Appears in *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB-2000).*

A Probabilistic Learning Approach to Whole-Genome Operon Prediction

Mark Craven[†‡]
craven@biostat.wisc.edu

David Page[†‡]
page@biostat.wisc.edu

Jude Shavlik[‡†]
shavlik@cs.wisc.edu

Joseph Bockhorst[‡†]
joebock@cs.wisc.edu

Jeremy Glasner[*]
jeremy@genome.wisc.edu

[†]Dept. of Biostatistics & Medical Informatics
University of Wisconsin
5730 Medical Sciences Center
1300 University Avenue
Madison, Wisconsin 53706

[‡]Dept. of Computer Sciences
University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

[*]Dept. of Genetics
445 Henry Mall
University of Wisconsin
Madison, Wisconsin 53706

Abstract

We present a computational approach to predicting operons in the genomes of prokaryotic organisms. Our approach uses machine learning methods to induce predictive models for this task from a rich variety of data types including sequence data, gene expression data, and functional annotations associated with genes. We use multiple learned models that individually predict promoters, terminators and operons themselves. A key part of our approach is a dynamic programming method that uses our predictions to map every known and putative gene in a given genome into its most probable operon. We evaluate our approach using data from the *E. coli* K-12 genome.

Introduction

The availability of complete genomic sequences and microarray expression data calls for new computational methods for uncovering the regulatory apparatus of cells. We have begun a research project at the University of Wisconsin that is developing machine learning approaches for predicting new regulatory elements, such as transcription-control signals and operons, and regulatory relationships among genes using a rich variety of data sources, including genomic sequence data and expression data. In this paper we present an approach to predicting previously unknown operons in prokaryotic organisms. Our approach first involves learning a model that is able to estimate the probability that an arbitrary sequence of genes constitutes an operon. Given such a learned model, the second component of our approach is a dynamic programming method that is able to assign every gene in the given genome to its most probable operon. With these two components together we are able to predict an *operon map* for an entire genome. We evaluate our approach using data from the *E. coli* K-12 genome.

Several other research groups (Overbeek *et al.* 1999; Tamames *et al.* 1997) have recently addressed the task of predicting *functionally coupled* genes by identifying clusters of genes that are conserved across different genomes. Although the task we consider is somewhat different (we are interested in sequences of genes that are transcribed together, not just functionally related), we consider these methods to be complementary to ours in that they are based on cross-genome information, whereas our approach is based on information present within a single genome. Functional annotation associated with genes and inter-gene spacing have also been used to predict operons (Blattner *et al.* 1997). Our approach differs from this work in that it uses (i) other features in addition to these two, (ii) a machine learning method to determine how much to weight each feature in its predictions, and (iii) a dynamic programming algorithm to produce a consistent operon map.

Other research groups have also used dynamic programming to find a consistent interpretation of the predictions made by learned models (Snyder & Stormo 1993; Xu, Mural, & Uberbacher 1994). These previous efforts have addressed the task of assembling predicted exons into gene models. In contrast, our work formulates a dynamic programming approach to assigning genes to their most likely operons.

One notable aspect of our approach is that it exploits a rich variety of data types that provide useful evidence for the task of predicting operons. The data types that our learned models consider include microarray expression data, DNA sequence features, spatial

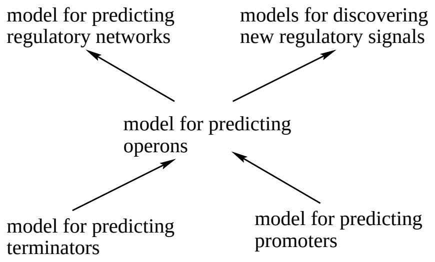

Figure 1: Our multi-level learning approach to discovering gene regulatory mechanisms. The tasks at the highest level represent those that are motivating our research. The task at the middle level represents the main focus of this paper. Those at the lowest level represent learning tasks that we address in order to make better predictions for the middle level task. The arrows represent the flow of predictions.

features (e.g., the relative positions of genes), and functional annotations for genes. As our experiments indicate, all of these data types provide some predictive value for our task. There has been much recent interest in uncovering the regulatory interactions among genes using microarray expression data (Eisen *et al.* 1998; Brown *et al.* 1999; Friedman *et al.* 2000). These approaches, unlike our method, have not incorporated other types of data as evidence.

Another compelling aspect of our approach is that it involves a multiple machine learning tasks operating at different levels of detail. As stated previously, the main task that we consider here is predicting which sequences of genes constitute operons. This task does not represent the overall goal of our work, however, but only an intermediate step in being able to attack such problems as inferring networks of regulatory interactions, and discovering new subclasses of sequences involved in controlling gene transcription. Moreover, we hypothesize that the operon prediction task can be improved by first identifying certain control sequences, such as promoters and terminators, in the genome. The recognition of these sequences is itself not a well understood task, and thus we also address the learning subtasks of constructing predictive models of these sequences. Figure 1 illustrates the relationships among the learning tasks motivating our work, and the subtasks that we address here. The predictions made by a model learned at one level are passed to one or more learning components at the next higher level to be used as input features. This idea of decomposing a given learning task into several, simpler subtasks has been around for some time (Fu & Buchanan 1985; Shapiro 1987), and has been applied more recently in robotics and simulated robotic systems (O'Sullivan 1998; Stone 1998).

The organization of the rest of the paper is as follows. In the next section, we describe in more detail the task of recognizing operons and discuss the available data. The two subsequent sections review and elaborate on our recent work (Craven *et al.* 2000) in learning models that can be used to score candidate operons. The first of these sections describes the machine learning approach we use, and the second describes the problem representation. We then present the main contribution of this paper, which is a dynamic programming approach to making whole-genome operon predictions using our learned operon models. The penultimate section presents a detailed empirical evaluation of this approach, and the final section provides discussion and conclusions.

Problem Domain

Currently, our primary task is to predict operons in the *E. coli* genome, although the approach we are developing is applicable to other prokaryotic organisms. The genome of *E. coli*, which was sequenced at the University of Wisconsin (Blattner *et al.* 1997), consists of a single circular chromosome of double-stranded DNA. The chromosome of the particular strain of E. coli (K-12) in our data set has 4,639,221 base pairs. *E. coli* has approximately 4,400 genes, which are located on both strands.

The definition of *operon* that we use throughout the paper is a sequence of one or more genes that, under some conditions, are transcribed as a unit. There are several aspects of this definition that are important to note. First, genes that are transcribed individually are included in this definition; we refer to these special cases as *singleton* operons. Second, our definition treats as multiple operons those cases (such as *rpsU-dnaG-rpoD* in *E. coli*) in which multiple promoters and/or terminators result in different subsequences of a larger gene sequence being transcribed under different conditions. We consider each of the distinct gene sequences that can be transcribed as a unit to be an operon.

Figure 2 illustrates the concept of an operon. The transcription process is initiated when RNA polymerase binds to a *promoter* before the first gene in an operon. The RNA polymerase then moves along the DNA using it as a template to produce an RNA molecule. When the RNA polymerase gets past the last gene in the operon, it encounters a special sequence called a *terminator* that signals it to release the DNA and cease transcription.

The data that we have available for learning a model of operons, some of which come from the RegulonDB (Salgado *et al.* 2000), include the following:

- complete DNA sequence of the genome,
- beginning and ending positions of 3,033 genes and 1,372 putative genes,
- positions and sequences of 438 known promoters, and 289 known terminators (147 rho-dependent and 142 rho-independent),
- functional annotation codes characterizing 1,668 genes; these are taken from a three-level, 123-leaf hierarchy (Riley 1996),

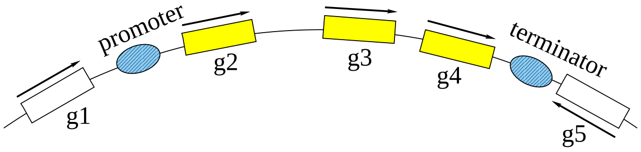

Figure 2: The concept of an operon. The curved line represents part of the *E. coli* chromosome and the rectangular boxes on it represent genes. An operon is a sequence of genes, such as $[g2, g3, g4]$ that is transcribed as a unit. Transcription is controlled via an upstream sequence, called a *promoter*, and a downstream sequence, called a *terminator*. A promoter enables the molecule performing transcription to bind to the DNA, and a terminator signals the molecule to detach from the DNA. Each gene is transcribed in a particular direction, determined by which of the two strands it is located. The arrows in the figure indicate the direction of transcription for each gene.

- gene expression data characterizing the activity levels of the 4,097 genes and putative genes across 39 experiments,
- 365 known operons.

It is estimated (F. Blattner, personal communication) that there are several hundred undiscovered operons in *E. coli*. Our immediate goal is to predict these operons using a model learned from the data described above.

An interesting aspect of our learning task is that we do not have a set of known non-operons to use as negative examples. The nature of scientific inquiry is such that several hundred operons have been identified in *E. coli*, but little attention has been focused on identifying sequences of genes that *do not* constitute operons.

We are able, however, to assemble a set of 6633 putative non-operons by exploiting the fact that operons rarely overlap with each other. Given this rule, we generate a set of negative examples by enumerating every sequence of consecutive genes, from the same strand, that overlaps, but does not coincide with a known operon. We know that some of these generated non-operons are actually true operons, because operons do overlap in some cases. However, the probability of this is small and our learning algorithms are robust in the presence of noisy data.

Machine Learning Approach

In recent work (Craven *et al.* 2000), we have applied several machine learning algorithms to the task of distinguishing operons from non-operons. In this section we review the application of one particular algorithm, naive Bayes, to this task.

Given a candidate operon – any consecutive sequence of genes on the same strand – we would like to estimate the probability that the candidate is an operon given the available data. Bayes' rule tells us we can determine this as follows:

$$\Pr(O|D) = \frac{\Pr(D|O)\Pr(O)}{\Pr(D)} \quad (1)$$

where O is a random variable indicating whether or not a candidate is an actual operon, and D represents the data we have available to make our determination.

The heart of the task is to estimate the likelihood of the data of interest given the two possible outcomes for O. Using naive Bayes, we make the assumption that our features are independent of one another given the class and therefore make the following approximation:

$$\Pr(D|O) \approx \prod_i \Pr(D_i|O) \quad (2)$$

where D_i is the ith feature.

All of our features provide numeric characterizations of candidate operons. To represent the conditional distribution of each feature given the class, we use a histogram approach. The first step of this procedure is to choose the "cutpoints" that define the bins. This procedure selects the bin boundaries such that each bin contains about 150 training examples (pooling positive and negative examples). Should some bin have no examples of one class, we assume 0.5 examples of that class fell into that bin. This smoothing method avoids zero-valued probabilities which cause Bayes' rule to produce zero as its estimated probability. We have also tried using kernel density estimates to represent our conditional distributions, but we found that the predictive accuracy of this method was no better than our histogram approach.

Problem Representation

In this section we describe the features that our learning method uses to assess the probability that a given candidate actually is an operon.

Length and Spacing Features

We use several features that relate to the size and intergenic spacing of operons and non-operons:

- Operon size: The number of genes in the candidate operon.
- Within-operon spacing: The *mean* and the *maximum* spacing (# DNA base pairs) between the genes contained in a candidate operon, e.g., the distances between *g2* and *g3* and between *g3* and *g4* in Figure 2. Since the genes in an operon are transcribed together, there might be constraints on inter-gene spacing. This feature is not defined for singleton operons (operons consisting of one gene).
- Distance to neighboring genes: The distances to the *preceding* (*g1* in Figure 2) and *following* (*g5* in Figure 2) genes. Notice that these genes are not part of the candidate operon.
- Directionality of the neighboring genes: These are two Boolean-valued features, indicating whether or not the directionality of the preceding and following genes individually match the directionality of the genes within the candidate operon. Recall that all the genes within an operon are transcribed in the same direction.

We refer to these last two features collectively as our neighboring genes features.

Functional Annotation Features

Since the genes in an operon typically act together to perform some common function, we expect the functions of the individual genes to be related to each other. The functions of many genes in *E. coli* have been described using a three-level hierarchy (Riley 1996). The levels of this hierarchy represent broad, intermediate, and detailed functions. For example one path from the root to a leaf in this hierarchy is : root→metabolism of small molecules→carbon energy metabolism→anaerobic respiration.

We measure the "functional distance" between two genes in terms of how deeply into this hierarchy they match: genes with completely identical functions have a distance of 0, genes with identical broad and intermediate functions only have a distance of 2, genes with identical broad functions only have a distance of 4, and genes without common function have a distance of 6. In cases where the function is unknown for one or both genes at a given level, we split the difference: e.g., if two genes share a broad function but the intermediate function of one is unknown, the distance is 5.

Given the distance measure between the annotations for pairs of genes, we use three features to measure the functional homogeneity of a candidate operon. Collectively, we refer to these as the functional annotation features. The first feature represents the mean pairwise functional distance for genes within the candidate operon. This feature is computed simply by computing the functional distance between each pair of genes in the candidate operon and then computing the mean of these pairwise distances. We also consider the functional distance between the gene preceding the candidate and genes within the candidate. Specifically, we compute the mean of the pairwise distances between the preceding gene and each of the genes within the operon. The third feature is analogous except that it uses the first gene *after* the candidate operon.

Transcription Signal Features

Another type of evidence associated with operons are transcription control signals, such as promoters and terminators. Thus, to decide if a given sequence of genes represents an operon or not, we would like to look upstream from the first gene in the sequence to see if we find a promoter, and to look downstream from the last gene to see if we find a terminator. The task of recognizing promoters and terminators, however, is not easily accomplished. Although there are known examples of both types of sequences, the sufficient and necessary conditions for them are not known. Thus, to use promoters and terminators as evidence for operons, we first need some method that can be used to predictively identify them.

Our approach is to use the known examples of these two types of signals to learn statistical models for predicting them. Specifically, we induce *interpolated Markov models* (IMMs) (Jelinek & Mercer 1980) that characterize the known promoters and terminators. IMMs have been used previously for modeling biological sequences (Salzberg *et al.* 1998), although the particular task here is somewhat different. An interpolated Markov model consists of a set of Markov models of different orders. An IMM makes a prediction in a given case by interpolating among the statistics represented in the models of different order.

We learn three separate IMMs, each of which is trained to recognize a particular type of signal: promoter, rho-dependent terminator, or rho-independent terminator. Our training set for promoters consists of 438 sequences, each of which is 81 bases long and contains a promoter in it. Since these promoter sequences are aligned to a common reference point (the first base transcribed), we can obtain statistics about the likelihood of a particular base at a particular position in a promoter. The terminator data set is similar, except that it consists of 289 sequences of length 58.

Our IMMs represent the probability of seeing each of the possible DNA bases at each position in the given signal. To assess the probability that a given sequence S is a promoter (or terminator), we calculate the product of the IMM's estimated probability of seeing each base in the sequence.

$$\Pr(S|\text{model}) = \prod_{i=1}^{n} \text{IMM}(S_i) \tag{3}$$

Here S_i is the base at the ith position in sequence S and n is the length of the sequence we are evaluating.

To assess the probability of seeing a given base in a particular position, our IMM interpolates between a 0th-order Markov model and a 1st-order Markov model.

$$\text{IMM}(S_i) = \frac{\lambda_{i-1,1}(S_{i-1}) \Pr_{i,1}(S_i) + \lambda_{i,0} \Pr_{i,0}(S_i)}{\lambda_{i-1,1}(S_{i-1}) + \lambda_{i,0}} \tag{4}$$

The notation $\Pr_{i,1}(S_i)$ represents the probability of seeing base S_i at the ith position under a 1st-order model, and $\Pr_{i,0}(S_i)$ represents the same under a 0th-order model. Whereas the 0th-order model simply represents the marginal probabilities of seeing each base at the given position, the 1st-order model represents the conditional probabilities of seeing each base given the previous base in the sequence S_{i-1}.

$$\Pr_{i,0}(S_i) = \Pr_i(S_i) \qquad \Pr_{i,1}(S_i) = \Pr_i(S_i|S_{i-1}) \tag{5}$$

We use a simple scheme to set the values of the λ parameters, which represent the amount of weight we give each model being interpolated. For all positions i, $\lambda_{i,0}$ is set to 1. The parameter $\lambda_{i-1,1}(S_{i-1})$ is set to 1 if the training data included at least m occurrences of the base S_{i-1} at position $i-1$, and is set to 0 otherwise. The intuition here is that we trust a 1st-order probability only if we had sufficient data from which to estimate it. In all of our experiments, we set m to 40.

Once we have induced our promoter and terminator IMMs, we can use them to look for instances of these two signals in the neighborhood of a candidate operon. We do this by "scanning" the promoter model along the 300 bases immediately preceding the first gene in a candidate operon, and similarly by scanning the terminator model along the 300 bases immediately following the last gene in the candidate. In scanning a model, we move it one base at a time. From this scanning process, we get a sequence of predicted probabilities; one for each position of the model. In the case of terminators, we actually have two separate IMMs to consider (the model for rho-dependent terminators and the model for rho-independent terminators) to get the probability for each position. We combine the estimates of these two models by assuming that the given sequence S could represent either a rho-dependent or a rho-independent terminator, and that these two possibilities are independent of one another.

In order to convert these two sequences of predicted probabilities into features that can be used to classify a candidate operon, we keep track of the greatest predicted probability for each scan. In other words, we characterize a candidate operon by two features: the promoter feature is the strongest predicted promoter we find upstream from the candidate, and the terminator feature is the strongest predicted terminator we find downstream.

Gene Expression Features

Recent *microarray* technology (*Nature Genetics* Supplement 1999) enables simultaneous measurement of the messenger RNA (mRNA) levels of thousands of genes under various experimental conditions. The Wisconsin *E. coli* Genome Project has begun generating such data, and for the work presented in this paper we use data from the first 39 experiments. Since our expression data comes from cDNA arrays, we have two measurements (fluorescence intensities) for each gene in each experiment: the relative amount of mRNA under some experimental condition versus the relative amount under some baseline condition. We refer to these two sets of measurements from a given array as the two *channels* of the array. From this data, we compute two sets of features for our learning algorithm. The first set of features is based on the ratios of the two measurements, and the second set is based on the raw measurements themselves.

For the features based on ratios, we associate with each gene g_k a vector $\vec{r}_k$ of length m, where m is the number of experiments and the elements of the vector are the measured ratios for the gene. In some cases, an element of a vector may be undefined, due to measurement problems in these experiments. Since the genes within an operon are coordinately controlled, we expect the expression vectors of two such genes to be more correlated than the expression vectors of two randomly chosen genes. Hence, the first set of features we use for classifying candidate operons are based on pairwise correlations between the expression vectors of genes of interest. We use a correlation metric similar to one that has been used previously for analyzing gene expression data (Eisen *et al.* 1998). The correlation between the vector for kth gene and the vector for the lth gene is given by:

$$\mathrm{corr}(\vec{r}_k, \vec{r}_l) = \frac{\displaystyle\sum_{\vec{r}_k(j), \vec{r}_l(j) \text{ defined}} \frac{\vec{r}_k(j)}{\Phi_{\vec{r}_k, \vec{r}_l}} \frac{\vec{r}_l(j)}{\Phi_{\vec{r}_l, \vec{r}_k}}}{|\vec{r}_k(j), \vec{r}_l(j) \text{ defined}|}. \tag{6}$$

Here $\vec{r}_k(j)$ and $\vec{r}_l(j)$ represent the jth vector values for $\vec{r}_k$ and $\vec{r}_l$ (the ratios for the jth experiment), and $\Phi_{\vec{x}, \vec{y}}$ is defined as follows:

$$\Phi_{\vec{x}, \vec{y}} = \sqrt{\frac{\displaystyle\sum_{\vec{x}(j), \vec{y}(j) \text{ defined}} \vec{x}(j)^2}{|\vec{x}(j), \vec{y}(j) \text{ defined}|}} \tag{7}$$

In computing the correlation between a pair of genes, we consider only those experiments for which both genes have defined ratios (i.e., neither vector has a missing element for the specified position).

We use three features based on these correlations to characterize a candidate operon c: (i) the mean correlation of every pair of genes in c, (ii) the correlation between the first gene in c and the previous gene in the sequence, (iii) the correlation between the last gene in c and the next gene in the sequence.

The ratio correlation measure is appropriate to identify genes that have similar expression profiles. However, we expect an even stronger association among genes that are in the same operon. Since these genes are transcribed together, the *absolute* amount of transcript produced should be similar. Consequently, we also derive features from the raw measurements (fluorescence intensities).

For these features, we treat each channel as a separate experiment, but since the conditions on the baseline channels from some experiments are the same, we merge data from these experiments after normalizing by the total signal intensity over the array.

The basic idea of the features we compute using raw intensity values is the following. Given a candidate operon and a single experiment, how likely is it that we would see the intensity values we recorded for the genes in the candidate if these genes were actually in the same operon? Our approach is based on the model that each intensity value is the result of some true underlying signal (the amount of transcript present) plus normally distributed random noise. Let $\vec{a}_{cj}$ be a vector of expression values from the jth experiment for the genes in candidate operon c. If c actually is an operon, the likelihood of the observed measurements $\vec{a}_{cj}$ is given by:

$$\Pr(\vec{a}_{cj} | O = \text{ true}) \propto \prod_i N(\vec{a}_{cj}(i), \mu_{cj}, \sigma_{cj}). \tag{8}$$

Here O is the random variable indicating whether or not the candidate c is an operon, $\vec{a}_{cj}(i)$ represents the intensity value of the ith gene in the candidate for the jth experiment, and $N(\vec{a}_{cj}(i), \mu_{cj}, \sigma_{cj})$ represents the density value of $\vec{a}_{cj}(i)$ under a normal distribution with parameters μ_{cj} and σ_{cj}. In our model, μ_{cj} represents the true signal for the operon in this experiment, and σ_{cj} represents the standard deviation of the noise for the given operon and experiment. Our approach estimates μ_{cj} by the mean of the intensity values in the jth experiment for the genes in the candidate. The parameter σ_{cj} is estimated from the training set operons. In particular, we fit a linear function in (μ_j, σ_j) space, where each data point in this space is determined from a known operon in the training set (singletons are ignored). This model accounts for two sources of noise; one that depends on signal strength (Chen, Dougherty, & Bittner 1997), and one that does not.

We also consider the marginal probability of seeing the observed intensity values:

$$\Pr(\vec{a}_{cj}) \propto \prod_i \lambda_j \exp(-\lambda_j \vec{a}_{cj}(i)). \tag{9}$$

Here we assume that the distribution of intensity values over all genes in an experiment is exponential. The parameter λ_j is determined by the maximum likelihood estimate from all of the data in an experiment.

Now to assess a candidate operon we compute the following likelihood ratio in which we consider all experiments and treat them as independent of one another:

$$L(c) = \frac{\prod_j \Pr(\vec{a}_{cj} | O = \text{ true})}{\prod_j \Pr(\vec{a}_{cj})} \tag{10}$$

Using this approach, we calculate three features for a candidate operon c: (i) the likelihood ratio $L(c)$ for all of the genes c, (ii) the likelihood ratio for the first gene in c and the previous gene in the sequence, and (iii) the likelihood ratio for the last gene in c and the next gene in the sequence.

Collectively, we refer to all six of the features described in this section (the three ratio-based features and the three absolute features) as the expression data features.

Constructing a Whole-Genome Operon Map

Up to this point, our discussion has focused on the task of classifying isolated operons. However, we are primarily interested in making operon predictions not for isolated candidates, but for entire genomes. In this section, we address the task of predicting *operon maps*. We define an operon map to be a set of predictions that assigns every gene in a given genome to one or more operons. Recall that our definition of an operon is a sequence of one or more genes that are transcribed as a unit under some conditions.

In our discussion here, we will focus on a special case of an operon map – one that assigns each gene in a genome to exactly one operon. We refer to this special case as a *one-operon-per-gene* map. We address

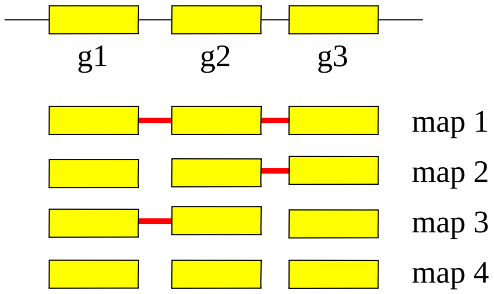

Figure 3: Alternative operon maps for a run of genes. The top of the figure shows a run of three genes $\langle g1, g2, g3 \rangle$. Below this run are shown the four possible *one-operon-per-gene* operon maps for this run. Each map assigns every gene to exactly one hypothesized operon, where operons are indicated by bars connecting genes. For example, *map 1* assigns all three genes to the same operon, and *map 4* assigns each gene to its own operon.

this special case for several reasons. First, we believe that it provides a pretty good approximation to the true operon map of a given organism. In other words, under most conditions, most genes are part of only one operon. Second, we believe that this *one-operon-per-gene* map will be more accurately predicted and easier to interpret than a general operon map. And third, we have developed an efficient algorithm for finding the optimal *one-operon-per-gene* map given certain assumptions. Figure 3 illustrates the concept of a *one-operon-per-gene* map.

The remainder of this section describes our algorithm for determining the optimal *one-operon-per-gene* map. First, we address the issue of how to evaluate a given map, and then we describe a dynamic programming algorithm for finding the optimal map.

Evaluating a Map

We have already discussed how we can evaluate a candidate operon, now we turn our attention to the matter of evaluating a candidate operon *map*. That is, given several competing operon maps, how can we score them so that we can decide which one appears to be most probable.

We define a *run* of genes to consist of a sequence of consecutive genes that is uninterrupted by either (i) an RNA gene, (ii) a gene on the opposite strand, or (iii) a known operon[1]. We assume that operons do not include RNA genes and that they do not "bridge" genes on the opposite strand. Thus we can process each run of genes separately when building an operon map. Under these assumptions, the first gene in a run must be the first gene in some operon, and the last gene in a run must be the last gene in some operon.

Given a candidate operon map for a run of genes R

[1]When conducting computational experiments, we require that such a known operon be in the training set; otherwise it is treated as any other candidate operon.

we can evaluate it by determining how probable the map is given the run:

$$\Pr(\text{map}|R) = \frac{\Pr(R|\text{map})\Pr(\text{map})}{\Pr(R)}. \qquad (11)$$

Assuming a uniform prior over the maps under consideration, this simplifies to:

$$\Pr(\text{map}|R) \propto \Pr(R|\text{map}). \qquad (12)$$

Thus we can evaluate a map by asking how probable the run of genes is given the map. Assuming that the genes are independent of one another we have the following:

$$\Pr(R|\text{map}) = \prod_{g_i \in R} \sum_{o_j \in \text{map}} \Pr(g_i|o_j)\Pr(o_j). \qquad (13)$$

Here g_i represents the *ith* gene in the run R, o_j represents the jth candidate operon in the map, and $\Pr(o_j)$ represents the probability that o_j actually is an operon.

To understand the motivation for this formulation, consider a clustering problem in which we want to find the centers of a fixed number of clusters (e.g. a mixture of Gaussians) One way to evaluate a particular clustering is to ask how probable the available data is given the cluster centers. This is typically done by assuming that the data points are independent, and then considering the probability of each point given the clustering. The probability of a point under the clustering is typically given by the product of the probability that each cluster would have emitted the point and the prior probability for the cluster. Our task is analogous in the following way. We want to partition the run of genes into operons, thus each gene is like a data point and each hypothesized operon in the map is analogous to a cluster. To determine the probability of each gene given the map, we ask how likely it is that the gene is explained by each operon.

We can simplify the expression above because there is a deterministic relationship between candidate operons and genes. Either a gene is included in a candidate operon or it is not. Therefore, the term $\Pr(g_i|o_j)$ is 1 for exactly one operon in the map, and 0 for the rest. Thus we have:

$$\Pr(R|\text{map}) = \prod_{g_i \in R} \Pr(o_j) \qquad (14)$$

where o_j is the operon that encompasses g_i in the given map. Instead of taking into account each gene in the run, as we do in this expression, we can simply consider the length of each operon in the map. Making this simplification and taking the logarithm of $\Pr(R|\text{map})$, we get our score for a given operon map:

$$\text{Score}(\text{map}) \equiv \log \Pr(R|\text{map}) = \sum_{o_j \in \text{map}} |o_j| \times \log \Pr(o_j). \qquad (15)$$

Here $|o_j|$ indicates the length (in terms of genes) of operon o_j. $\Pr(o_j)$, the probability that some candidate is an operon, is determined by the learned model for recognizing operons that we described earlier.

Dynamic Programming Approach

Now that we have discussed how to score a candidate operon map, we turn our attention to the task of finding the optimal map for a run of genes. The notion of optimality here is the best *one-operon-per-gene* map, given the scoring scheme described above.

Given a run of genes $R = \langle g_1 ... g_l \rangle$, suppose we score each candidate operon in this run using the approach above and store these scores in a two-dimensional array S. The element $S(i,j)$ in this array holds the score of the operon of length j that ends at position i in the run.

Now, using a dynamic programming approach, we fill in an array M to determine the optimal map. The element $M(i)$ represents the score of the best map through position i. This partial map, represented by $M(i)$, must assign every gene in $\langle g_1 ... g_i \rangle$ to exactly one operon, and none of these operons may extend beyond g_i. The final element of the array $M(l)$ therefore represents the best map that assigns every gene in the run to exactly one operon, none of which extends beyond the run. This is what we have defined as the optimal *one-operon-per-gene* map.

We initialize the array M by setting $M(0) = 0$, and we use the following recurrence relation to determine the value for every other value $M(i)$:

$$M(i) = \max \begin{cases} S(i,1) & + & M(i-1), \\ S(i,2) & + & M(i-2), \\ & \vdots & \\ S(i,i) & + & M(i-i). \end{cases} \qquad (16)$$

In order to recover the optimal map, we simply need to keep track of how we obtained the maximum value for each element $M(i)$. In other words, we need to keep track of the length of the last operon in the map represented by each $M(i)$.

It is easy to see that this algorithm can be implemented using a pair of nested loops in which the outer loop ranges over i, and the inner loop ranges over the possible alternatives for determining a given $M(i)$. The time complexity of this approach is thus $O(l^2)$ where l is the length of the run being processed.

Theorem 1 *Given any run of l genes, the algorithm returns an optimal (maximum scoring) one-operon-per-gene map of that run, and $M(l)$ is the score of this optimal map.*

Proof: Actually we prove the following stronger result: for each $1 \leq k \leq l$, the algorithm finds an optimal operon map ending with the operon of length k (i.e., the operon consisting of the last k genes in the run), and $S(l,k) + M(l-k)$ is the score of this map. Because every operon map must end with an operon having some length $1 \leq k \leq l$, and because the algorithm returns the map having the maximum $S(l,k) + M(l-k)$ (to which $M(l)$ is then set), this result implies the theorem.

Consider an arbitrary run R of l genes. For any $1 \leq k \leq l$, let $map(R, k)$ denote the operon map computed by the algorithm for R that ends in the operon of length k (last k genes in R). Let $|map(R, k)|$ denote the number of operons in this map. We show by induction on $|map(R, k)|$ that $map(R, k)$ is optimal among the operon maps that end with the operon of length k, and $S(l, k) + M(l-k)$ is the score of $map(R, k)$. For the base case, $|map(R, k)| = 1$ so R must consist of exactly k genes. Hence there exists only one operon map ending with the operon of length k, which is $map(R, k)$, so it necessarily is optimal. Furthermore, the score of $map(R, k)$ is simply $S(l, k) + 0 = S(l, k) + M(0) = S(l, k) + M(l - k)$ as desired.

For the inductive case, $|map(R, k)| \geq 1$. Consider the map consisting of all but the last operon in $map(R, k)$. By the inductive hypothesis, this map must be an optimal map for the run R', where R' consists of the first $l - k$ genes in R. This map for R' must end in an operon of some length $1 \leq k' \leq l - k$. Therefore we may denote this map for R' as $map(R', k')$. Again by the inductive hypothesis, its score must be $S(l - k, k') + M(l - k - k')$. Because $map(R', k')$ is optimal for R' and R' is the run from 1 to $l - k$, $M(l - k) = S(l - k, k') + M(l - k - k')$. Because $map(R, k)$ is $map(R', k')$ followed by the operon consisting of the final k genes in R (which has score $S(l, k)$), the score of $map(R, k)$ is $S(l, k) + M(l - k)$ as desired. It remains to verify that $map(R, k)$ is optimal among maps of R ending with the operon of length k. Suppose for the sake of contradiction that some other operon map for R also ends with the operon consisting of the last k genes of R yet scores higher than $map(R, k)$, i.e., has a score higher than $S(l, k) + M(l - k)$. Then this operon map must begin with a better (higher scoring) map for R' than the score of $M(l - k)$, contradicting that our map for R' was optimal. □

Empirical Evaluation

In this section we evaluate the predictive accuracy of our approach. We run 10-fold cross-validation experiments with a data set consisting of 365 known operons and 6633 sequences of genes thought not to be operons.[2] There are several questions that we want to answer in our experiments:

- What is the overall accuracy of our approach?
- How does the accuracy of our constructed operon map compare to the accuracy obtained by classifying isolated candidate operons?
- What is the predictive value of individual features?

To address the first two questions above, we consider two approaches to classifying candidate operons. In the first, we use our learned naive Bayes models to classify candidate operons. These classifications are made independent of one another. In the second approach, we (i) use our learned models to assign a probability to each candidate operon, and then (ii) use our dynamic programming approach to construct a genome-wide operon map based on the predicted probabilities. Since this second step enforces a global consistency to our predictions by assigning every gene to exactly one operon, it effectively changes the classifications of some candidate operons.

As a baseline, we also consider the expected accuracy that results from randomly selecting operon maps from a uniform distribution over the possible maps. We can determine this expected value by computing for each candidate operon c in a test set, the probability that the candidate would be a predicted operon in a randomly chosen map:

$$\Pr(c \in \text{map}) = \frac{1}{2^{|c|+1-b-e}}. \tag{17}$$

Here $|c|$ represents the length of the operon, $b = 1$ if the candidate is at the beginning of a run of genes and $b = 0$ otherwise, and $e = 1$ if the candidate is at the end of a run of genes and $e = 0$ otherwise. This probability tells us how frequently a positive example would be correctly classified or a negative example would be incorrectly classified in a randomly selected map.

For each training/test set split, we learn new promoter and terminator IMMs, leaving out of these model's training sets those promoters and terminators that are associated with test-set operons. In this way, we can ensure that our operon predictions are not biased by using information that is closely linked to a given test case (i.e., a known promoter or terminator), that we would not have in the case of a currently undiscovered operon.

Table 1 shows the overall accuracy rates for this experiment, as well as the false positive and true positive rates. The false positive (FP) rate is defined as $\frac{FP}{FP+TN}$, and the true positive (TP) rate is defined as $\frac{TP}{TP+FN}$. The first row shows the results for the learned operon models when they are treated as classifiers. The second row shows the results when we use our dynamic programming method to construct an operon map, and use the map to classify operon candidates. The net effect of altering our predictions by constructing a map is that we do not correctly identify as many true operons, but on the other hand, we do not predict nearly as many false positives. Overall, the accuracy of the predictions made by the map are slightly better than the predictions made by the classifiers alone. We consider this to be an encouraging result, especially since

[2] Of the 6633 non-operons, only 5145 actually appear in some test set. The reason for this is subtle. We randomly distribute the known operons into the 10 test sets. For each known operon, we assign all of the overlapping negative examples to the same test set *except* for those examples that *also* overlap a known operon in the corresponding *training* set. This process ensures that no test-set example overlaps an example in the corresponding training set. Since non-operons can overlap multiple known operons, this process leads to 1488 non-operons not appearing in any test set.

Table 1: Predictive accuracy for the learned models alone (naive Bayes), the operon map made from the predictions of the learned models, and for randomly chosen operon maps.

method	acc. (%)	FP rate (%)	TP rate (%)
naive Bayes	90.0	8.9	74.5
operon map	94.1	4.0	67.7
random map	87.0	9.1	30.8

the operon map has the added benefit of ensuring that our operon predictions are consistent with one another (given the *one-operon-per-gene* assumption). Also note that this task is difficult in that the number of "non-operon" gene sequences that overlap a true operon is generally quadratic in the length of the true operon. Thus, there are many opportunities for an inaccurate prediction to reduce the accuracy of our predictions on true operons. Finally, note that the accuracy criterion we use here is very stringent. A classification is considered either right or wrong; there is no partial credit for cases in which we predict the extent of an operon almost correctly.

We perform two experiments to evaluate the contribution of our individual features to the predictive accuracy of our models. In the first experiment, we consider making our predictions *using only* a single feature, or a small group of closely related features. We collectively refer to both of these cases as "feature groups." In the second experiment, we learn models that *leave out* a single feature group. Table 2 shows the accuracy of operon maps made using models that consist of single feature groups, and Table 3 shows the accuracy of operon maps made using models that leave one feature group out. For reference, both tables also include the result from Table 1 for the naive Bayes models that use all features.

Tables 2 and 3 illustrate several interesting points. First, the features vary quite a bit in their predictive accuracy. Second, none of the individual feature groups is as predictive as the model as a whole. This result indicates the value of combining evidence from a variety of sources.

Table 3 shows that, in most cases, leaving a single feature group out of the model does not adversely affect its predictive accuracy. The feature group that has the most impact when left out is the functional annotation group. This result is somewhat disappointing since we suspect that including this feature biases our results to some extent. The explanation for this concern is that since the functional annotation codes have been assigned to genes manually by people with knowledge of the known operons, they possibly allow information about true operons to "leak" into our test sets. We conjecture that the known operons are more consistent and complete in their functional annotation than many true, but unknown operons. Unfortunately, we do not know how to estimate the extent of this bias nor correct for it.

Table 2: Predictive accuracy for the operon maps made using individual feature groups.

representation	FP rate (%)	TP rate (%)
using all features	4.0	67.7
functional annotation	4.5	65.7
within-operon spacing	5.9	59.4
expression data	5.5	46.6
promoter	4.6	44.1
terminator	7.2	41.3
operon size	15.0	37.7
neighboring genes	4.5	37.2

Table 3: Predictive accuracy for the operon maps made *leaving out* individual feature groups.

representation	FP rate (%)	TP rate (%)
leaving out none	4.0	67.7
terminator	3.9	68.7
promoter	4.1	68.7
neighboring genes	4.4	68.7
expression data	4.2	68.1
operon size	3.4	67.9
within-operon spacing	3.6	66.3
functional annotation	4.3	61.0

This issue of some features being more informative for our labeled examples than they will be in general also crops up in the case of our promoter and terminator features. However, in this case, we ensure that our experiments are not biased by leaving out of the promoter (terminator) training sets those promoters (terminators) associated with test-set operons.

We can get a better sense of the predictive value of our models by considering ROC (receiver operating characteristic) curves. An ROC curve shows the relationship between the true positive and false positive rates as we vary a threshold on the confidence of our predictions. For example, the results in the first row in Table 1 were determined by treating our naive Bayes model as a classifier; when the posterior probability of a candidate operon was greater than 0.5, we classified it as an operon. The ROC curve shown in the upper left part of Figure 4, on the other hand, shows how the accuracy of this model changes as we vary the threshold on this posterior probability. We get a single ROC curve from our 10 learned models by pooling the test-set predictions of these models. This curve is informative since, unlike the overall accuracy numbers in Table 1, it does not depend on prior probabilities of the two classes or any particular misclassification costs (this is a property of ROC curves). Figure 4 also illustrates that our learned models have considerable predictive value. A model that guessed randomly would result in an ROC "line" defined by: TP rate = FP rate.

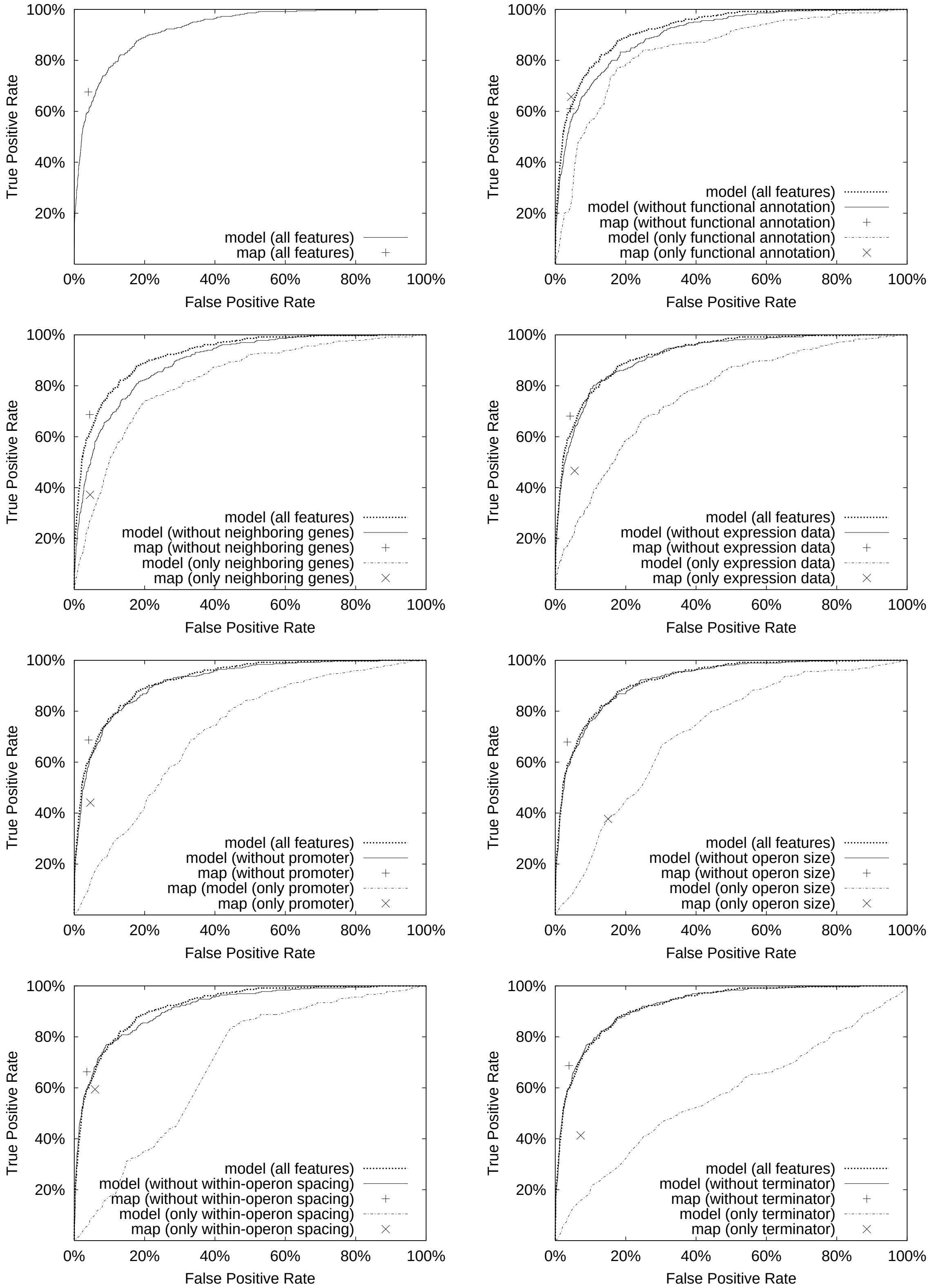

Figure 4: ROC plots. The ROC curve in the upper left plot represents our naive Bayes model, and the point in this figure represents the predictive accuracy of the operon map produced by our dynamic programming algorithm. The other seven plots in this figure illustrate the predictive power of individual feature groups. Each of these plots shows (i) the ROC curve for predictions made leaving out the given feature group, (ii) the ROC curve for predictions made using only the given feature group, and (iii) the points corresponding to operon maps constructed from both of these models. Additionally for reference, each plot repeats the ROC curve from the upper left plot (for the model that uses all of the features).

Note that this curve represents the accuracy of the naive Bayes model classifying candidate operons independently of one another. It is not obvious how we can generate ROC curves for our operon maps, but we can compare the predictive accuracy of our operon maps to the ROC curve for our naive Bayes models by plotting the operon map as a single point in this space. The plot in the upper left part of Figure 4 shows that this point lies above the ROC curve for the naive Bayes model. This result indicates that the operon map results in accuracy that is superior to the accuracy of the naive Bayes predictions alone.

ROC curves can also provide a better indication of the predictive value of our individual features. The other plots in Figure 4 provide an ROC analysis of our individual feature groups. Each of these plots shows (i) the ROC curve for predictions made leaving out the given feature group, (ii) the ROC curve for predictions made using only the given feature group, and (iii) the points corresponding to operon maps constructed from both of these models. Additionally for reference, each plot repeats the ROC curve from the upper left plot (for the model that uses all of the features).

Figure 4 reinforces several points that were made earlier. First, no individual feature group is as predictive as the models that use all features. Second, in most cases, leaving out an individual feature group does not weaken predictive accuracy. Third, the process of constructing an operon map, which enforces consistency among predictions, nearly always provides a boost in predictive accuracy.

Conclusions

We have presented an approach to predicting operons in prokaryotic genomes that involves two major components. The first component involves learning naive Bayes models that use a rich variety of data types – sequence data, expression data, etc. – to estimate the probability that a given sequence of genes constitutes and operon. The second component is a dynamic programming method that constructs an *operon map* for an entire genome or part of one. By assigning every gene to its most probable operon, the operon map provides a consistent interpretation of the predictions made by the model.

The empirical evaluation of our approach indicated that operons can be predicted with fairly high accuracy, and there is value in combining evidence from various data sources. Moreover, the operon maps produced by our dynamic programming method are more accurate than the individual predictions generated by the naive Bayes models.

Another lesson of our experiments pertains to the relative value of gene expression data. There is currently great interest in identifying sets of related genes and discovering regulatory relationships using expression data like ours. Our results suggest a cautionary note here: we were able to obtain much more accurate operon predictions by considering other types of data in conjunction with expression data.

There are several main issues that we plan to investigate in future research. First, there are additional sources of evidence, such as the binding sites of various regulatory proteins, that we plan to incorporate into our models. Second, we are investigating alternative representations for several of our feature types, including expression data and terminators. We believe that our current terminator models do not provide much predictive value because they use a representation which is unable to capture important information about RNA base pairing. We are currently developing an approach to predicting terminators that uses stochastic context free grammars, which are able to represent base-pairing information. Third, since more gene expression data is becoming available, we will be incorporating this additional data into our predictions. Moreover, we plan to investigate methods for handling this data that take into account the relationships among the experiments that are generating the data (some experiments are much more closely related than others). Fourth, we plan to begin addressing the top-level learning tasks shown in Figure 1 that motivated our work on the operon prediction task.

A fundamental challenge facing the computational biology community is determining the functions of genes in newly determined genomes, and the relationships among these genes. We argue that this task is best addressed by employing a wide array of data sources as evidence, and we believe that the work presented here represents a promising first step in this general approach.

Acknowledgments

This research was supported by NSF Grant IRI-9502990, NIH Grant R01 GM35682-15A1, and the University of Wisconsin Graduate School. Thanks to Fred Blattner, Christina Kendziorski, Nicole Perna and Craig Richmond for helpful discussions regarding this research.

References

Blattner, F. R.; Plunkett, G.; Bloch, C. A.; Perna, N. T.; Burland, V.; Riley, M.; Collado-Vides, J.; Glasner, J. D.; Rode, C. K.; Mayhew, G. F.; Gregor, J.; Davis, N. W.; Kirkpatrick, H. A.; Goeden, M. A.; Rose, D. J.; Mau, B.; and Shao, Y. 1997. The complete genome sequence of Escherichia coli K-12. *Science* 277:1453–1474.

Brown, M. P. S.; Grundy, W. N.; Lin, D.; Cristianini, N.; Sugnet, C.; Ares, M.; and Haussler, D. 1999. Support vector machine classification of microarray gene expression data. Technical Report UCSC-CRL-99-09, Department of Computer Science, University of California, Santa Cruz.

Chen, Y.; Dougherty, E. R.; and Bittner, M. L. 1997. Ratio-based decisions and the quantitative analysis of cDNA microarray images. *Journal of Biomedical Optics* 2(4):364–374.

Craven, M.; Page, D.; Shavlik, J.; Bockhorst, J.; and Glasner, J. 2000. Using multiple levels of learning and diverse evidence sources to uncover coordinately controlled genes. In *Proceedings of the Seventeenth International Conference on Machine Learning*. Stanford, CA: Morgan Kaufmann.

Eisen, M. B.; Spellman, P. T.; Brown, P. O.; and Botstein, D. 1998. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences, USA* 95(25):14863–14868.

Nature Genetics Supplement. 1999. The chipping forecast. Volume 21, January.

Friedman, N.; Linial, M.; Nachman, I.; and Pe'er, D. 2000. Using Bayesian networks to analyze expression data. In *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology (RECOMB)*.

Fu, L., and Buchanan, B. G. 1985. Learning intermediate concepts in constructing a hierarchical knowledge base. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 659–666.

Jelinek, F., and Mercer, R. L. 1980. Interpolated estimation of Markov source parameters from sparse data. In Gelsema, E. S., and Kanal, L. N., eds., *Pattern Recognition in Practice*. North Holland. 381–397.

O'Sullivan, J. 1998. Transferring learned knowledge in a lifelong learning mobile robot agent. In *Proceedings of the 7th European Workshop on Learning Robots*.

Overbeek, R.; Fonstein, M.; D'Souza, M.; Pusch, G. D.; and Maltsev, N. 1999. The use of gene clusters to infer functional coupling. *Proceedings of the National Academy of Science (USA)* 96:2896–2901.

Riley, M. 1996. E.coli gene products: Physiological functions and common ancestries. In Curtiss, R.; Lin, E.; Ingraham, J.; Low, K.; Magasanik, B.; Neidhardt, F.; Reznikoff, W.; Riley, M.; Schaechter, M.; and Umbarger, H., eds., *Escherichia coli and Salmonella*. American Society for Microbiology, 2nd edition. 2118–2202.

Salgado, H.; Santos-Zavaleta, A.; Gama-Castro, S.; Millán-Zárate, D.; Blattner, F. R.; and Collado-Vides, J. 2000. RegulonDB (version 3.0): Transcriptional regulation and operon organization in Escherichia coli K-12. *Nucleic Acids Research* 28(1):65–67.

Salzberg, S.; Delcher, A.; Kasif, S.; and White, O. 1998. Microbial gene identification using interpolated Markov models. *Nucleic Acids Research* 26(2):544–548.

Shapiro, A. 1987. *Structured Induction in Expert Systems*. Reading, MA: Addison Wesley.

Snyder, E. E., and Stormo, G. D. 1993. Identification of coding regions in genomic DNA sequences: An application of dynamic programming and neural networks. *Nucleic Acids Research* 21(3):607–613.

Stone, P. 1998. *Layered Learning in Multi-Agent Systems*. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University. Technical report CMU-CS-98-187.

Tamames, J.; Casari, G.; Ouzounis, C.; and Valencia, A. 1997. Conserved clusters of functionally related genes in two bacterial genomes. *Journal of Molecular Evolution* 44:66–73.

Xu, Y.; Mural, R. J.; and Uberbacher, E. C. 1994. Constructing gene models from accurately-predicted exons: An application of dynamic programming. *Computer Applications in the Biosciences* 10(6):613–623.