, W. (1995). Using decision trees for coreference resolution. In Mellish, C., editor, *Proceedings of the Fourteenth International Conference on Artificial Intelligence*, pages 1050–1055.

McCord, T. B., Elias, J. H., and Westphal, J. A. (1971). Mars: The spectral albedo (0.3-2.5μ) of small bright and dark regions. *Icarus*, 14:245–251.

McSween Jr., H. Y., Murchie, S. L., Crisp, J. A., Bridges, N. T., Anderson, R. C., Bell, J. F., Britt, D. T., Bruckner, J., Dreibus, G., Economou, T., Ghosh, A., Golombek, M. P., Greenwood, J. P., Johnson, J. R., Moore, H. J., Morris, R. V., Parker, T. J., , Rieder, R., Singer, R., and Wanke, H. (1999). Chemical, multispectral, and textural constraints on the composition and origin of rocks at the Mars Pathfinder landing site. *Journal of Geophysical Research*, 104(E4):8679–8715.

Merényi, E., Singer, R. B., and Miller, J. S. (1996). Mapping of spectral variations on the surface of Mars from high spectral resolution telescopic images. *Icarus*, 124:280–295.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill.

Mitkov, R. (1999). Anaphora resolution: the state of the art. Working paper (Based on the COLING'98/ACL'98 tutorial on anaphora resolution).

Mjolsness, E. and DeCoste, D. (2001). Machine learning for science: State of the art and future prospects. *Science*, 293(5537):2051–2055.

Montolio, P., Gasull, A., Monte, E., Torres, L., and Marqués, F. (1992). Analysis and optimization of the k-means algorithm for remote sensing applications. In de la Blanca, N. P., Sanfeliu, A., and Vidal, E., editors, *Pattern Recognition and Image Analysis*, volume 1 of *Series in Machine Perception and Artificial Intelligence*, pages 155–170. World Scientific, Singapore.

Morris, R. V., Lauer Jr., H. V., Lawson, C. A., Gibson Jr., E. K., Nace, G. A., and Stewart, C. (1985). Spectral and other physicochemical properties of submicron powders of hematite (α-Fe_2O_3), maghemite (γ-Fe_2O_3), magnetite (Fe_3O_4), goethite (α-FeOOH), and lepidocrocite (γ-FeOOH). *Journal of Geophysical Research*, 90:3126–3144.

MUC-6 (1995). *Proceedings of the Sixth Message Understanding Conference*. Morgan Kaufmann, San Francisco, CA.

MUC-7 (1998). *Proceedings of the Seventh Message Understanding Conference*. Morgan Kaufmann, San Francisco, CA.

Murtagh, F. (1985). A survey of algorithms for contiguity-constrained clustering and related problems. *The Computer Journal*, 28(1):82–88.

Narendra, P. M. and Goldberg, M. (1977). A non-parametric clustering cheme for Landsat. *Pattern Recognition*, 9:207–215.

Navigation Technologies (1996). *Software Developer's Toolkit.* Navigation Technologies, Inc., Sunnyvale, CA, 5.7.4 Solaris edition.

Ng, V. and Cardie, C. (2002). Combining sample selection and error-driven pruning for machine learning of coreference rules. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing.*

Nigam, K., McCallum, A., Thrun, S., and Mitchell, T. (1998). Learning to classify from labeled and unlabeled documents. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 792–799, Madison, WI. The AAAI Press.

Oliver, M. and Webster, R. (1989). A geostatistical basis for spatial weighting in multivariate classification. *Mathematical Geology*, 21:15–35.

Pelleg, D. and Moore, A. (1999). Accelerating exact k-means algorithms with geometric reasoning. In *Knowledge Discovery and Data Mining*, pages 277–281.

Perruchet, C. (1983). Constrained agglomerative hierarchical classification. *Pattern Recognition*, 16(2):213–217.

Pieters, C. M., Mustard, J. F., and Sunshine, J. M. (1996). Quantitative mineral analyses of planetary surfaces using reflectance spectroscopy. In Dyar, M. D., McCammon, C., and Schaefer, M. W., editors, *Mineral Spectroscopy: A Tribute to Roger G. Burns*, pages 307–325. The Geochemical Society.

Popescu-Belis, A. and Robba, I. (1998). Three new methods for evaluating reference resolution. In *Proceedings of the LREC 1998 Workshop on Linguistic Coreference.*

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA.

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(366):846–850.

Roberts, S., Gisler, G., and Theiler, J. (1996). Spatio-spectral image analysis using classical and neural algorithms. In Dagli, C. H., Akay, M., Chen, C. L. P., Fernández, B. R., and Ghosh, J., editors, *Smart Engineering Systems: Neural Networks, Fuzzy Logic, and Evolutionary Programming*, volume 6 of *Intelligent Engineering Systems Through Artificial Neural Networks*, pages 425–430. ASME Press, New York, NY.

Rogers, S., Langley, P., and Wilson, C. (1999). Mining GPS data to augment road models. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 104–113, San Diego, CA. ACM Press.

Schowengerdt, R. A. (1983). *Techniques for Image Processing and Classification in Remote Sensing*. Academic Press.

Schroedl, S., Rogers, S., , and Wilson, C. (2000). Map refinement from GPS traces. Technical Report RTC Report No. 6, DaimlerChrysler Research and Technology North America.

Selim, S. Z. and Ismail, M. A. (1984). K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(1):81–87.

Selman, B., Mitchell, D. G., and Levesque, H. J. (1996). Generating hard satisfiability problems. *Artificial Intelligence*, 81:17–29.

Shmoys, D. B., Tardos, E., and Aardal, K. (1997). Approximation algorithms for facility location. In *Proceedings of the Twenty-ninth Annual ACM Symposium on the Theory of Computing*, pages 265–274.

Sibson, R. (1973). SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34.

Soh, L. and Tsatsoulis, C. (1999). Segmentation of satellite imagery of natural scenes using data mining. *IEEE Transactions on Geoscience and Remote Sensing*, 37(2):1086–1099.

Soon, W. M., Ng, H. T., and Lim, D. C. Y. (2001). A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27(4):521–544.

Talavera, L. and Béjar, J. (1999). Integrating declarative knowledge in hierarchical clustering tasks. In *Proceedings of the International Symposium on Intelligent Data Analysis*, pages 211–222, Amsterdam, The Netherlands. Springer-Verlag.

Theiler, J. and Gisler, G. (1997). A contiguity-enhanced k-means clustering algorithm for unsupervised multispectral image segmentation. In *Proceedings of SPIE (International Society for Optical Engineering)*, volume 3159, pages 108–118.

Trouilleux, F., Gaussier, E., Bes, G. G., and Zaenen, A. (2000). Coreference resolution evaluation based on descriptive specificity. In *Proceedings of the LREC 2000 Workshop on Linguistic Coreference*.

Tung, A. K. H., Ng, R. T., Lakshmanan, L. V. S., and Han, J. (2001). Constraint-based clustering in large databases. In *Proceedings of the 2001 International Conference on Database Theory*, pages 405–419.

Urquhart, R. (1982). Graph theoretical clustering based on limited neighbourhood sets. *Pattern Recognition*, 15(3):173–187.

Vilain, M., Burger, J., Aberdeen, J., Connolly, D., and Hirschman, L. (1995). A model-theoretic coreference scoring scheme. In *Proceedings of the Sixth Message Understanding Conference*, pages 45–52, San Francisco, CA. Morgan Kaufmann.

Wagstaff, K. and Cardie, C. (2000). Clustering with instance-level constraints. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 1103–1110, Palo Alto, CA. Morgan Kaufmann.

Wagstaff, K., Cardie, C., Rogers, S., and Schroedl, S. (2001). Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 577–584, Williamstown, MA. Morgan Kaufmann.

Wharton, S. W. (1983). A generalized histogram clustering scheme for multidimensional image data. *Pattern Recognition*, 16(2):193–199.

Yoo, J., Gray, A., Roden, J., Fayyad, U. M., de Carvalho, R. R., and Djorgovski, S. G. (1996). Analysis of digital POSS-II catalogs using hierarchical unsupervised learning algorithms. In Jacoby, G. H. and Barnes, J., editors, *Astronomical Data Analysis Software and Systems V*, volume 101 of *ASP Conference Series*, pages 41–44.

Zahn, C. (1971). Graph-theoretical methods for detecting and describing Gestalt clusters. *IEEE Transactions on Computing*, C-20:68–86.

Zhang, T., Ramakrishnan, R., and Livny, M. (1996). BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*.

Zhu, X., Chu, T., Zhu, J., and Caruana, R. (2001). Heuristically inducing a distance metric from user preferences for clustering. Unpublished manuscript.