Learning from Imperfect Data in Theory and Practice

by

Donna Karen Slonim

M.S., University of California at Berkeley (1991)
B.S., Yale University (1990)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1996

Signature of Author ______________________________

Department of Electrical Engineering and Computer Science
May 3, 1996

Certified by ______________________________

Ronald L. Rivest
Professor of Computer Science
Thesis Supervisor

Accepted by ______________________________

F. R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Learning from Imperfect Data in Theory and Practice

by

Donna Karen Slonim

Submitted to the Department of Electrical Engineering and Computer Science
on May 3, 1996,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Abstract

This thesis explores several problems of learning with noisy or incomplete data. Most machine learning applications need to infer correct conclusions from available information, although some data may be incorrect and other important data may be missing. Human learners often compensate effortlessly for imperfect data. Even animals can learn from conditioning in which they are rewarded inconsistently. However, it seems difficult to build such fault-tolerance into machine learning systems. In this thesis, we describe algorithms for handling imperfect data in several projects that range from the theoretical to the practical.

In Chapter 2 we present new formal models of learning with a teacher who makes mistakes or fails to answer some questions, and we show that learning can succeed in these models. We first consider learning with a "randomly fallible teacher" who is unable to answer a random subset of the learner's questions. We present a probabilistic algorithm for learning monotone DNF formulas in this model; asymptotically, our algorithm runs as quickly as the error-free algorithm even when half the questions remain unanswered. We then introduce a learning model in which queries on "borderline" examples may recieve incorrect answers. We describe efficient algorithms for learning intersections of halfspaces and subclasses of DNF formulas in this new model. Our positive results are the first in a learning model where the teacher's ignorance depends in a realistic way on the question asked.

Our results in Chapter 3 show how teams of learners can work together to learn graphs in the absence of key information that distinguishes nodes. On a graph with indistinguishable nodes, a robot cannot tell if it is placed on a node that it has previously seen. This problem models that of a real robot learning with imperfect or missing sensor data. We describe a probabilistic polynomial-time algorithm for two cooperating robots to learn any strongly-connected directed graph, even graphs that would most likely require exponential time to explore by walking randomly. We also present a random-walk algorithm that is more efficient than the previous algorithm for the special class of graphs with high conductance. Our work illustrates that even in the extreme case where the environment provides no information for distinguishing nodes (or where the robots'

landmark-recognition sensors fail completely), there is sufficient information available for two cooperating robots to learn the map perfectly.

In Chapter 4 we examine the application of machine learning techniques and algorithm design to a real problem in molecular biology: building large-scale human gene maps using the new technique of radiation hybrid mapping. We represent uncertainty about noise in the data with a hidden Markov model. Our algorithms then search an exponentially-large space for maps that are likely to have produced the observed data. While the theoretical model guides our search, alone it is insufficient. Thus we investigate several practical methods for solving the problem despite the limits of the model. Finally, we use these methods to build the first radiation hybrid map of the entire human genome. Our work demonstrates that an approach combining theoretical models and practical search heuristics can yield excellent results in a real application of learning from imperfect data.

Keywords: computational learning theory, machine learning, computational biology, query learning, imperfect teachers, graph exploration, noisy data, radiation hybrids, hidden Markov models, physical mapping, genome mapping.

Thesis supervisor: Ronald L. Rivest

Title: Professor of Computer Science

Acknowledgments

First, I want to acknowledge the help and guidance of my thesis advisor, Ron Rivest. Ever since I decided I wanted to work with Ron because his desk (what you could see of it) sported a well-used thesaurus and a copy of Bartlett's Quotations, I have been astounded by his ability to offer brilliant advice on almost any topic. Over the years, Ron has patiently listened to innumerable half-baked ideas and has suggested unexpected paths around a variety of obstacles, ranging from unprovable conjectures to writer's block to job negotiations. His calm and rational advice has been essential to my survival of several years as a graduate student and to my development as a researcher.

Many other people have contributed their ideas and suggestions to this thesis. I am grateful to Eric Lander and Tomás Lozano-Pérez for serving on my thesis committee and for their constructive comments that helped improve this document. Special thanks to my coauthors: Dana Angluin, Avrim Blum, Michael Bender, Prasad Chalasani, Sally Goldman, Leonid Kruglyak, Eric Lander, and Lincoln Stein, for being great fun to collaborate with and for allowing me to include our joint work in this thesis. Many thanks to Lisa Tucker-Kellogg and Rob Schapire for their insightful comments on earlier drafts of some of the work presented here. I am indebted to Barbara Bryant, Eric Jordan, Jeff Keller, Peter Orbaek, Dana Ron, and especially Mark Torrance, not only for sitting though a practice defense talk, but for their penetrating questions and comments that helped transform an acceptable talk into a good one. Thanks also to Lester Hui and Lois Bennett for spending an entire afternoon helping me convert Macintosh PICT files into postscript so I could include pictures of maps in Chapter 4.

During my years as a graduate student, I have enjoyed the financial support of a National Science Foundation graduate fellowship and of a National Institutes of Health Training Grant in Genomic Sciences (grant # 1 T32 HG00039-01). Thanks to all those who helped administer these grants, especially Jackie Sciacca, Judy Halloran, Lissa Natkin, Julie Morrill, and Paul Matsudaira.

I would also like to acknowledge the many people whose indirect contributions helped make this thesis possible. I want to express special gratitude to Dana Angluin, my mentor and undergraduate advisor, for encouraging my research interest in theoretical computer science, for convincing me that whatever success I have had is not *entirely* due to luck (some of it, no doubt, can be credited to her good advice and friendship!), and for introducing me to many of the wonderful people I have worked with in the COLT community. I owe a great debt to Lenore Cowen and Merrick Furst, who helped convince me to stay in graduate school after completing my Master's degree. And I will always be grateful to Charles Leiserson for teaching me about teaching.

Many thanks to my past and present office-mates: Lenore Cowen, Pearl Tsai, Tal Rabin, Trevor Jim, and Peter Orbaek, and to all the TOC folks who have made the third floor of LCS such a fun place to work. To David Jones, for helping me comprehend the vagaries both of LaTeX and of the Area II Committee, and for dragging me out to Toscanini's when it looked like I needed it, even if it was snowing. To Toscanini's, for being the only ice-cream shop outside of New York City that actually knows how to make an egg cream. And to Be Hubbard for everything: helping me thread my way through Ron's busy schedule; being supportive and cheerful through several years of administrative nightmares, an area exam, and a job search; administering chocolate when necessary; and always having a sense of humor about it all.

Thanks to Andy Oakland, for his understanding through many tough times and for helping me to embody that rare paradox: a graduate student with a *life*! And finally, I would like to acknowledge the help of my parents, Harriet and Michael Slonim, who have supported me enthusiastically through every endeavor, some even more Herculean than this. I thank them for all their patience, compassion, and good advice, and I dedicate this thesis to them.

Publication Notes

Much of the work in this thesis has been published elsewhere in some form.

The work outlined in Section 2.2 is joint work with Dana Angluin. This work has appeared in the journal *Machine Learning* [13], and an extended abstract was published in the *Proceedings of the Fourth Annual Workshop on Computational Learning Theory* [12]. The work described in Section 2.3 is joint work with Avrim Blum, Prasad Chalasani, and Sally Goldman. An extended abstract of this paper appeared in the *Proceedings of the Eighth Annual Conference on Computational Learning Theory* [23].

Chapter 3 is joint work with Michael Bender. This work was published in extended abstract form in the *Proceedings of the 35th Annuanl Symposium on Foundations of Computer Science* [19].

The work in Chapter 4 is part of a large collaborative project at the Whitehead Institute/MIT Center for Genome Research. The integrated map project has been described in the journal *Science* [60]; this paper includes some of the results in Section 4.7. A joint paper with Leonid Kruglyak, Eric Lander, and Lincoln Stein, describing our work on radiation hybrid mapping in more detail, will appear soon [104].

Table of Contents

Chapter 1

Introduction

Machine learning is a crucial aspect of many computer applications. Consider the problem of designing a computer system to aid in medical diagnosis. It would be impractical to require the designer to pre-program the system with every possible combination of symptoms that might occur. Rather, one would prefer to design a system that can learn from examples and draw conclusions about how to handle new situations.

Another example might be a biologist studying the relationships between protein amino-acid sequences and the proteins' functions in the body. Since computers are better suited than humans to finding patterns in vast amounts of sequence data, a computer program might detect novel correlations, suggesting areas for future research. In this case the computer *must* learn from the data; no one can explicitly tell the computer what to look for, since no one knows precisely what is wanted. As the role of technology in society increases further, the demand of applications for machine learning will expand accordingly. Thus the development of algorithms that learn is a major challenge in computer science.

An essential aspect of any practical learning algorithm is the need to learn from imperfect data. Few real-world problems operate under perfect conditions. The medical diagnosis system may have to learn from inconsistent data provided by different doctors, or may be missing information about crucial symptoms. The biologists' data may be derived from experiments that are subject to many types of errors. Some crucial experiments might produce inconclusive results. Nonetheless, even imperfect data can contain a great deal of valid information. Therefore we would like to design systems that learn as much as possible from the data available.

This thesis explores several problems of learning with noisy or incomplete data. Most machine learning applications need to infer correct conclusions from available information, even though some data may be incorrect and other important data may be missing. Human learners often compensate effortlessly for imperfect data. Even animals can learn from conditioning in which they are rewarded inconsistently. However, it seems difficult to build such fault-tolerance into machine learning systems. In this thesis, we describe algorithms for handling imperfect data in several projects that range from the theoretical to the practical.

Each chapter of the thesis treats a different aspect of the problem of learning with imperfect data. Chapter 2 discusses some theoretical problems of machine learning with the help of imperfect teachers. In this chapter, we present new models of learning with a teacher that makes mistakes or fails to answer some questions, and we show that formal learning can succeed in these models. Our results in Chapter 3 show how teams of learners can work together to learn graphs in the absence of key information that distinguishes nodes. While this work is also theoretical, it may have applications in designing exploration algorithms for robots in unknown environments.

In Chapter 4, we look at applications of machine learning techniques and algorithm design to a real problem in molecular biology: building large-scale human gene maps using the new technique of radiation hybrid mapping. Unlike the previous two chapters, in which the learner is generally a machine, our goal in this chapter is to use computers to help *us* learn from imperfect data. Many techniques from the machine learning literature are applicable to this problem. Furthermore, the project described in this chapter is a case study that explores the tradeoffs between theory and practice. We develop a theoretical model of the noisy data and then explore practical methods for solving the problem despite the limits of our model. Finally, we demonstrate that such hybrid methods can be successful: we describe how we have constructed the first radiation hybrid map of the entire human genome.

While the three projects described in this thesis are clearly very different, there are some common threads that bind them together. Each project illustrates the union of the theoretical and the practical: our theoretical models in Chapters 2 and 3 are informed by the need to represent noisy or incomplete information in more realistic ways, while the real application in Chapter 4 is solved through the use of theoretical modeling. In each project, the learner is faced with the task of drawing accurate conclusions from imperfect data. And in each case, we solve the problem by presenting algorithms that we prove, either theoretically or empirically, can successfully learn from imperfect data.

The rest of this chapter presents a more detailed overview of each part of the thesis.

Imperfect Teachers

The first part of the thesis discusses formal models of learning. This work is in the field of computational learning theory. Since its inception with Valiant's seminal paper [108] in 1984, this field has sought to define new mathematical models of machine learning, to design efficient learning algorithms within these models, and to apply the techniques and knowledge developed in this process to practical learning problems.

Within this paradigm, Chapter 2 focuses on the problem of concept learning with the help of an imperfect teacher. In concept learning, there is some known domain $\mathcal{X}$ of objects and an unknown target concept $f \subseteq \mathcal{X}$. The learner's goal is to learn efficiently to classify objects with respect to f. The learner does not know f, but is helped by the knowledge that f belongs to some known concept class $\mathcal{C} = \{f_1, f_2, \ldots\}$ (whose size may be infinite).

As input, the learner is presented with a number of randomly-chosen correctly-classified examples. For example, suppose that the domain is the set of points in the plane and the concept class $\mathcal{C}$ is the set of all concepts that can be represented as the intersection of two halfspaces in the plane. Then the learner might see the labeled examples shown in Figure 1.1a, where each point is labeled "+" if it is within the intersection of the two halfspaces and "–" otherwise. the learner's goal is to determine the target concept f shown in Figure 1.1b, given the labeled examples.

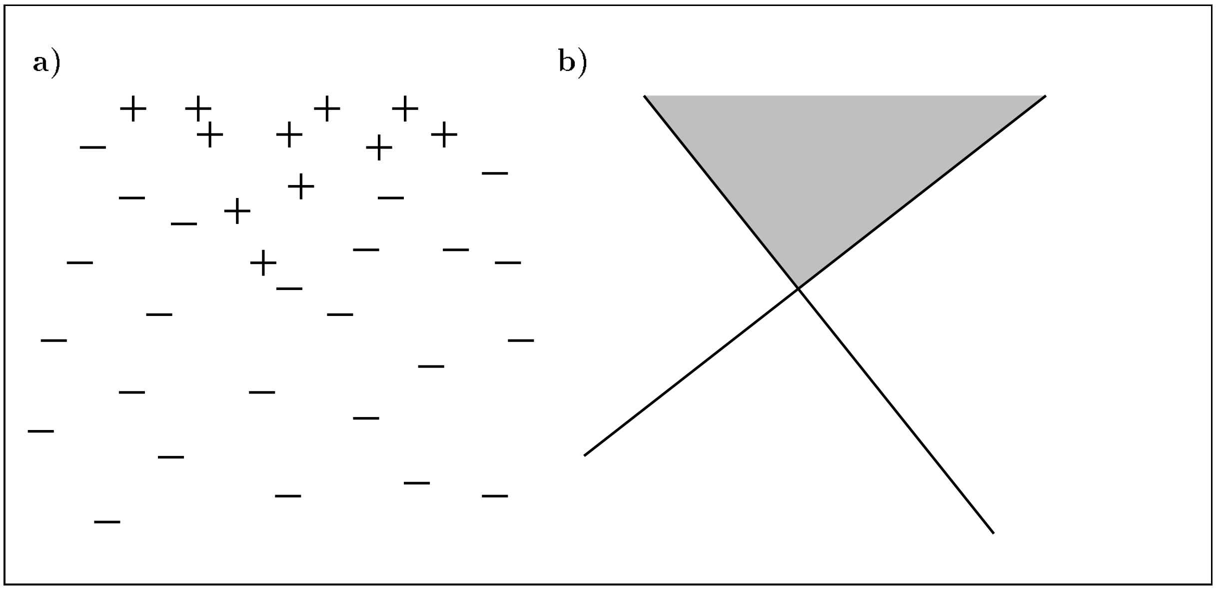


Figure 1.1: Learning the intersection of two halfspaces in the plane. a) The learner sees only labeled examples chosen at random from the domain of points in the plane. b) The learner's goal is to determine the underlying target concept, shown here. All positive examples are chosen from the shaded region.

In the stronger "query learning" model, the learner is also allowed to ask an omniscient teacher how to classify specific examples. Previous work has shown that a learner who asks questions can learn more complicated concepts than one who learns by passive observation alone. Our work explores what happens when the learner can ask queries, but the teacher is not omniscient.

Section 2.2 studies the problem of learning with a "randomly fallible teacher" who is unable to answer a random subset of the learner's questions. Our work in this section introduces the first general model of query learning with an imperfect teacher.

We describe an algorithm that with high probability learns certain classes of boolean formulas efficiently from this teacher, even when half the queries asked are not answered. Our results hold in a limited sense even when the teacher may answer incorrectly. By defining a measurable tradeoff between asking queries and learning from examples only, our model yields some insight into the degree of additional information that queries provide a learning algorithm.

In many cases, however, it is unrealistic to assume that a teacher's ignorance is random. The more recent work described in Section 2.3 proposes an intermediate approach, in which queries on examples near the boundary of a target concept may receive incorrect or "don't care" responses. The randomly-chosen examples, however, are not chosen from the boundary region and are labeled correctly. The motivation behind our model is that the boundary between positive and negative examples may be complicated or "fuzzy."

We describe efficient algorithms for learning intersections of halfspaces and subclasses of DNF formulas in this new model. Our positive results are the first in a learning model where the teacher's ignorance depends in a realistic way on the question asked.

Learning Graphs with Indistinguishable Nodes

Chapter 3 shows how teams of robots can work together to learn graphs in cases where a single robot alone would be helpless. Consider the problem of a robot trying to construct a street map of an unfamiliar city by driving around. Since many streets are one-way, the robot may be unable to retrace its steps, but it can learn by using street signs to distinguish intersections. However, if it is nighttime and there are no street signs (or if the robot's sensors provide imperfect data), the task becomes significantly more challenging.

We describe a polynomial-time algorithm for two cooperating robots to solve an abstraction of this problem. Instead of learning a city, they learn a strongly-connected directed graph. Nodes are indistinguishable, so a robot cannot tell if it is placed on a node that it has previously seen. However, the robots can see each other when they are both at the same node. They learn about the graph by moving around it and noting when they see each other.

This problem models that of a real robot learning with imperfect or missing sensor data. Robots that rely on vision for navigational purposes often get lost; their landmark-recognition systems can fail for a variety of reasons. Our work illustrates that even in the extreme case where the environment provides no information for distinguishing nodes (or where the robots' landmark-recognition sensors fail completely), there is sufficient information available for two cooperating robots to learn the map perfectly.

With high probability, two robots using our algorithm can learn any graph in polynomial time, even if they would expect to need exponential time to explore the graph by walking randomly. We also show that no probabilistic polynomial-time algorithm for a single robot can solve the same problem. Thus, our work demonstrates that two robots are strictly more powerful than one.

Algorithms for Building Human Genome Maps

An intense worldwide effort is underway to determine the location, DNA sequence, and function of human genes. Physical maps play an important part in this process. A physical map of a chromosome shows the relative locations and estimated distances between landmarks along the chromosome. A recent technique for building physical maps uses radiation hybrid data, which may be subject to many types of experimental error. Chapter 4 discusses the application of machine learning techniques and algorithm design to the problem of inferring accurate physical maps from noisy radiation hybrid data.

We use a hidden Markov model to represent uncertainty about noise in the data. The model allows us to estimate the likelihood that a given arrangement of the markers produced the observed data. We then try to find a likely arrangement of the markers that is as close to the true order as possible. Evaluating all possible arrangements of n markers requires $O(n!)$ computations. Since maps generally contain several hundred markers, exhaustive search is computationally infeasible.

We build our maps in a top-down fashion by first finding a sparse but accurate map that spans the entire region. Then we greedily add additional markers into the map, indicating the degree of confidence we have in each placement. Thus, researchers using the map know which markers to believe and which to treat with suspicion.

We also develop a greedy divide-and-conquer algorithm for ordering markers with a bottom-up approach. This algorithm seeks a maximum-likelihood order while obeying certain local constraints. We compare this algorithm to a number of experiments with standard combinatorial techniques and show that our software consistently finds better orderings on large data sets (all algorithms do Both our ordering schemes take advantage of local ordering information in addition to maximum-likelihood constraints and are much more effective than maximum-likelihood searching alone. Thus we conclude that while our theoretical model of the noisy data guides our search for good maps, it is insufficient by itself.

Finally, we use these methods to build the first radiation hybrid map of the entire human genome. Our work demonstrates that an approach combining theoretical models and practical search heuristics can yield excellent results in a real application of learning from imperfect data.

CHAPTER 2

Learning With Imperfect Teachers

2.1 Introduction

Imagine a student learning, for the first time, to recognize which animals are dogs and which are not. Suppose that to accomplish this task, the learner sits on a bench on the Boston Common with her teacher. As animals pass by, the teacher points to each one and tells the student whether or not it is a dog. If she sat there long enough, the learner would eventually learn to recognize most dogs and to distinguish them from other animals such as squirrels, horses, and birds. She could learn the concept "dog" quite well without ever seeing all of the dogs or all of the "non-dogs" in the world. However, her knowledge of dogs might not be perfect. It would be very unlikely for the student to see a wolf or a Chinook (a rare breed of sled dog) pass by. If she encountered one later, she might not realize that a Chinook is a dog or that a wolf is not. Nonetheless, she would be able to classify correctly most animals that are likely to appear in downtown Boston.

Learning by observing in this fashion is known as "passive learning," since the learner has no control over the examples seen. Sometimes, however, it is hard to learn quickly from passive observation alone. In an "active learning" model, the learner may ask the teacher questions as well as observe correctly-classified examples. For example, a student trying to learn about dogs might point to a police horse and ask the teacher whether it is a dog. Or she might generate the hypothesis "a dog is any animal with four legs," and ask the teacher if her hypothesis is correct.

In 1984, Valiant introduced a formal learning model intended to capture these notions [108]. The model is known as *distribution-free* or PAC-learning, where PAC stands for "probably approximately correct." Formally, a *concept* $f : \mathcal{X} \longrightarrow \{0,1\}$ is a boolean function over an instance space $\mathcal{X}$. In the example above, the concept "dog" is such a function on the instance space of all animals. In the rest of this chapter we consider two instance spaces: the boolean domain $\{0,1\}^n$ and the continuous domain R^n. A point $x \in \mathcal{X}$ is an *example*, and it is called a *positive example* of f if $f(x) = 1$ and a *negative example* of f if $f(x) = 0$. A concept class $\mathcal{C}$ is a set of such functions f, along with an associated representation language for describing them. For instance, $\mathcal{C}$ might be the class of all boolean formulas over n variables, represented in disjunctive normal form (DNF).

In the PAC learning model, to obtain information about an unknown target function $f \in \mathcal{C}$ the learner is shown labeled positive and negative examples of f, drawn randomly according to some unknown distribution D over $\mathcal{X}$. The learner is also given error parameters $\epsilon, \delta > 0$ as input. Its goal is to output the description of a function h that, with probability at least $1 - \delta$, has probability at most ϵ of disagreeing with f on a randomly drawn example from D. Thus, one can say that h is *probably* (with probability $\geq 1 - \delta$) *approximately correct* (has *error* $\leq \epsilon$). An algorithm $\mathcal{A}$ *PAC-learns* $\mathcal{C}$ if for any $f \in \mathcal{C}$, any distribution D, and any $\epsilon, \delta > 0$, $\mathcal{A}$ meets this goal and runs in time polynomial in n (the size of an example), $1/\epsilon$, $1/\delta$, and $|f|$ (the description length of the target function). A class $\mathcal{C}$ for which such an algorithm exists is said to be *PAC-learnable.*

In defining the learner's goal, we have not specified what a "description of a function

h" is. An algorithm is said to be a *proper* learning algorithm if the hypothesis h is always chosen from the description language associated with the concept class. On the other hand, we may allow a learning algorithm to output *any polynomial-time algorithm* as a hypothesis. This less constrained model is sometimes called "PAC-predictability" [58, 57].

Active learning is also known as "query learning," since the learner is allowed to ask queries of an omniscient oracle. Many types of queries are possible, but the two most commonly-studied questions are *membership queries* and *equivalence queries*. Membership queries ask if some example $x \in \mathcal{X}$ is in the target concept; the answer to a query $MQ(x)$ is $f(x)$. Equivalence queries ask if the learner's current hypothesis h is correct or not. The answer to an equivalence query $EQ(h)$ is a counterexample x such that $h(x) \neq f(x)$, if such a counterexample exists, and the empty set otherwise.

Sometimes it is difficult to specify the hypothesis h exactly and succinctly. However, one can use the PAC model to approximate an equivalence query without formally specifying the hypothesis h. Blumer, Ehrenfeucht, Haussler and Warmuth [27] proved that *any* hypothesis consistent with a labeled random sample of size

$$\Omega\left(\frac{\log 1/\delta}{\epsilon} + \frac{\log |\mathcal{C}|}{\epsilon}\right)$$

is a PAC-hypothesis; i.e., with probability at least $1 - \delta$, the hypothesis is ϵ-good. Therefore, to ask a probabilistic equivalence query, we simply draw a large enough sample and look for an object in the sample that is classified differently by the teacher and the learner. If such an object exists, it is a counterexample to h; otherwise, h is probably approximately correct. Thus, any class $\mathcal{C}$ that is learnable by equivalence queries alone is also PAC-learnable [5], though the converse is not true [22].

We use *PAC-memb* to refer to the variation of the PAC model in which the learner can make membership queries. Likewise we say that a concept class is *exactly learnable* if it is learnable with membership and equivalence queries. Many concept classes, such as read-once formulas, monotone DNF formulas, and deterministic finite automata, are known to be efficiently learnable in active learning models but are not PAC learnable. Thus, membership queries provide some additional power to a learning algorithm.

Chapter Overview

All of the standard learning models defined above assume that the learner has the help of a teacher who is omniscient and well-intentioned. In most machine-learning tasks, there is little incentive for the teacher to knowingly corrupt data for the sake of confusing the learner. (There are, of course, exceptions to this rule. For example, a computer program might learn to play chess by studying its opponents. However, it is useful to study learning applications that are not adversarial.) Thus it is reasonable to assume that the teacher is well-meaning.

The assumption that the teacher is always correct, however, is a less reasonable one. Teachers in machine-learning projects are often human experts or computer databases that ultimately rely on human knowledge. All of these sources are fallible. A human teacher may not know the answer to a question or may simply be wrong. Even if a question is answered correctly, there may be noise corrupting the response that the learner receives. Thus, for computational learning theory results to be applicable in any realistic setting, we must explore the problem of learning from fallible teachers.

The work in this chapter explores several new learning models in which the teacher may not know all the answers. Section 2.2 briefly describes work from my Masters Thesis on learning with a randomly-fallible teacher. This section also introduces the reader to many key ideas in Boolean concept learning. In Section 2.3, I describe recent work on learning with the help of a teacher who may make mistakes on borderline examples. In both cases, we validate the new learning models by describing algorithms that learn despite their teachers' shortcomings.

2.2 Randomly Fallible Teachers

2.2.1 Introduction

Consider the problem of teaching a computer to recognize verbs in English sentences. One approach for the teacher is to present sample sentences, pointing out some verbs as *positive examples* of the target concept and some other words as *negative examples*. From this, the learner might develop a general idea of the target concept. But in some sentences, other words may deceptively appear in verb form. A natural extension allows the learner to ask specific questions of the type, "is *this* word a verb?" In most cases, the teacher will know the answer from context. For example, in the sample sentence "The department stores open at nine," the learner might consider the possibility that "department" is the subject and "stores" is used as a verb, but would be corrected by the teacher. However, there may be instances in which even the teacher is unsure. The sentence "Tom is running back for his school football team" has at least two legitimate interpretations; the word "running" is a verb in one case but not in another. Without more information, the teacher cannot answer the learner's question, "is 'running' a verb?" Can the learner still learn, even when the teacher is sometimes unsure?

This section answers that question in the affirmative for the class of monotone DNF formulas. We introduce a new fault-tolerant model of algorithmic learning using an equivalence oracle and an *incomplete membership oracle*, in which the answers to some of the learner's membership queries may be unavailable. The advantage of this model is that it imitates the natural fallibility of teachers in most learning systems. Previous work on query models has generally assumed an omniscient teacher that answers all queries with perfect accuracy. Such assumptions are impractical; even well-intentioned teachers are seldom all-knowing. It is important to consider the degree of teacher fallibility that these models can tolerate. This section demonstrates that efficient learning is possible even when the teacher is unable to answer a constant fraction (less than one) of the questions asked. By defining a measurable tradeoff between membership and equivalence queries, our model yields some insight into the degree of additional information that membership queries provide a learning algorithm.

Previous Work

There has been a good deal of work done on errors in the distribution-free model of learning introduced by Valiant [108]. Results are encouraging for the case of random misclassification errors. In this benign error model, the teacher produces labeled positive or negative examples, where the label for any example is incorrect independently with probability η. Angluin and Laird [11] show that information-theoretically, as long as η is less than $\frac{1}{2}$, a sequence of labeled examples of length polynomial in $(\frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{1-2\eta})$ is sufficient for PAC-learning. In particular, they show that k-CNF formulas are PAC-learnable in polynomial time with a random noise rate of less than $\frac{1}{2}$.

Other work has focused on the case of malicious misclassification errors in examples. Valiant [109] poses the question of learning k-CNF formulas despite an adversarial teacher that draws random positive or negative examples, but with error probability β returns an arbitrary response instead of the correctly labeled example. Valiant shows that a small rate of error can be tolerated in this model. Kearns and Li [64] show that Valiant's error bound is tight; they use an information-theoretic argument to prove that a malicious error rate of at most $O(\epsilon)$ is tolerable when PAC-learning any distinct concept class $\mathcal{C}$. A number of other papers further explore various models in which the examples themselves or their classifications are corrupted (see Laird [70]; Shackelford and Volper [99]; Sloan [101, 102]; among others).

Less is known about errors in query models. Sakakibara [97] proposes a model of noise in queries, which assumes that every time a query is asked there is some independent probability of getting the wrong answer. Sakakibara gives a general technique to repeat a query sufficiently often to increase the confidence in the answer to a very high level, which allows existing algorithms to be used with appropriate modifications.

However, in some practical situations the problems of missing or incorrect information may not be so easy to remedy. For example, when we ask a teacher to classify a given element of the universe as a positive or negative example of the target concept, it may happen that the teacher simply does not know, and will not know no matter how many times we ask the same question. To address this problem, Goldman, Kearns, and Schapire [53] consider a model of persistent noise in membership queries that is related to the model we adopt here.

Overview of the Model

Our model relies on the definition of a *minimally adequate teacher* [5], in which a learner tries to learn a target concept f from a known concept class $\mathcal{C}$. In this (error-free) model, the learner is assisted by a teacher that answers two types of queries. A membership query on a given point tells whether or not that point is a positive example of f; such a query is answered "yes" or "no". An equivalence query tests a hypothesis h, returning $\emptyset$ if h is equivalent to f, and a counterexample x such that $h(x) \neq f(x)$ otherwise. In this model, the choice of the counterexample is arbitrary.

We consider a *randomly fallible minimally adequate teacher*. In particular, we define an *incomplete membership oracle* that, for each point in the instance space, performs one flip of a biased coin that lands "heads" with probability p. On any example in the instance space whose coin landed "heads", membership queries are always answered with "I don't know". On all other points in the instance space the membership oracle always answers correctly. In other words, with probability p, the teacher may be unsure about a given example and will never gain any more information about it. Note that this situation is "benign" in the sense that the algorithm has only to deal with missing information – the information it gets is guaranteed to be correct.

For equivalence queries we assume that the answers remain correct; that is, the answer "$\emptyset$" is returned if and only if the queried element h is equivalent to the target concept f and otherwise the answer is a counterexample x such that $h(x) \neq f(x)$. This assumption means that exact identification of the target concept is still possible (since an algorithm could simply perform identification by enumeration using equivalence queries.)

In this new model we must specify the type of adversary selecting the counterexamples. (This is also an issue in the standard model when randomized learning algorithms are considered, as Maass [77] has shown.) We assume that the adversary is "on-line." That is, the choice of a counterexample may depend on the target hypothesis and the history of the computation to the point at which the query is asked, including the hypothesis queried, all previous queries and their answers, and any previous coin-flips of the learning algorithm. However, the choice of counterexample may not depend on the answers to membership queries not yet made. This adversary is strong enough to generate the "worst-case" counterexamples used to provide lower bounds for equivalence queries [7], but it cannot predict the blind spots of the incomplete membership oracle.

Discussion of the Model

It may at first seem odd to assume that membership queries are flawed while equivalence queries remain correct. However, consider the situation in which a learning algorithm is attempting to predict the classification of a sequence of examples (produced and classified by Nature) with the assistance of a teacher who can correctly classify some but not all of the possible examples (modeled by an incomplete membership oracle.) If we have an efficient learning algorithm using equivalence queries and an incomplete membership oracle, then by a general transformation [74] we can obtain an efficient algorithm for the prediction task in the mistake bounded model that uses an incomplete membership oracle.

It may also seem unrealistic to consider a teacher whose failures occur uniformly at random. Despite the weight of precedent, it is natural to look for a more reasonable way of modeling a teacher's limitations. One possibility is for the teacher to be less certain about data points that are "close to the border" of the target concept. This model is particularly appealing for certain graphic concepts such as handwriting recognition or intersections of half-planes. We present preliminary results for such a model in

Section 2.3. However, reasoning about randomly flawed teachers has proven informative and somewhat more tractable.

The Importance of Membership Queries

Certain classes of concepts, such as deterministic finite state acceptors and monotone DNF formulas, have been shown to be learnable in polynomial time using equivalence and membership queries, but not using equivalence queries alone ([108, 5, 6, 7]). A natural question is to investigate which of these concept classes remain learnable in polynomial time in the appropriate sense in this new model, with an incomplete membership oracle. We may then derive some measure of the "importance" of membership queries to the learning algorithm as we vary the failure probability p from 0 (complete information from membership queries) to 1 (no information from membership queries.)

Angluin and Kharitonov [9] explore a number of cryptographic limitations on the power of membership queries. They show that, assuming the intractability of either testing quadratic residues modulo a composite, inverting RSA encryption, or factoring Blum integers, there is no polynomial time prediction algorithm using membership queries for several important concept classes, including the classes of Boolean formulas, 3μ-formulas, NFA's, and CFG's. They also show that if one-way functions exist, then membership queries provide no additional power in learning general CNF or DNF. That is, either these classes are learnable without membership queries, or they are not learnable even with membership queries. Their work is especially relevant in light of the results of this section, showing that membership queries do provide additional power to some learning algorithms, even when only a fraction of the queries are answered. It would be interesting to find a general characterization of the concept classes for which this is the case.

Monotone concept classes are particularly promising for this new model, since there is the hope of reconstructing missing information from responses to additional queries. In this section we examine the learnability of monotone DNF formulas over n variables.

There is a known algorithm for exactly learning these formulas from a minimally adequate teacher in time polynomial in n and m, where m is the number of terms in the target formula ([108],[6]). We present an algorithm for the new model that, for any failure probability $p < 1$, produces with probability at least $1 - e^{-s}$ a hypothesis equivalent to the target concept in time polynomial in n, m and s. The running time of this algorithm is not dominated by the failure probability for moderate p; when $p \leq \frac{1}{2}$, the expected total number of queries is $O(mn^2)$.

We observe that when p is nonzero, there is a nonzero probability that the algorithm will obtain *no* information from any of its membership queries. However, there is no algorithm that runs in time polynomial in n and m and exactly identifies any monotone DNF formula using equivalence queries only [7]. Thus, the quantification of "with high probability" is necessary in the statement of our main result.

2.2.2 Preliminaries

The target concepts are monotone formulas in disjunctive normal form (DNF) over the variables $x_1, \ldots, x_n$ for some positive integer n. For example, for $n = 20$,

$$x_1 x_4 + x_2 x_{17} x_3 + x_9 x_5 x_{12} x_3 + x_8$$

is a possible target concept. The number of terms in the target formula will be denoted by m; in this example $m = 4$. Note that there is an efficient algorithm to minimize the number of terms of a monotone DNF formula. We shall assume that the target formula f has been minimized.

The *instance space* of examples is the set of all possible vectors of n **0**'s and **1**'s; that is, the set $\{\mathbf{0}, \mathbf{1}\}^n$. A monotone DNF formula h is interpreted as denoting the set of vectors from the instance space that satisfy h. If vector v satisfies formula h we write $h(v) = 1$; otherwise $h(v) = 0$. We view the instance space as a lattice, with componentwise "or" and "and" as the lattice operations. The top element is the vector of all **1**'s, and the bottom element is the vector of all **0**'s. The elements are partially ordered by $\leq$, where $v \leq w$ if and only if $v[i] \leq w[i]$ for all $1 \leq i \leq n$.

For convenience, we introduce an alternative representation of monotone DNF formulas in which each term is represented by the minimum vector in the ordering $\leq$ that satisfies the term. Thus, the vector **10011** denotes the term $x_1x_4x_5$. In this representation, if h is a monotone DNF formula and v is a vector in the instance space, v satisfies h if and only if for some term w of h, $w \leq v$. Figure 2.1 shows the four-variable lattice, with the enclosed area containing all vectors satisfying the formula $x_1x_4 + x_1x_2 + x_3$.

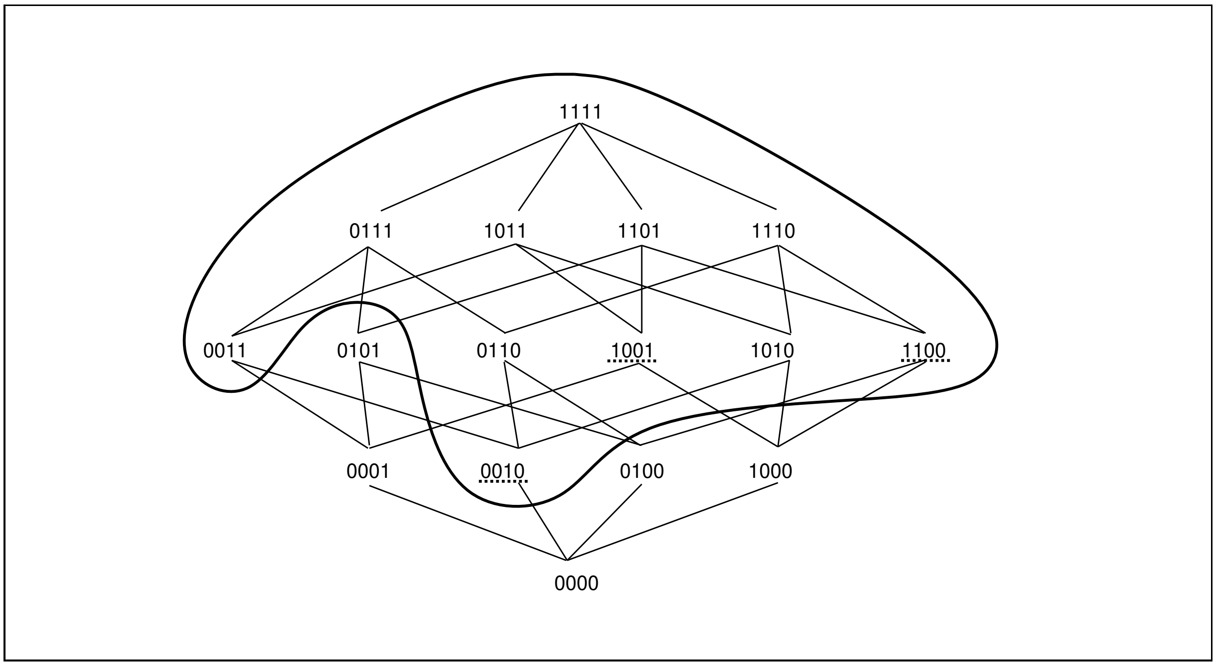


Figure 2.1: The target concept $x_1x_4 + x_1x_2 + x_3$.

In this representation, since we have assumed that the target concept f is minimized, the terms of f are precisely the minimal positive examples of f, also called *minterms*. Similarly, we define a *maxterm* t_m of the formula to be a maximal negative example of f. That is, $f(t_m) = 0$, but if any variable not in t_m is added, the resulting vector will force the target formula to 1.

The *descendants* of a vector v are all the vectors w in the instance space such that

$w \leq v$. For any nonnegative constant d, the *d-descendants* of v are all the descendants w of v that can be obtained from v by replacing at most d **1**'s by **0**'s. For example, consider the term $x_2x_3x_4$, represented as **0111**. Its set of 1-descendants, {**0111**, **0011**, **0101**, **0110**}, contains the term itself and its three children. The set of 2-descendants is the union of the set of 1-descendants with **0111**'s grandchildren: **0001**, **0010**, and **0100**. Figure 2.2 shows the set of 2-descendants of the vector **0111**.

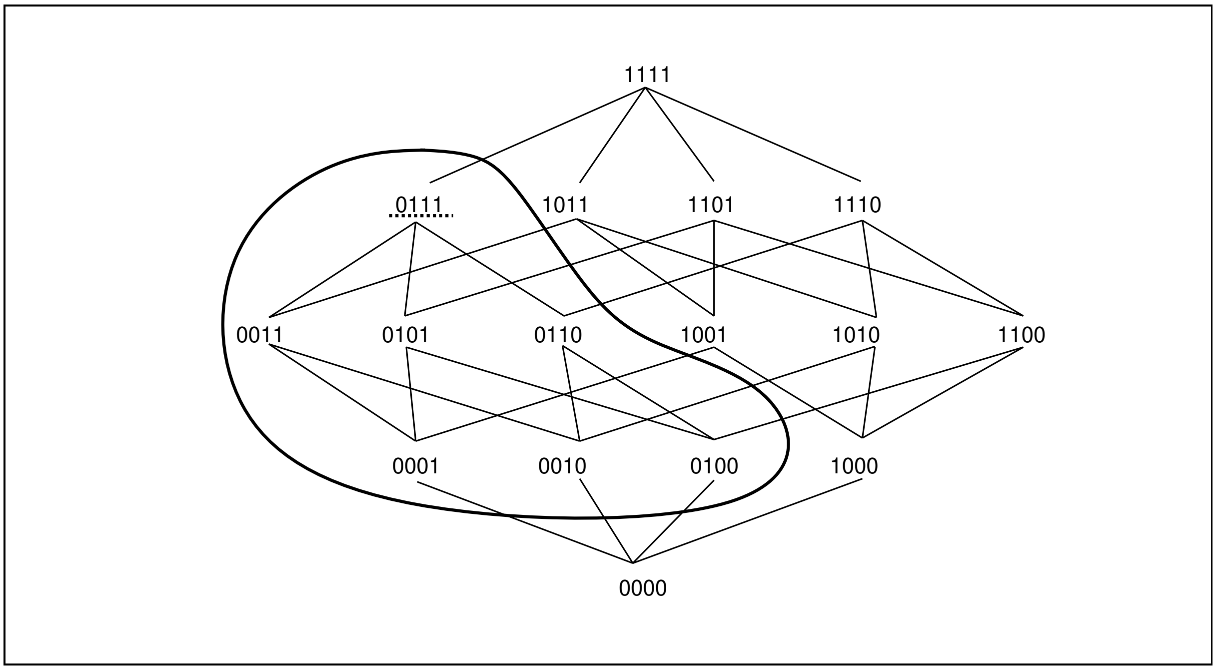


Figure 2.2: The 2-descendants of $x_2x_3x_4$.

2.2.3 Using Incomplete Membership Queries

A key subprocedure in the monotone DNF algorithm of Angluin [6] takes a positive example v of the target concept f and uses membership queries to reduce v to a minimum positive example of f. The algorithm starts with the empty formula and uses equivalence queries to generate new positive counterexamples to reduce. The result of each reduction is a new term of f, which is added to the current hypothesis. After m iterations of this process the current hypothesis is equivalent to f.

Our new algorithm is based on the same idea, but must use an incomplete membership oracle. The difficulty that arises is as follows. The reduction process has a current positive example v of f and makes membership queries for each of the children of v. As long as at least one of these membership queries is answered "yes," say for the vector y, then v can be replaced by y and the process repeated. However, eventually the process arrives at a positive example v of f such that membership queries for all of the children of v are answered either "no" or "I don't know." If there is at least one child of v answered "I don't know," then v may or may not be a minimum positive example of f.

We therefore modify the reduction process so it may be used with an incomplete membership oracle. The goal is to take an initial positive example v of f and reduce it to a positive example that is "likely" to be "not too far above" a minimum positive example of f. By adding all sufficiently close descendants of this vector as terms to the current hypothesis, we will be "likely" to add a new term of f, and therefore, to make progress towards exact identification of f. In so doing, we may add terms to the current hypothesis that do not imply f, but these will eventually be removed in response to negative counterexamples.

The idea of the new reduction process is to use membership queries to search not only the children of v but also all the "close enough" descendants of v, looking for a vector $y \leq v$ that is answered "yes." If such a y is found, then the search continues with y in place of v. If no such y is found after querying all the "close enough" descendants of v, then v is returned. Note that the descendants are searched in breadth-first order – first the children of v, then the grandchildren, etc. The parameter $d \geq 1$ specifies the depth of search from v.

Suppose **Reduce** is called with an incomplete membership oracle for f, and inputs v (a positive example of f) and $d \geq 1$. It is clear that **Reduce** must eventually return a vector v' such that $v' \leq v$, v' is a positive example of f, and membership queries for all the proper d-descendants of v' were answered either "no" or "I don't know." In Angluin and Slonim [13], we determine the values of d for which the probability that there is NO

Reduce(v, d):
$D := \{d | d$ is a proper d-descendant of v $\}$
for each $y \in D$ in breadth-first order
  do if *membership-query*(y) = "yes"
    then return Reduce(y, d)
return v

minimum positive example of f among the d-descendants of v' is at most 1/2. We prove that this property holds for any

$$d \geq \log(2 + \frac{1 + \log(1/(1-p))}{\log(1/p)}) - 1.$$

In particular, for $p \leq 1/2, d = 1$ is sufficient; as $p = (1 - \alpha)$ approaches 1, it suffices to choose

$$d \geq \lceil \log(1/\alpha) + \log\log(1/\alpha) \rceil.$$

2.2.4 Learning Monotone DNF Formulas

Now we are in a position to describe the learning algorithm. The algorithm begins with the empty hypothesis (false on all vectors) and makes equivalence queries and processes counterexamples one at a time until the hypothesis is equivalent to the target function. Because it tries to "guess" some of the terms in the target concept, the algorithm may introduce terms that do not actually imply the target concept. Thus a counterexample v may be a negative example of f, in which case $f(v) = 0$ and $h(v) = 1$. The algorithm processes such a v by removing all the terms from the current hypothesis that are satisfied by the vector v.

Otherwise, a counterexample v is a positive example of f. Intuitively, what we'd like to do is to call **Reduce** on v to obtain a vector y that with probability at least 1/2 has a minimum positive example of f among its d-descendants for some moderate value of d. Then we would add all the d-descendants of y as terms to the current hypothesis h

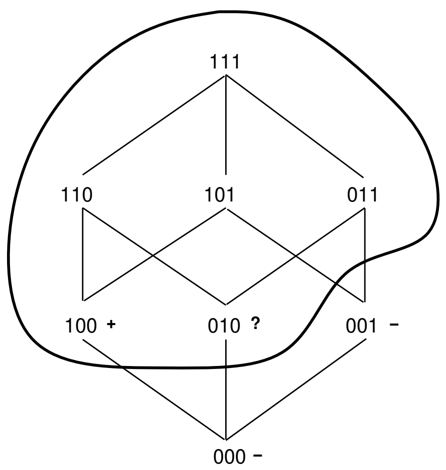


Figure 2.3: The $+, -$, and ? symbols respectively indicate positive, negative, and "I don't know" answers to membership queries. The first positive counterexample is **110**; vectors **010**, **100**, and **000** are queried. If the second counterexample is **011**, vector **010** would be queried again.

and continue. If this worked for each positive example, we'd expect to add a new term of f to h for every two positive counterexamples, which would be sufficient progress.

The difficulty is that in order to apply our probabilistic bounds, we must guarantee that at each call to **Reduce** with argument v, all the descendants of v that are positive examples of f have not been previously queried. This is certainly true the first time **Reduce** is called, but may not be true on subsequent calls. For example, consider the target concept $f = x_1 + x_2$ over three variables, as shown in Figure 2.3, and assume that the initial (positive) counterexample is **110**. When called with (**110**, 1) as its arguments, **Reduce** performs membership queries for the children **010** and **100**. Suppose the first query is answered with "I don't know" and the second with "yes." Then the process is iterated with **100**, and the child **000** is queried. Suppose the answer in this case is "no." The hypothesis is then set to x_1 and an equivalence query is made. Assume now the counterexample is **011**. Then **Reduce** is called with arguments (**011**,1) and makes membership queries for the children **001** and **010**. However, **010** is a descendant of the argument that is a positive example f, and it has already been queried and answered with "I don't know." Thus we can no longer claim that the probability a membership query on **010** returns "I don't know" is exactly p.

Our solution for this difficulty is to add to the current hypothesis as terms ALL the vectors queried by **Reduce** that result in an answer of either "yes" or "I don't know." This guarantees that when a positive counterexample v to the current hypothesis is generated, none of the descendants of v that are positive examples of f has been previously queried. (Otherwise they would still be present as terms in the current hypothesis, contradicting the fact that v is a counterexample to the current hypothesis.) In fact, a vector u ($u \neq y$) queried by **Reduce** that receives the answer "yes" must be a direct ancestor of the returned vector y, which is the *last* vector whose query by **Reduce** answers "yes". A positive counterexample to a hypothesis containing y would be a counterexample to any such vector u as well. So it is sufficient to add to the hypothesis just the vector y and all those vectors queried by **Reduce** that result in "I don't know" answers.

Thus we assume that **Reduce** has been modified to return two results: the vector y previously returned, and the set Q of all the vectors queried by the top-level and all the recursive calls to **Reduce** that resulted in answers of "I don't know." Note that Q must include all the proper d-descendants of y that are positive examples of f, since they must all have been queried when **Reduce** returns, and they must all have been answered "I don't know." The modified version of **Reduce** will be called **mod-Reduce**, and will have an additional parameter Q, which should initially be the empty set.

We can now specify **Learn-mDNF**, our learning algorithm for monotone DNF formulas that uses an equivalence oracle and an incomplete membership oracle. The choice of constant d depends on p as described above. Recall that we denote a term by the minimum vector satisfying it, so that the current hypothesis h may be thought of as a set of vectors.

```
mod-Reduce(v, d, Q):
D := {d|d is a proper d-descendant of v }
for each y ∈ D in breadth-first order
    do if membership-query(y) = "I don't know"
        then Q = Q ∪ {y}
    do if membership-query(y) = "yes"
        then return mod-Reduce(y, d, Q)
return v, Q
```

```
Learn-mDNF:
h := the empty formula
while (v = equivalence-query(h)) ≠ ∅
    do if (h(v) = 1)
        then remove from h all w | w ≤ v
    else y, Q := mod-Reduce(v, d, ∅)
        h := h ∪ {y} ∪ Q
Output h and halt
```

The analysis in Angluin and Slonim [13] shows that this algorithm exactly identifies f and with high probability runs in time polynomial in n, m, and d. For $p \leq 1/2$, the algorithm requires an expected $O(mn^2)$ queries.

2.2.5 Handling Some Errors

One important question is what happens when membership queries may be answered incorrectly. In the case of missing information, at least the learner can rely on the correctness of the information that is obtained. With possibly incorrect answers, the learning problem seems to become harder. One natural extension of the current model is to permit the membership oracle to give one of a variety of "corrupted" answers for those strings whose coin flip results in "heads," while requiring correct answers on the remaining strings.

For one variant, namely $1 \rightarrow 0$ one-sided errors, a minor modification to **Learn-mDNF** permits it to cope with such errors. In this model, for each string in the instance space whose coin flip results in "heads," the answer of the membership oracle is always "no." The modification to **Learn-mDNF** is to add a term to h for EVERY vector queried with membership queries, since now an answer of "no" may be given for a positive example. The analysis of positive examples answered "no" is then the same as the previous analysis of positive examples answered "I don't know." The key observation is that queries answered "yes" are answered correctly; in fact, this modification copes with a model in which corrupted examples may be answered either "no" or "I don't know" arbitrarily.

The dual problem, that of $0 \to 1$ one-sided errors, is more difficult. A trivial modification of **Learn-mDNF** is no longer sufficient, because the reduction procedure will not terminate until queries on all descendants of some vector return "no". One might assume that if $1 \to 0$ errors are easily handled by an algorithm that finds minterms of the target formula, then a similar bottom-up approach could be applied to handle $0 \to 1$ errors and find all maxterms of the target formula. However, for a monotone DNF formula with m minterms over n variables, there may be up to n^m maxterms! Thus, this approach is impractical as well.

2.2.6 Future Directions

There are a number of additional questions in this area that could be investigated. Kharitonov suggests the problem of combining our model with errors in equivalence queries. Persistent errors in deterministic equivalence queries are too strong an adversary; clearly, if an equivalence oracle ever claims the hypothesis is correct when it is not, the algorithm will fail. It is more interesting to ask what happens when the equivalence oracle is approximated by a sufficiently large number of labeled random examples, drawn according to some natural probability distribution. What happens if there is random misclassification noise in the sampling used by the equivalence oracle, in addition to "I don't know" or erroneous answers to membership queries?

It would be interesting to determine if anything can be done in the case of a *malicious* incomplete membership oracle, where an adversary is given some control over which membership queries the teacher cannot answer. In one such model, given failure bound p, the adversary is allowed to specify up to $p \times 2^n$ vectors for which the teacher cannot answer membership queries. Certainly, if p is a constant fraction, any algorithm can be forced to make an exponential number of queries to learn monotone DNF. For example, let $p = 1/4$; the adversary refuses to answer all queries on vectors whose first two bits are set to 1. Then the problem of learning any target concept where every term contains both x_1 and x_2 is effectively reduced to learning with just equivalence queries, which is known to require exponential time. Exactly how much power does an adversary need to prevent learning in a malicious model?

Another question is whether we can find polynomial time algorithms in this model for other learning problems known to have polynomial time algorithms using equivalence and membership oracles. Goldman and Mathias [54] show how to exactly identify k-term DNF formulas in this model, proving that the model can be successfully applied to non-monotone concept classes. It would be interesting to determine what other types of concept classes (geometric concepts?) can be learned in this model.

A final important goal is to explore methods of coping with persistent errors in membership queries beyond the $1 \rightarrow 0$ one-sided errors considered above. Since PAC-learning can tolerate a large degree of random misclassification errors, general (two-sided) random errors in membership queries might seem approachable. Indeed, the results of Goldman, Kearns, and Schapire [53] show that the classes of logarithmic-depth read-once majority formulas and logarithmic-depth positive NAND formulas can be learned with high probability using only membership queries, even if the membership queries are subject to persistent two-sided errors.

The analysis of our algorithm relies on the assumption that all the "I don't know"s are distributed independently at random. It would be informative to find useful results for a more natural model, as discussed in Section 2.2.1. Preliminary but promising results in such a model are described in the next section.

2.3 Unreliable Boundary Queries

2.3.1 Introduction

In most of the theoretical work on concept learning, the environment is modeled as an omniscient oracle that classifies all objects as positive or negative instances of the concept to be learned. Thus, it is assumed that there is a well-defined boundary separating positive from negative examples. In many cases, however, classification may be much less clear. For example, consider a membership query algorithm for learning to recognize the number 3 from pixel images. A typical strategy would involve taking a 3 and a non-3 (maybe a picture of a 2) and asking for classifications of examples halfway between them until two nearby examples with different classification are found. A problem with this type of approach[1], as noticed by Lang and Baum [72], is that questions of this sort that are near the concept boundary may result in unreliable answers. Merging an image of a 2 and a 3 tends to produce something that looks a bit like both, and that we don't really care about anyway since we don't expect to see one in practice.

More generally, one unrealistic aspect of the PAC-with-membership-query model is that it relies much more heavily on its assumptions than the passive PAC model. In both models, one typically assumes there is a target function belonging to some class $\mathcal{C}$ that is labeling the data, and one then tries to prove that one's algorithm will succeed under that assumption. In the passive model, however, all that is really needed is that the target function and the distribution on examples conspire in such a way that the data *actually seen* is consistent with a function in $\mathcal{C}$. In contrast, with membership queries one needs the function on the entire space to be consistent with some function in $\mathcal{C}$. For instance, suppose a learning algorithm is using a simple hypothesis class (say a simple neural network) to learn images of 3's. For a passive algorithm, one would want the data observed to be consistent with some hypothesis in the class. For a membership query algorithm, however, one needs the stronger condition that the target concept *over*

[1] Particularly when a human "expert" serves as the membership query oracle.

the entire input space can actually be represented in such a simple form. The difference is that typical images of 3's may be distinct enough from images of other characters that many simple consistent hypotheses exist. However, if one were to probe the exact boundary of the "3" concept, one would likely find it has a complicated structure that even depends on which "expert" you ask.

In this section we propose and study a model for learning with membership queries that addresses the above issues. The basic idea of our model is that queries near the boundary of a target class may receive either incorrect or "don't care" responses. But, in partial compensation, we assume the distribution of examples has zero probability mass on the boundary region. (The motivation is that the oracle responds incorrectly or doesn't know the answers because these examples do not actually appear in the world, and thus it does not matter how the learner classifies them.) We do not *require* the oracle to answer incorrectly or state "I don't care" in the boundary region, since that would just make the learning problem that of learning a different (perhaps ternary) target concept in the standard model. In that case, one could then simply perform binary search between the boundary and non-boundary examples, defeating the purpose of the model. One way of viewing our model (although our model is actually a bit more general) is that the true target concept is in fact some horribly complicated function, but differs from a simple function only in a boundary region that has zero probability measure. See Figure 2.4 for an example.

The contributions of this work are: (1) the introduction of the model of learning with unreliable boundary queries, (2) an efficient algorithm that PAC-learns the intersection of two halfspaces with membership queries when the boundary queries are noisy, and (3) efficient algorithms to exactly learn (with membership queries) several subclasses of monotone DNF formulas when there are one-sided false positive errors in the boundary queries for a small boundary size.

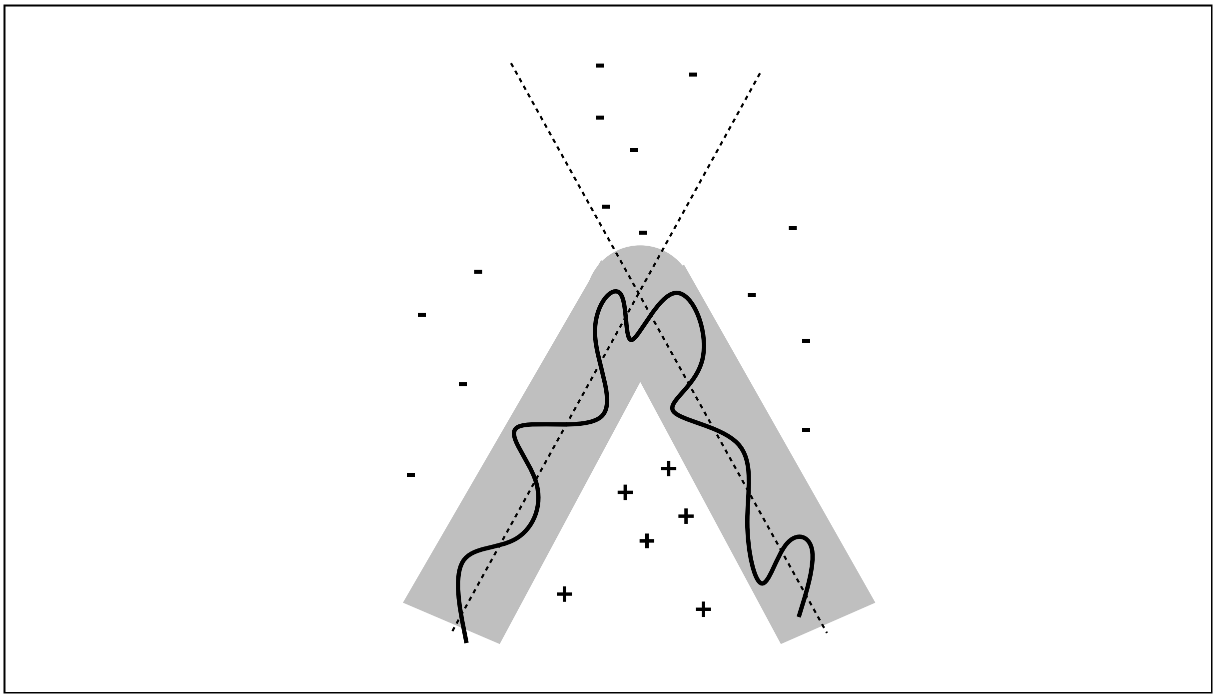


Figure 2.4: The thick curve is the actual concept boundary. However, because the distribution has zero probability mass in the shaded region, we can view the concept as an intersection of two halfspaces in our model.

2.3.2 Definitions

Given a concept f over an instance space $\mathcal{X}$ that has a distance metric, we say that the *distance to the boundary* of an example x is the distance to the nearest example y such that $f(x) \neq f(y)$. For continuous input spaces we use the infimum over distances to y's such that $f(x) \neq f(y)$. In the boolean domain we use the Hamming distance as our metric. Thus an example is at distance 2 from the boundary if it is possible to flip two bit positions and change its classification. In continuous domains, the L_2 metric is often most natural. We define the *boundary region of radius* r to be the set of examples whose distance to the boundary is at most r. We define the *negative boundary region of radius* r to be the set of all examples x in the boundary region such that $f(x) = 0$.

We now define the *unreliable boundary query* (UBQ) model. This model is the same as the standard PAC-memb model except for the following difference: there is a value r (the boundary radius) such that any query to an example in a boundary region of radius r *may* receive an incorrect response, and the example distribution D has zero probability measure in that boundary region. In the *incomplete boundary query* (IBQ) model, the learner never receives an incorrect response to a query, but in the boundary region might receive the answer "don't care". We also consider a one-sided false-positive-only UBQ model in which the learner may receive false positive answers to any queries in the negative boundary region, but receives correct answers in the positive boundary region. In this case, the distribution D is only required to have zero probability on the negative boundary region. (Note that an algorithm for this model can learn in the IBQ setting by simply treating each "don't care" response as positive.) Finally, we extend these definitions to the exact learning model by requiring that counterexamples to equivalence queries not be chosen from the boundary region.

2.3.3 Related Work

There has been a great deal of theoretical work on PAC or mistake-bound learning in cases where the training examples may be mislabeled [6, 70, 101, 64] and additional work in models that allow attribute noise [99, 52, 75]. The p-concepts model of Kearns and Schapire [65] also falls somewhat into this category.

There have also been a number of results on learning with randomly generated noisy responses to membership queries. Sakakibara [97] considers the case where each membership query is incorrectly answered with a fixed probability, but where one can increase reliability by asking the *same* membership query several times. In models of *persistent* membership query noise, repeated queries to the same example receive the same answer as in the first call. Goldman, Kearns and Schapire [53] give a positive result for learning certain classes of read-once formulas under this noise model. Their work uses membership queries to simulate a particular distribution. Frazier and Pitt [49] show that CLASSIC sentences are learnable in this noise model, using the fact that many distinct membership queries can be formulated that redundantly yield the same information.

Angluin and Slonim [13] introduce a model of incomplete membership queries (described in Section 2.2), in which a membership query on a given instance may persistently generate a "don't know" response. The "don't know" instances are chosen uniformly at random from the entire domain and may account for up to a constant fraction of the instances. Additional positive results in this model are obtained by Goldman and Mathias [54]. This model allows for a large number of "don't know" instances, but positive results in this model are typically highly dependent on the precisely uniform nature of the noise.

Sloan and Turan [103] introduce the *limited membership query model.* In this model, an adversary may arbitrarily select some number ℓ of examples on which it refuses to answer membership queries (or answers "don't know"), but the learner is now allowed to ask a number of queries polynomial in ℓ. Sloan and Turan present algorithms in this model for learning the class of monotone k-term DNF formulas with membership queries alone and the class of monotone DNF formulas with membership and equivalence queries. Angluin and Kriķis [10] introduce a similar model of *malicious membership queries* in which the adversary may respond with *incorrect* answers instead of "don't know". Their paper proves that the class of monotone DNF formulas is learnable in this model. Angluin [8] has shown that read-once DNF formulas are also learnable with malicious membership queries.

The main difference in motivation between our model and those above is that most previous work supposes that there is a clear boundary between the positive and negative examples with some noise included. Our goal is to model the very different situation in which the classification of examples in the boundary region is just not well defined (for example, a "2" merged with a "3"). Our model is more difficult than those above in the sense that the membership query errors or omissions are chosen by an *adversary* (unlike the random noise models [13]), and algorithms must run in time that is polynomial in the usual parameters *regardless* of the number of queries that might receive incorrect answers (unlike [103, 10]). For example, in the case of a 1-term monotone DNF formula with the boundary radius $r = 1$, there may be exponentially many (in n) instances in the boundary region. (Example: let $x_4x_7x_9$ be the target term. Then all positive instances, and all negative instances with exactly one of $\{x_4, x_7, x_9\}$ turned off, are in the boundary region of radius 1.) On the other hand, to partially compensate for this difficulty, we restrict membership query errors or omissions to the boundary region and we require that counterexamples to equivalence queries be chosen from outside the boundary region.

In other related work, Frazier, Goldman, Mishra and Pitt [48] introduce a learning model in which there is incomplete information about the target function due to an ill-defined boundary. While the omissions in their model may be adversarially placed, all examples labeled with "?" (indicating unknown classification) must be consistent with knowledge about the concept class. They require the learner to construct a *ternary* function with values {0,1,?} that, with high probability, correctly classifies most randomly drawn instances, and give positive results for the classes of monotone DNF formulas and d-dimensional boxes. One of the key differences between their model and ours is that they allow time polynomial in the complexity of that ternary function: thus if the "?" region has a complicated shape, then their learner is allowed a correspondingly longer time. In our model, a learner is required to learn quickly regardless of the complexity of the "?" region.

2.3.4 Learning an Intersection of Two Halfspaces

We now describe one of our main positive results: an algorithm for learning an intersection of two halfspaces in n dimensions in the UBQ model, for any boundary radius r (see Figure 2.5). Our algorithm is an extension of an algorithm of Baum [16, 17] for learning the simpler class of intersections of two *homogeneous* halfspaces in the standard PAC-with-queries model[2].

The idea of Baum's algorithm is to reduce the problem of learning an intersection of two homogeneous halfspaces to the problem of learning an XOR of halfspaces, for which a PAC algorithm exists [24]. (That algorithm produces a hypothesis that is

[2] A halfspace is homogeneous if its bounding hyperplane passes through the origin.

the threshold of a degree-2 polynomial.) The idea of the reduction is to notice that negative examples in the quadrant opposite from the positive quadrant—the troublesome examples keeping the data set from being consistent with an XOR of halfspaces—are exactly those examples $\vec{x}$ such that $-\vec{x}$ is positive. His algorithm is as follows:

> Draw a sufficiently large set S of examples.[3] Mark all of the negative examples $\vec{x} \in S$ which have the property that a membership query to $-\vec{x}$ returns "positive". Then find a linear function P such that $P(\vec{x}) < 0$ for all the marked (negative) examples and $P(\vec{x}) \geq 0$ for all the positives. Finally, run the XOR-of-halfspaces learning algorithm of [24] to find a hypothesis H' that correctly classifies $\{\vec{x} \in S : P(\vec{x}) \geq 0\}$. The final hypothesis is:
>
> "If $P(\vec{x}) < 0$ then predict negative, else predict $H'(\vec{x})$."

Baum's algorithm seems appropriate for our model because it does not explicitly try to query examples near the boundary. In fact, it is almost the case that if a negative example has distance at least r from the boundary, then the example $-\vec{x}$ has distance at least r from the boundary as well. This fails only on the negative examples in the "A-shaped" region shown in Figure 2.5.

Our algorithm for learning an intersection of (not necessarily homogeneous) halfspaces in the UBQ model is a small extension of Baum's algorithm, though the analysis requires a bit more care. In our algorithm, instead of reflecting through the origin, we reflect through a positive example. We use a potential function to prove that some "good" positive example for reflection must exist. (The algorithm tries all of them.) Specifically, our algorithm is the following:

> Draw a sufficiently large set S of examples.
>
> For each positive example $\vec{x}_{pos} \in S$ do the following. For each negative example $\vec{x}_{neg} \in S$, query the example $2\vec{x}_{pos} - \vec{x}_{neg}$, and if the response to

[3] The VC-dimension [110] of the hypothesis class is $O(n^2)$. Blumer, Ehrenfeucht, Haussler and Warmuth [28] have shown that a sample of size polynomial in the VC-dimension of the hypothesis class is sufficient for PAC-learning, so the number of examples needed is polynomial in n.

that query is "positive", then mark $\vec{x}_{neg}$. Now, attempt to find a linear function P such that $P(\vec{x}) < 0$ for all the marked (negative) examples and $P(\vec{x}) \geq 0$ for all the positives. If no such function exists, then repeat this step using a different positive example $\vec{x}_{pos} \in S$ (we prove below that this step must succeed for *some* positive example $\vec{x}_{pos}$).

Finally (assume we have found a legal linear function P), let S' be the set of $\vec{x} \in S$ such that $P(\vec{x}) \geq 0$, and use the XOR-of-halfspaces learning algorithm to find a hypothesis H' that correctly classifies the examples in S'. The final hypothesis is:

"If $P(\vec{x}) < 0$ then predict negative, else predict $H'(\vec{x})$."

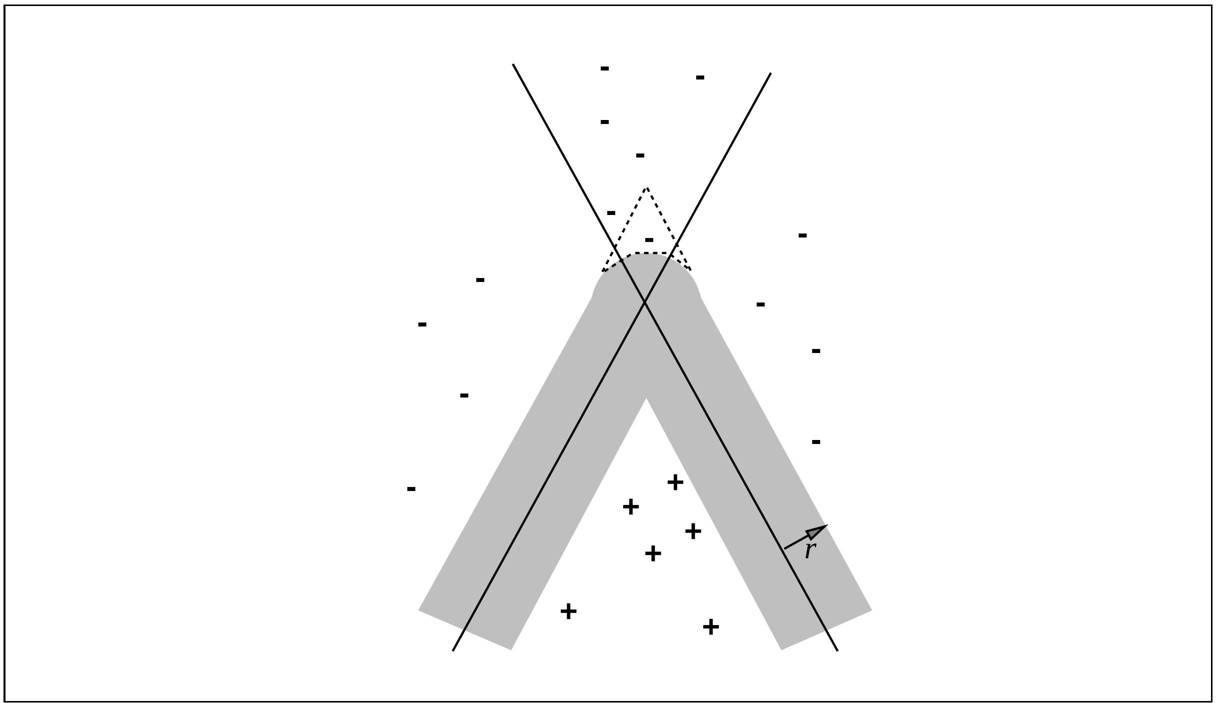


Figure 2.5: An intersection of 2 halfspaces. The boundary region is shaded. Notice that its apex is curved, which complicates the proof somewhat.

Theorem 1 *For any radius r of the boundary region, our algorithm succeeds in the UBQ model.*

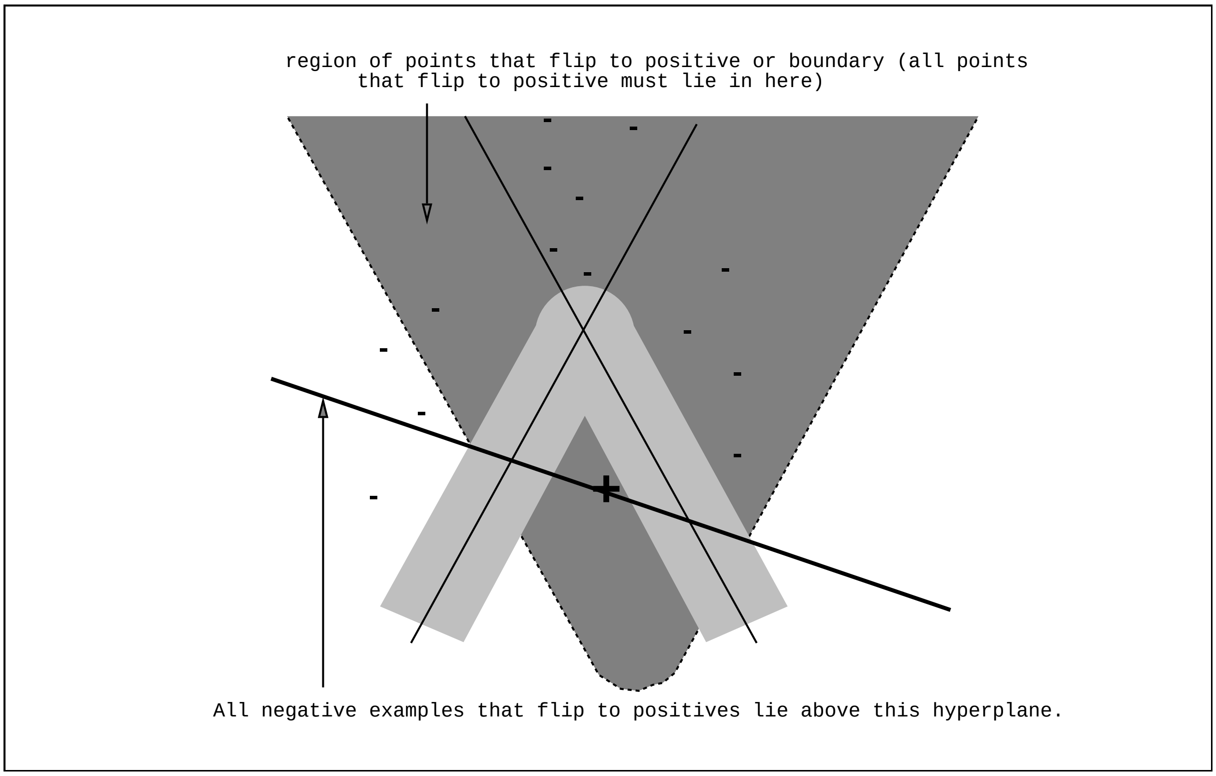


Figure 2.6: For clarity, $\vec{x}_{pos}$ is the only positive example shown. All marked negative examples lie within the dark-shaded region, which is the reflection of the positive and boundary regions through $\vec{x}_{pos}$. Lemma 2 states that the intersection of this region with the non-boundary negative region is linearly separable from the set of positive examples in S. The hyperplane pictured is the linear equality $P(x) = 2$ from that lemma.

Before giving a proof of correctness, we point out the simplifying observation that our algorithm is invariant under translation. If we add some vector $\vec{v}$ to each $\vec{x} \in S$, this results in adding $\vec{v}$ to each point of the form $2\vec{x}_{pos} - \vec{x}_{neg}$ as well. In particular, this means that if we can prove that our algorithm succeeds when the hyperplanes are homogeneous, then this implies that our algorithm also succeeds in the general (non-homogeneous) case. Therefore, we can assume in our proof for simplicity that the hyperplanes are, in fact, homogeneous.

We now fix some notation. Let r be the radius of the boundary region (which, notice, is not used by the algorithm). We define the distance between a point x and a set S as the infimum over all $y \in S$ of $d(x, y)$. The target concept is defined by two *unit* vectors

$\vec{p}_1$ and $\vec{p}_2$, and the positive region $POS = \{\vec{x} : \vec{p}_1 \cdot \vec{x} \geq 0 \text{ and } \vec{p}_2 \cdot \vec{x} \geq 0\}$. We define the "opposite quadrant" to be $\{\vec{x} : \vec{p}_1 \cdot \vec{x} < 0 \text{ and } \vec{p}_2 \cdot \vec{x} < 0\}$. We say a point (or example) is "non-boundary" if it is not within the boundary region.

The negative non-boundary region NEG_{nb} is the set of negative points not in the boundary region. Formally,

$$NEG_{nb} = \{\vec{x} : (\vec{p}_1 \cdot \vec{x} < 0 \text{ or } \vec{p}_2 \cdot \vec{x} < 0) \text{ and } d(\vec{x}, POS) > r\}.$$

Notice that if either $\vec{p}_1 \cdot \vec{x} < -r$ or $\vec{p}_2 \cdot \vec{x} < -r$ then $\vec{x}$ is in NEG_{nb}, though these are not necessary conditions (see Figure 2.5). In fact, let us define

$$NEG_{far} = \{\vec{x} : \vec{p}_1 \cdot \vec{x} < -r \text{ or } \vec{p}_2 \cdot \vec{x} < -r\},$$

so $NEG_{far} \subseteq NEG_{nb}$. To get necessary conditions for lying in the region NEG_{nb}, notice that if

$$-r \leq \vec{p}_1 \cdot \vec{x} < 0 \text{ and } \vec{x} \in NEG_{nb}$$

then it must be the case that $(\vec{x} + r\vec{p}_1) \cdot \vec{p}_2 < 0$ (otherwise the point $\vec{y} = \vec{x} + r\vec{p}_1$ would be in the positive region implying that x is in the boundary). Similarly, if

$$-r \leq \vec{p}_2 \cdot \vec{x} < 0 \text{ and } \vec{x} \in NEG_{nb}$$

then $(\vec{x} + r\vec{p}_2) \cdot \vec{p}_1 < 0$. Thus,

$$NEG_{nb} \subseteq NEG_{far} \cup \{\vec{x} : (\vec{p}_1 \cdot \vec{x} < 0 \text{ and } \vec{p}_2 \cdot \vec{x} < -r\vec{p}_1 \cdot \vec{p}_2) \text{ or } (\vec{p}_2 \cdot \vec{x} < 0 \text{ and } \vec{p}_1 \cdot \vec{x} < -r\vec{p}_1 \cdot \vec{p}_2)\}. \tag{2.1}$$

We begin by showing that the negative examples in the opposite quadrant do in fact get marked by our algorithm.

Lemma 1 *For any non-boundary positive example $\vec{x}_{pos}$ and any negative example $\vec{x}_{neg}$ in the opposite quadrant, the point $2\vec{x}_{pos} - \vec{x}_{neg}$ is a non-boundary positive example.*

Proof: Since $\vec{x}_{neg}$ lies in the opposite quadrant, we have $\vec{p}_1 \cdot \vec{x}_{neg} < 0$ and $\vec{p}_2 \cdot \vec{x}_{neg} < 0$. Since $\vec{x}_{pos}$ is a non-boundary positive example, we know that $\vec{p}_1 \cdot \vec{x}_{pos} > r$ and $\vec{p}_2 \cdot \vec{x}_{pos} > r$. Therefore,

$$\vec{p}_1 \cdot (2\vec{x}_{pos} - \vec{x}_{neg}) \geq \vec{p}_1 \cdot 2\vec{x}_{pos} > r$$

and

$$\vec{p}_2 \cdot (2\vec{x}_{pos} - \vec{x}_{neg}) \geq \vec{p}_2 \cdot 2\vec{x}_{pos} > r.$$

□

What remains to be shown is that there exists a positive example $\vec{x}_{pos}$ such that the set of negative examples marked when using $\vec{x}_{pos}$ for reflection is linearly separable from the positives. In particular, we show the positive example $\vec{x} \in S$ that minimizes $(\vec{p}_1 \cdot \vec{x} + r)(\vec{p}_2 \cdot \vec{x} + r)$ will succeed. Letting $\vec{x}_{pos}$ be that example and $a_1 = \vec{p}_1 \cdot \vec{x}_{pos}$ and $a_2 = \vec{p}_2 \cdot \vec{x}_{pos}$, we show that a legal separator is the linear inequality $\frac{\vec{p}_1 \cdot \vec{x} + r}{a_1 + r} + \frac{\vec{p}_2 \cdot \vec{x} + r}{a_2 + r} \geq 2$. (Intuitively, what we want is for $\vec{x}_{pos}$ to be the "closest" positive example to the origin according to some measure, and the correct notion of "closest" is that of being on the hyperbola $(\vec{p}_1 \cdot \vec{x} + r)(\vec{p}_2 \cdot \vec{x} + r) = c$ for minimum c.)

Lemma 2 *Let $\vec{x}_{pos}$ be the example $\vec{x} \in S$ minimizing $(\vec{p}_1 \cdot \vec{x} + r)(\vec{p}_2 \cdot \vec{x} + r)$ and let $a_1 = \vec{p}_1 \cdot \vec{x}_{pos}$ and $a_2 = \vec{p}_2 \cdot \vec{x}_{pos}$. Then the linear function*

$$P(\vec{x}) = \frac{\vec{p}_1 \cdot \vec{x} + r}{a_1 + r} + \frac{\vec{p}_2 \cdot \vec{x} + r}{a_2 + r}$$

is at least 2 for each positive example $\vec{x}$ and at most 2 for each negative example $\vec{x}$ that is marked when using $\vec{x}_{pos}$ for reflection.

Proof: First we consider the positive examples. Let $\vec{x}'$ be some positive example in S. Define α and β so that $(\vec{x}' \cdot \vec{p}_1 + r) = \alpha(a_1 + r)$ and $(\vec{x}' \cdot \vec{p}_2 + r) = \beta(a_2 + r)$. By definition of $\vec{x}_{pos}$ we have $\alpha\beta \geq 1$, and by definition of the positive region we know both α and β are at least 0. These inequalities imply that $\alpha + \beta \geq 2$, which implies $P(\vec{x}') \geq 2$.

Now consider the negative examples. The set of examples $\vec{x}$ with the property that

$2\vec{x}_{pos} - \vec{x}$ might be classified as positive by a membership query is pictured in Figure 2.6. Any such example must belong to the set

$$\text{MAYFLIP} = \{\vec{x} : \vec{p}_1 \cdot \vec{x} \leq 2a_1 + r \text{ and } \vec{p}_2 \cdot \vec{x} \leq 2a_2 + r\}.$$

We now consider the possible cases for marked negative examples $\vec{x} \in S$, using the characterization of the negative non-boundary region given by Equation (2.1) (cases 1 and 2 below handle the possibility that $\vec{x} \in NEG_{far}$).

Case 1. Suppose $\vec{x} \in \text{MAYFLIP} \cap \{\vec{x} : \vec{p}_1 \cdot \vec{x} < -r\}$. Then $P(\vec{x}) < 0 + \frac{2a_2+2r}{a_2+r} = 2$.

Case 2. Suppose $\vec{x} \in \text{MAYFLIP} \cap \{\vec{x} : \vec{p}_2 \cdot \vec{x} < -r\}$. Then $P(\vec{x}) < \frac{2a_1+2r}{a_1+r} + 0 = 2$.

Case 3. Suppose $\vec{x} \in \{\vec{x} : \vec{p}_1 \cdot \vec{x} < 0 \text{ and } \vec{p}_2 \cdot \vec{x} < -r\vec{p}_1 \cdot \vec{p}_2\}$.

Then $P(\vec{x}) < \frac{r}{a_1+r} + \frac{r(-\vec{p}_1 \cdot \vec{p}_2+1)}{a_2+r} < 1 + \frac{2r}{a_2+r}$, which is at most 2 since $a_2 \geq r$.

Case 4. Suppose $\vec{x} \in \{\vec{x} : \vec{p}_2 \cdot \vec{x} < 0 \text{ and } \vec{p}_1 \cdot \vec{x} < -r\vec{p}_1 \cdot \vec{p}_2\}$. Same reasoning as Case 3. □

Proof of Theorem 1: Follows from Lemmas 1 and 2. □

2.3.5 Learning Subclasses of Monotone DNF Formulas

In this section we describe algorithms to learn two subclasses of monotone DNF formulas in the UBQ model with one-sided false-positive error, for small values of the boundary radius r. (Learnability in the one-sided UBQ model implies learnability in the IBQ model by simply treating each "don't care" response as positive.)

Specifically, we give an algorithm to learn the class of "read-once monotone DNF formulas in which each term has size at least 4" in the UBQ model with boundary radius $r = 1$. While this is clearly a highly-restrictive class, it is not difficult to show, using standard *prediction preserving reductions* [86], that it is as hard to learn as general DNF formulas in the passive PAC model. Thus our algorithm demonstrates that unreliable queries provide some power over the passive model in a boolean setting. We also give an algorithm to properly learn a subclass of constant-term DNF formulas for any constant r. While the class of k-term DNF formulas is learnable in the passive model, membership queries are required for proper learnability.

One reason for studying the one-sided, false-positive error model is that the monotonicity of the target class provides some inherent ability to handle false-negative errors. In a related model, Angluin and Slonim [13] show how to learn monotone DNF with random false-negative answers to membership queries allowed anywhere in the domain (not just in the boundary region). However, it is not known how to extend their results to handle false positive errors. We hope that the results presented here may be combined with these previous results to produce general techniques for learning monotone concepts with two-sided noise.

Let $y_1, \ldots, y_n$ denote the n boolean variables, and $x = (x_1, \ldots, x_n)$ denote an example. As is commonly done, we view the sample space, $\{0,1\}^n$, as a lattice with top element $\{1\}^n$ and bottom element $\{0\}^n$. The elements are partially ordered by the relation $\leq$, where $v \leq w$ if and only if each bit in v is less than or equal to the corresponding bit in w. The *descendants* (respectively, *ancestors*) of a vector v are all vectors w in the sample space such that $w \leq v$ (respectively, $w \geq v$). For a monotone term, by moving down in the lattice (i.e. changing a 1 to 0), the term can only be "turned off". Thus every monotone term can be represented uniquely by the minimum vector in the ordering $\leq$ for which it is true.

Let $A(i,v)$ be the set of examples obtained by flipping exactly i zeros to ones in vector v. The *parents of* v are the elements of $A(1,v)$, and the *grandparents of* v are the elements of $A(2,v)$. Likewise, $D(i,v)$ is the set of examples obtained by flipping exactly i ones to zeros in v, and for a set V of examples we let $D(i,V) = \cup_{v \in V} D(i,v)$. We define the *children of* v as all elements in $D(1,v)$, and the *siblings of* v are all elements in $D(1, A(1,v))$.

We often think of examples as terms and vice-versa, associating with a monotone term the minimal positive example that satisfies it. For example v let $term(v)$ denote the most specific monotone term that v satisfies. Thus, we say an example is a sibling of a term, meaning that it is a sibling of that term's associated example. Given an example x we define $vars(x)$ to be the set of variables set to 1 by x. We also treat a term t as the set of variables it contains.

We now describe the high-level algorithm that is used to obtain both of our results. Our hypothesis h contains candidates for terms of the target function f, and possibly some additional terms used to ensure that provided counterexamples are not in the boundary region of any known terms. We begin with $h = \emptyset$. We then enter a loop in which we make an equivalence query with our current hypothesis, and then ask a collection of membership queries to update our hypothesis in light of the counterexample received, until a successful equivalence query is made. We maintain the invariant that after i positive counterexamples have been received, h contains i distinct terms $t_1, \ldots, t_i$ of the target concept (and possibly other terms that may or may not be in the target concept).

Following our definitions in Section 2.3.2, we say an example x is *in the boundary region of term* t if $t(x) = 0$, but there exists an $x' > x$ such that $t(x') = 1$ and $dist(x, x') \leq r$. We define the set of *boundary/positive examples* $B = \{v \mid v \text{ is in the positive or boundary region of some term in } h \cap f\}$, where "$h \cap f$" denotes the set of terms that appear in both formulas. Note that B depends not only on f, but on the current hypothesis h as well. Thus, there may be some example $u \in B$ such that u is in the boundary region of some term t in h, but is a truly positive example of the target function f. Such a u might be returned as a positive counterexample to an equivalence query made on h. Thus to maintain the invariant, we need to show that we can always use counterexample u to find a *new* term of f not already in h.

We now describe how a counterexample x is processed so that we can maintain this invariant. When it receives a negative counterexample, our algorithm simply removes from h all terms that classify x as positive. Clearly no term from f will be removed. No terms in the boundary of f will be removed either, since no counterexamples are chosen from the boundary region.

If x is a positive counterexample we first run the procedure Exit-Boundary(x), which returns an example $v \notin B$ such that MQ(v) is positive (this could be a false positive). We then run the following process to "reduce" v so it is "near" a new target term. To ensure that we do not rediscover a known term, the procedure Move-Further *must* return an example that is not in B. Below is our procedure, which is guaranteed to add a new term from f to h.

1. Let v be the example returned from Exit-Boundary. (So v is positive or in the boundary region of a new term of f.)

2. So long as v has some child to which a membership query reports "positive", replace v by that child and repeat. This is the standard Reduce procedure common to many monotone-DNF learning algorithms. (Note that since $v \notin B$ and the target formula is monotone, it follows that no child of v could be in B.)

3. We now call a procedure Move-Further(v) for which one of two cases will occur:

   Case 1: Move-Further(v) returns an example $v' \notin B$ such that v' has strictly fewer ones than v and MQ(v') is positive. In this case we return to step 2 using v' as the current example.

   Case 2: Move-Further(v) reports failure. Here we are guaranteed that v is "near" a new term t_{i+1} of f. Example v is "near" t_{i+1} if the number of irrelevant variables in $vars(v)$ is at most the number of relevant variables from t_{i+1} missing from $vars(v)$, which in turn is at most r. Formally, we require that

   $$|vars(v) - t_{i+1}| \leq |t_{i+1} - vars(v)| \leq r.$$

   In this case we call the procedure Generate-Candidates(v), which returns a polynomial-sized set T of terms with the guarantee that $t_{i+1} \in T$. We then add all terms in T to h.

At this point our algorithm is ready to make its next equivalence query.

Lemma 3 *Given that* Exit-Boundary, Move-Further, *and* Generate-Candidates *satisfy the stated conditions and run in polynomial time for some subclass $\mathcal{C}$ of monotone DNF formulas, the above procedure learns $\mathcal{C}$ in the false-positive-only UBQ model (or the IBQ model) in polynomial time.*

Proof: If there are no counterexamples to h, we are done. The number of positive counterexamples received is at most the number of terms in the target DNF. Once Exit-Boundary is completed, all examples that the Reduce process recurses on have the property that they are positive examples (possibly false positive) outside the positive or boundary region of any term of $h \cap f$. Thus from the correctness of Move-Further and Generate-Candidates, we are guaranteed that a new term is added to h after at most n calls to Move-Further. Furthermore, each negative counterexample removes at least one "extra" term placed in h by Generate-Candidates, and we are guaranteed that there are at most a polynomial number of such terms. Thus, there are only a polynomial number of negative counterexamples, so our algorithm runs in polynomial time as long as all of the provided procedures do. □

Learning A Subclass of Read-Once Monotone DNF Formulas

We now describe how to complete the generic procedure above to obtain an algorithm that learns the class $\mathcal{C}$ of "read-once monotone DNF formulas in which each term has size at least 4" in the UBQ model where $r = 1$.

We begin by describing a utility routine, Study-Example, used in the algorithm. This routine takes an example returned by Move-Further (which is guaranteed to be "near to" some term of the target) and produces a more useful approximation to that term. The desired behavior of the routine Study-Example is specified in Property 1.

Property 1 *Let f be a function in $\mathcal{C}$, and let v be an example such that there exists a term t_{i+1} of f such that v is either equal to, a sibling of, or a child of t_{i+1}. Then* Study-Example *produces an approximation $\hat{t}_{i+1}$ of t_{i+1} along with one of these two guarantees:*

(1) $\hat{t}_{i+1}$ is equal to t_{i+1} or a parent of t_{i+1} (so it is a superset of t_{i+1}), or

(2) $\hat{t}_{i+1}$ is equal to t_{i+1} or a child of t_{i+1} (so it is a subset of t_{i+1}).

The Study-Example routine asks a membership query on all siblings of v. Let P be the set of siblings for which the membership oracle replied "yes." Then Study-Example outputs based on the following cases:

1. If $P = \emptyset$, let $\hat{t}_{i+1} = term(v)$ and report "subset".
2. Otherwise, let u be the term containing exactly the variables in $\cup_{p \in P} vars(p)$.
   (a) If $|u| = |term(v)| + 1$, let $\hat{t}_{i+1} = u$ and report "superset".
   (b) Otherwise ($|u| > |term(v)| + 1$), if some variable $y_i \in vars(v)$ is "responsible for" at least two of the variables in $u - vars(v)$ in the sense that two variables in $u - vars(v)$ are set to 1 in examples setting y_i to 0, then let $\hat{t}_{i+1} = v - \{y_i\}$ and report "subset". (If there are several such variables y_i, just pick one.) For example, let $v = 0011$ and $P = \{1001, 1010, 0101\}$. Then $u = x_1 x_2 x_3 x_4$ and $u - vars(v) = \{x_3, x_4\}$. Here x_3 is responsible for x_1 and x_2, since x_3 is 0 in 1001 and 0101. However, x_4 is only responsible for x_2. Thus in this case we would let y_i be x_3.
   (c) Else let $\hat{t}_{i+1} = u$ and report "superset". (Note: we separate this case from case (a) just for convenience in the proof.)

Lemma 4 *The routine* Study-Example *as described above correctly satisfies Property 1.*

Proof: Note that no siblings of v are in the boundary region of any of the *other* terms in the target function. That is because a sibling of v may have at most two variables set to 1 that are *not* in t_{i+1}, and every other term must have at least four variables not in t_{i+1} (since they all have size at least 4 and the target function is read-once). Thus, we may analyze the routine as if t_{i+1} were the only term in the target function.

We can see that Study-Example behaves correctly in case (1) by noting that if v is positive but none of v's siblings are positive (or false positive), then $term(v)$ is either t_{i+1} or a child of t_{i+1}. The correctness of case (2a) is similarly easy to see because if v is a child of t_{i+1} then u equals t_{i+1} and otherwise u is a parent of t_{i+1}. Notice that if v is a child of t_{i+1} then either case (1) or (2a) occurs.

The reasoning for case (2b) is as follows. If $v = t_{i+1}$ then $\hat{t}_{i+1}$ is a child of t_{i+1}. On the other hand, if v is a sibling of t_{i+1}, then the only variable y_i in v that can possibly be "responsible for" more than one other variable in $u - vars(v)$ is the variable *not* in term t_{i+1}. In fact, if we are not in cases (1) or (2a) and v is a sibling of t_{i+1}, then case (2b) *must* occur because y_i will be responsible for the variable in t_{i+1} missing from v (several variables may be responsible for this one) as well as any other variables added to u. Thus, case (2c) is correct because it can only be reached if $v = t_{i+1}$. □

We now prove our main result of this subsection.

Theorem 2 *The class of read-once monotone DNF formulas where each term in the target formula has at least four variables is exactly learnable in the false-positive-only UBQ model (or the IBQ model) for boundary radius $r = 1$ using polynomial queries and time.*

Proof: By Lemma 3, we need only define subroutines Exit-Boundary, Move-Further, and Generate-Candidates, and show that they satisfy the conditions outlined in the previous section.

We first describe Generate-Candidates. The procedure Generate-Candidates(v) calls Study-Example(v). Study-Example(v) returns a term $\hat{t}_{i+1}$ and a label σ (either "subset" or "superset"). Next, add the pair $(\hat{t}_{i+1}, \sigma)$ to the set $\mathcal{L}$, which stores all the $\hat{t}_i$s and their labels. If label σ = "subset" then place $\hat{t}_{i+1}$ and its children into T. Otherwise, if label σ = "superset" then place $\hat{t}_{i+1}$ and its parents into T.

From Lemma 4 it follows that if Study-Example is called on example v such that v is equal to, a sibling of, or a child of a new term t_{i+1}, then t_{i+1} is placed in T. Thus the set $D(1,T)$ must include all children of t_{i+1}. We therefore have Generate-Candidates return $T \cup D(1,T)$, so all terms in $T \cup D(1,T)$ are added to h. It is clear that this procedure runs in time polynomial in n and adds only a polynomial-sized set of terms to h.

Next, we claim that the identity function satisfies all the requirements for the procedure Exit-Boundary. For positive counterexample x, we want Exit-Boundary(x) to return some example $v \notin B$ such that MQ(v) is positive. If x is a positive counterexample, then certainly MQ(x) is positive.

We now argue that x cannot be in B. Initially, whenever we add a new term t_{i+1} to h, we also add its children, as described above. (Note that it is not possible for Generate-Candidates to accidently add an additional term t_{i+2} to h without its children. If it did so, t_{i+2} would have to be in $D(1,T)$ but not in T. But since f is read-once and each term has size at least 4, it is not possible to have two terms of f, t_{i+1} and t_{i+2}, such that $t_{i+1} \in T$ and $t_{i+2} \in D(1,T)$.) Once a term of f or any of its children is placed in h, it cannot be removed by any negative counterexample (since any child of a term in f is in the boundary region of f). So no positive counterexample can be in B. Thus, Exit-Boundary(x) simply returns x.

We now describe the procedure Move-Further(v). Note that the input v has the properties that v is not in B and that MQ(v)=1. Furthermore, since v must have failed the standard Reduce procedure, $MQ(v')$ is negative for all $v' \in D(1,v)$.

1. For each $\hat{t}_j$ in $\mathcal{L}$ (for $j = 1, 2, \ldots, i$) that is labeled "subset", set every variable in $vars(\hat{t}_j)$ to 0 in v. (This new example is still in the positive or boundary region of a new term since f is read-once. And since f is monotone, this example is not in B.) We fix these variables at 0 for the remainder of this procedure.

2. Let V be the set of variables set to 0 by v and not fixed to 0 in Step 1. For each variable $y_\ell \in V$, consider the example v' obtained from v by flipping to 0 all

variables in the terms $\hat{t}_j$ that contain y_ℓ, and then flipping y_ℓ to 1. Let P be the set of all such examples for which a membership query reports "positive".

3. (a) If there is an example in P that has fewer 1's than v, then return this example.
   (b) If not, then query all children and grandchildren of examples in P and if one of them has fewer 1's than v and is reported as positive, then return this example.
   (c) Otherwise report failure.

Lemma 5 *The procedure* Move-Further(v) *as described above either returns an example $v' \notin B$ such that v' has strictly fewer ones than v and MQ(v') is positive, or it returns failure, in which case $|vars(v) - t_{i+1}| \leq |t_{i+1} - vars(v)| \leq 1$.*

Proof: Move-Further maintains the invariant that $v \notin B$ and v is in the positive or boundary region of a new term of f. We have already argued that this holds after Step 1. Thus, at this point, there exists some term $t_{i+1} \in f$, distinct from $t_1, \ldots, t_i$, such that v sets to 0 at most one variable in t_{i+1}.

We now argue that each example in P has at most one variable in common with term t_j for $1 \leq j \leq i$. If $v' \in P$ was obtained by flipping to 1 some variable y appearing in, say, term t_j ($j \leq i$) then one of two cases holds. If $\hat{t}_j$ is a "subset" of t_j, then y is the *only* variable that $vars(v')$ has in common with t_j, since all others in t_j have been fixed to 0. Otherwise, $\hat{t}_j$ is a "superset" of t_j. In this case, y is also in $\hat{t}_j$, so to obtain v' we flipped all the rest of the variables in $\hat{t}_j$ to 0. Thus, since each term of f has at least 4 literals, no example in P is in B. So if an example v' is returned in step (3a) or (3b) then it has the desired properties: $v' \notin B$, MQ(v') is positive, and $|vars(v')| < |vars(v)|$.

We now argue that if step (3c) reports failure then v satisfies $|vars(v) - t_{i+1}| \leq |t_{i+1} - vars(v)| \leq 1$. We have already argued that v is the the positive or boundary region of a new target term, t_{i+1}, and thus at most one relevant variable from t_{i+1} is missing from $vars(v)$ (i.e., $|t_{i+1} - vars(v)| \leq 1$). Furthermore, if $vars(v)$ contains all the variables in t_{i+1} then $t_{i+1} = vars(v)$, because all irrelevant variables would have been removed by the standard Reduce procedure. (Recall that one input condition on v is that no children of v are positive.) Therefore, if $|t_{i+1} - vars(v)| = 0$, then $|vars(v) - t_{i+1}| = 0$ as well.

Otherwise, suppose that $|t_{i+1} - vars(v)| = 1$, so v is missing one relevant variable, y_ℓ, from t_{i+1}. Then when y_ℓ is added in step (2), the membership query would be positive and thus this example would be added to P. Now suppose there were two or more variables in $vars(v)$ that were not in t_{i+1}. Then an example in which two of those variables were set to 0 would have been returned in either step (3a) or (3b). Thus if we reach step (3c) we know that $|vars(v) - t_{i+1}| \leq |t_{i+1} - vars(v)| \leq 1$ holds. □

Finally, we claim that since the example v returned by Move-Further satisfies the property $|vars(v) - t_{i+1}| \leq |t_{i+1} - vars(v)| \leq 1$, v must either be equal to t_{i+1}, or be a sibling or a child of t_{i+1}. Thus, Study-Example is called on a vector satisfying the conditions of Property 1, proving the correctness of Generate-Candidates and of Theorem 2.
□

Learning $(r+1)$-Separable k-Term Monotone DNF Formulas

We now show that a subclass of monotone k-term DNF formulas is properly learnable in the false-positive-only UBQ model for any constant boundary radius. We say that two terms t_i and t_j are *ℓ-separable* if there are ℓ variables in t_j that are not in t_i, and there are ℓ variables in t_i that are not in t_j. A monotone DNF formula f is *ℓ-separable* if all pairs (t_i, t_j) of terms of f are ℓ-separable.

Theorem 3 *The class of $(r+1)$-separable k-term monotone DNF formulas is exactly learnable in the false-positive-only UBQ model (or the IBQ model) using polynomial queries and time (for r and k constant). Furthermore, all equivalence queries made by our algorithm are $(r+1)$-separable k-term monotone DNF formulas.*

Proof: We first prove this result under the assumption that Generate-Candidates not only finds a set of candidates that contains some new term of the target formula, but has the power to "guess" which one is right. Thus h always contains a subset of the terms of the target. Then we argue that our algorithm can be modified to remove this assumption.

We first define the procedure Exit-Boundary. For each term t_i of f already in h, choose a set s_i of $(r+1)$ variables in t_i. Let $S = \cup_i s_i$. Then let $v' = v$ with all variables in S set to 0. The procedure Exit-Boundary(v) performs a membership query on each possible v' obtained in this fashion, and returns the first such example for which the membership oracle replies "yes." Thus, each query sets up to $(r+1)(k-1)$ variables in $vars(v)$ to 0. We now prove that Exit-Boundary is correct.

Lemma 6 *The procedure* Exit-Boundary(v) *successfully returns an example $v' \notin B$ for which $MQ(v') =$ "positive."*

Proof: Since v was a positive counterexample, it must be in the non-boundary positive region of some term t_{new} in $f - h$. Suppose it is also in the boundary region of some terms in h. Consider one such term t_{known}. Since f is $(r+1)$-separable, if we set to zero $r+1$ variables in $vars(v) \in t_{known} - t_{new}$ then v will no longer be in the boundary of t_{known}. However, we still know that all variables in t_{new} are in $vars(v)$ since we do not change any variables in t_{new}. (In fact, if all $r+1$ variables in $t_{known} - t_{new}$ are 1 in v, then it suffices to pick any r of them to set to 0, since we know that v is already in the boundary region of t_{known}.) We can repeat this for the at most $(k-1)$ other terms in h. Thus after setting at most $(r+1)(k-1)$ variables in $vars(v)$ to 0, we obtain an example that is not in B and is in the truly positive region of t_{new}. Since this is one of the examples queried by Exit-Boundary we know that at least one membership query will respond "yes".

Of course, it is possible that some other membership query responds "yes". However, note that Exit-Boundary never queries any example in B, since all examples queried are obtained by setting $r+1$ variables from each known term to 0. Thus, in this case we are still guaranteed that the example v' returned is not in B and that MQ(v') is positive. □

The procedure Move-Further works as follows. For each i such that $r+1 \leq i \leq rk$, it performs a membership query on all examples v' in $D(i, A(r,v))$ that are not in B and returns the first v' for which $\mathrm{MQ}(v') = 1$. If no such examples are found after all values of i have been tried, then it returns "failure." If Move-Further returns v', then v' must have strictly fewer ones than v, since $v' \in D(i, A(r,v))$ for some $i > r$.

We now argue that when Move-Further(v) reports failure the following two properties hold:

1. Example v sets to 0 at most r variables from term t_{i+1} of the target formula (i.e. $|t_{i+1} - vars(v)| \leq r$).

2. The number of variables *not* in t_{i+1} that are one in v is at most the number of variables *in* t_{i+1} that are zero in v (i.e. $|vars(v) - t_{i+1}| \leq |t_{i+1} - vars(v)|$).

Since $v \notin B$ and $\mathrm{MQ}(v) = 1$, v must be in the positive or boundary region of some new term from f. Since the adversary can reply "positive" only on an example that sets to 0 at most r variables from a term in f, the first property follows.

We now prove that the second property holds. Let t_{i+1} be any new term of f for which v has $\ell \leq r$ variables set to 0. Suppose that the second property fails. Thus there are at least $\ell + 1$ variables not in t_{i+1} that are one in v. Since the target is $(r+1)$-separable we know that t_{i+1} is not in B. Since t_{i+1} has at most r variables set to 0 in v, at least one example x in $A(r,v)$ has all variables in t_{i+1} set to 1. When adding these r 1's, we have at worst just set to 1 r variables in each of the known terms. For each term $t_i \in h$, let s_i be the set of all variables in t_i but not in t_{i+1}, and let $S = \cup_i s_i$. Let x' be example x with all variables in S set to zero. Since the target concept is monotone and $(r+1)$-separable, we know that x' is still in the positive region of term t_{i+1} and that $x' \notin B$. Move-Further queries examples in $D(i, A(r,v))$ for all i such that $r+1 \leq i \leq kr$, so x' must be one of the examples queried by the procedure. Finally, since there are at least $\ell + 1$ variables not in t_{i+1} that are 1 in v, and since $x' \in D(i, A(r,v))$ for some $i > r \geq \ell$, x' must have strictly fewer ones than v. But this contradicts the assumption that Move-Further(v) reported failure. Thus the second property holds. Also note that only a polynomial number of examples were queried (since r and k are constant). Thus this procedure runs in polynomial time.

Generate-Candidates(v) lets $T = \cup_{i=0}^{r} D(i, A(r, v))$ and non-deterministically guesses which one is in f. It follows from the correctness of Generate-Candidates that t_{i+1} is placed in T.

To remove the need to non-deterministically select the right term from T we just try all guesses. We halt when failure is detected because a negative counterexample is received or a $(k+1)$st positive counterexample is received. Since r and k are constant, only a polynomial number of runs occur and thus the overall queries and time complexity are still polynomial. □

The proof of Theorem 3 can be extended to obtain the following result.

Corollary 4 *The class of 2-term monotone DNF formulas is exactly learnable by the class of 2-term monotone DNF formulas in polynomial time in the false-positive-only UBQ model (or the IBQ model) with a boundary region of radius r (for constant r).*

Proof: Let $f = t_1 + t_2$. If t_1 and t_2 are $(r+1)$-separable then the result immediately follows. Thus, without loss of generality, assume that t_2 has all variables from t_1 except at most r of them, as well as any number of additional variables. If t_1 is placed in h first then no counterexample is created by t_2 since it is entirely contained within the boundary region of t_1. If t_2 is placed in h first, then we receive a positive counterexample for t_1 (unless it is contained within t_2's boundary in which case we are done). This counterexample is processed to add t_1 to h. □

2.4 Concluding Remarks

In this chapter, we have begun an investigation of modeling learning with the help of imperfect teachers. We have seen how to learn monotone boolean concepts despite the failures of a randomly-ignorant teacher. We have introduced two related models of learning with noise near the boundary of the target concept, and we have presented positive results in these models in both continuous and discrete domains.

However, there is much more work to be done. The algorithms described here learn fairly simple concept classes. While Goldman and Mathias [54] have shown that it is possible to learn non-monotonic concepts with an incomplete membership oracle, it has proven difficult to learn more complex concept classes in this model. We do not yet know how to extend our results in the unreliable boundary query model to learn general monotone DNF formulas or the intersection of more than two halfspaces. One exciting future project would be to find a general method for transforming classes of PAC-memb or exact-learning algorithms to work in the IBQ or UBQ model. The lack of such general positive results suggests that we need additional power to accurately model the practical problem of learning from imperfect teachers.

Why is it so difficult to generate strong, general positive results? One possibility is that our current models of fallible teachers are too strong. However, it is unlikely that this is the cause of the problem. A randomly-fallible teacher is actually a fairly benign model; there is only a low chance that a large set of critical answers is completely corrupted. Though the unreliable boundary model potentially allows adversarial behavior, these errors are confined to a restricted part of the instance space. Thus, well-designed learning algorithms can avoid being mislead by adversarial noise, as evidenced by our positive results. Perhaps our learning techniques are not powerful enough. However, I believe it is the definition of learning itself which is the problem.

In the field of computational learning theory, the vast majority of work on exact learning is evaluated in a worst-case scenario. Particularly in the study of learning with noisy or incomplete data, we tend to consider malicious adversaries in order to obtain worst-case bounds. The standard alternative is to postulate a uniform, random source of noise, a model that is well-understood but is not always realistic.

I suspect the answer lies somewhere in-between. In practice, most noise processes are neither malicious nor totally random, but are simply defined by processes that we do not completely understand. Perhaps, as in the PAC learning model where examples are chosen according to a specific (but not necessarily known) distribution D, we need a noise model where the noise varies over the instance space in some specific (but perhaps unknown) way. The challenge then would be to find a meaningful definition of learning in which it is possible to learn complicated concepts (perhaps in an approximate, PAC-like way) given any non-pathological source of noise. Kearns and Schapire's work on learning P-concepts [65] is a good preliminary description of such a model; a probabilistic concept may simply be the learner's representation of a noise process that is not well understood. I believe that more work along these lines is needed to suggest practical approaches to learning from imperfect data.

CHAPTER 3

The Power of Team Exploration: Two Robots Can Learn Directed Graphs with Indistinguishable Nodes

3.1 Introduction

Consider a robot trying to construct a street map of an unfamiliar city by driving along the city's roads. Since many streets are one-way, the robot may be unable to retrace its steps. However, it can learn by using street signs to distinguish intersections. Now suppose that it is nighttime and that there are no street signs. The task becomes significantly more challenging.

In this chapter we present a probabilistic polynomial-time algorithm to solve an abstraction of the above problem by using two cooperating learners. Instead of learning a city, we learn a strongly-connected directed graph G with n nodes. Every node has d outgoing edges labeled from 0 to $d-1$. Nodes in the graph are indistinguishable, so a robot cannot recognize if it is placed on a node that it has previously seen. Moreover, since the graph is directed, a robot is unable to retrace its steps while exploring.

One might imagine that a straightforward learning algorithm in this model would run in time polynomial in some property of the graph's structure such as cover time or mixing time. Any such algorithm could require an exponential number of steps, however, since the cover time and mixing time of directed graphs can be exponential in the number of nodes. In this chapter, we present a probabilistic algorithm for two robots to learn any strongly-connected directed graph in $O(d^2n^5)$ steps with high probability.

The two robots in our model can recognize when they are at the same node and can communicate freely by radio. Radio communication is used only to synchronize actions. In fact, if we assume that the two robots move synchronously and share a polynomial-length random string, then no communication is necessary. Thus with only minor modifications, our algorithms may be used in a distributed setting.

Our main algorithm runs without prior knowledge of the number of nodes in the graph, n, in time polynomial in n. We show that no probabilistic polynomial-time algorithm for a single robot with a constant number of pebbles can learn all unlabeled directed graphs when n is unknown. Thus, our algorithms demonstrate that two robots are strictly more powerful than one.

3.1.1 Motivation for the Model

Our work explores a worst-case scenario in which the robots cannot recognize any previously-seen location. While this model may appear extreme, it provides upper bounds for practical cases in which robots fail to recognize familiar locations.

An example of this problem is found in Horswill's thesis [59]. Horswill describes Polly, a robot that wanders the seventh floor of the MIT AI Laboratory and gives tours to visitors. Polly has a built-in map of her environment. Her navigation system relies on recognizing landmarks, such as the kitchen or the elevators, by matching sensor input (from an on-board vision system) to stored images of the landmarks. Polly resides in a fairly benign environment; the lighting and most obstacles remain fairly constant over time.

However, even with a correct map, Polly can get lost. Sometimes she fails to recognize landmarks due to the inadequacies of her sensors. At other times, Polly is concerned with other tasks (such as obstacle avoidance) while passing a landmark and simply misses it. In either case, Polly can reach a familiar spot on her map without recognizing her location. As in our model, she can take several distinct actions at each node (i.e., "go straight," "turn left") without knowing where she is. If a robot with a complete map can get lost, it is reasonable to imagine that a robot learning a map might often fail to recognize places it has seen before.

Our work shows that enough information is available for a team of two robots to learn graphs even if the nodes are completely indistinguishable. We show later how our general algorithm can use partial distinguishing information to expedite the learning process. Thus, if the robots can recognize landmarks some of the time, the learning algorithm runs more quickly.

3.1.2 Related Work

Theoretical Results on Graph Exploration and Team Learning

Previous results showing the power of team learning are plentiful, particularly in the field of inductive inference (see Smith [105] for an excellent survey). Several team learning papers explore the problems of combining the abilities of a number of different learners. Cesa-Bianchi *et al.* [36] consider the task of learning a probabilistic binary sequence given the predictions of a set of experts on the same sequence. They show how to combine the prediction strategies of several experts to predict nearly as well as the best of the experts. In a related paper, Kearns and Seung [66] explore the statistical problems of combining several independent hypotheses to learn a target concept from a known, restricted concept class. In their model, each hypothesis is learned from a different, independently-drawn set of random examples, so the learner can combine the results to perform significantly better than any of the hypotheses alone.

There are also many results on learning unknown graphs, but most previous work has concentrated on learning undirected graphs or graphs with distinguishable nodes. For example, Deng and Papadimitriou consider the problem of learning strongly-connected, directed graphs with *labeled* nodes, so that the learner can recognize previously-seen nodes. They provide a learning algorithm whose competitive ratio (versus the optimal time to traverse all edges in the graph) is exponential in the deficiency of the graph [45, 20]. Betke, Rivest, and Singh introduce the notion of *piecemeal* learning of undirected graphs with labeled nodes. In piecemeal learning, the learner must return to a fixed starting point from time to time during the learning process. Betke, Rivest, and Singh provide linear algorithms for learning grid graphs with rectangular obstacles [21], and with Awerbuch [15] extend this work to show nearly-linear algorithms for general graphs.

Rivest and Schapire [92, 93] explore the problem of learning deterministic finite automata whose nodes are *not* distinguishable except by the observed output. We rely heavily on their results in this chapter. Their work has been extended by Freund *et al.* [50], and by Dean *et al.* [44]. Freund *et al.* analyze the problem of learning finite automata with average-case labelings by the observed output on a random string, while Dean *et al.* explore the problem of learning DFAs with a robot whose observations of the environment are not always reliable. Ron and Rubinfeld [95] present algorithms for learning "fallible" DFAs, in which the data is subject to persistent random errors. Recently, Ron and Rubinfeld [94] have shown that a teacher is unnecessary for learning finite automata with small cover time.

Teamwork and Learning in Robotics

Team research in robotics tends to address issues of control. Parker[85] divides the results in this field into two camps: "swarm" cooperation and "intentional" cooperation. "Swarms" are teams of homogeneous robots that all independently perform the same task. For example, swarms of robots might be used for harvesting crops or for mine-sweeping fields in a war zone. A key property of a swarm is that no individual robot is essential for the completion of the task.

There have been many projects studying the learning and behavior of robot swarms.

Mataric [80] uses reinforcement learning [107] to teach "foraging" behavior to a team of twenty independently-controlled robots. Brooks, et al., [34] describe parallel algorithms for swarms of robots to select and clear a site for construction of a lunar research station. The robots in this system form a "herd;" each acts independently, following some simple rules that may involve the location and behavior of other nearby robots.

In contrast, intentional cooperation refers to cases in which several (perhaps heterogeneous) robots work together to achieve tasks that no one robot could complete alone. Often there is a single global controller and planner for the entire system. Rus, Donald and Jennings [96] explore team cooperation strategies for moving heavy furniture. They present several strategies, both with and without a global planner. Parker [85] describes fault-tolerant distributed cooperation methods for heterogeneous robot teams to solve complex tasks such as cleaning up a toxic spill. However, most cooperative projects require the robots to achieve a goal in an already-known environment, rather than asking the robots to *learn* a map of their world.

Some robotics projects do focus on environment learning. Mataric [79] shows how a single robot might learn a map of an unknown office environment. For her robot, whose navigation relies on sonar sensors and a compass, the ability to distinguish similarly-shaped landmarks is a key issue. Mataric solves this problem using the robot's estimated position, based on compass readings and approximate distances traveled.

Yanco and Stein [112] describe a project in which teams of two or three robots actually learn how to communicate with one another to work together. The robot teams develop their own communication language through reinforcement learning. The teams successfully learn small languages of up to 10 words. However, it is not clear how the training time scales as the desired language grows more complex.

Our model follows the intentional cooperation paradigm; neither learner can succeed alone. Our robots are homogeneous in the sense that they have the same capabilities, but they may perform different actions at different times. Finally, there is a sort of global control in our system; the two robots execute a single algorithm. However, if the robots have a shared clock or a reliable method of communication, no additional control source is needed.

A Single Robot: Searching Graphs with Pebbles

In our model a single robot is powerless because it is completely unable to distinguish one node from any other. However, when equipped with a number of pebbles that can be used to mark nodes, the single robot's plight improves. Rabin first proposed the idea of dropping pebbles to mark nodes [89]. This suggestion led to a body of work exploring the searching capabilities of a finite automaton supplied with pebbles.

Blum and Sakoda [25] consider the question of whether a finite set of finite automata can search a 2 or 3-dimensional obstructed grid. They prove that a single automaton with just four pebbles can completely search any 2-dimensional finite maze, and that a single automaton with seven pebbles can completely search any 2-dimensional infinite maze. They also prove, however, that no collection of finite automata can search every 3-dimensional maze. Blum and Kozen [26] improve this result to show that a single automaton with 2 pebbles can search a finite, 2-dimensional maze. Their results imply that mazes are strictly easier to search than planar graphs, since they also show that no single automaton with pebbles can search all planar graphs.

Savitch [98] introduces the notion of a maze-recognizing automaton (MRA), which is a DFA with a finite number of distinguishable pebbles. The mazes in Savitch's paper are n-node 2-regular graphs, and the MRAs have the added ability to jump to the node with the next higher or lower number in some ordering. Savitch shows that maze-recognizing automata and $\log n$ space-bounded Turing machines are equivalent for the problem of recognizing threadable mazes (i.e., mazes in which there is a path between a given pair of nodes).

Most of these papers use pebbles to model memory constraints. For example, suppose that the nodes in a graph are labeled with $\log n$-bit names and that a finite automaton with $k \log n$ bits of memory is used to search the graph. This situation is modeled by a single robot with k distinguishable pebbles. A robot dropping a pebble at a node corresponds to a finite automaton storing the name of that node. In our work, by contrast, we investigate time rather than space constraints. Since memory is now relatively cheap but time is often critical, it makes sense to ask whether a robot with any reasonable amount of memory can use a constant number of pebbles to learn graphs in polynomial time.

Cook and Rackoff generalized the idea of pebbles to jumping automata for graphs (JAGs) [39]. A jumping automaton is equipped with pebbles that can be dropped to mark nodes and that can "jump" to the locations of other pebbles. Thus, this model is similar to our two-robot model in that the second robot may wait at a node for a while (to mark it) and then catch up to the other robot later. However, the JAG model is somewhat broader than the two-robot model. Cook and Rackoff show upper and lower bounds of $\log n$ and $\log n / \log \log n$ on the amount of space required to determine whether there is a directed path between two designated nodes in any n-node graph. JAGs have been used primarily to prove space efficiency for st-connectivity algorithms, and they have recently resurfaced as a tool for analyzing time and space tradeoffs for graph traversal and connectivity problems (*e.g.* [18, 87, 46]).

Universal traversal sequences have been used to provide upper and lower bounds for the exploration of undirected graphs. Certainly, a universal traversal sequence for the class of directed graphs could be used to learn individual graphs. However, for arbitrary directed graphs with n nodes, a universal traversal sequence must have size exponential in n. Thus, such sequences will not provide efficient solutions to our problem.

3.1.3 Strategy of Our Learning Algorithm

The power behind the two-robot model lies in the robots' abilities to recognize each other and to move independently. Nonetheless, it is not obvious how to harness this power. If the robots separate in unknown territory, they could search for each other for an amount of time exponential in the size of the graph. Therefore, in any successful strategy for our model the two robots must always know how to find each other. One strategy that satisfies this requirement has both robots following the same path whenever they are in unmapped territory. They may travel at different speeds, however, with one robot scouting ahead and the other lagging behind. We call this a *lead-lag strategy.* In a lead-lag strategy the lagging robot must repeatedly make a difficult choice. The robot can wait at a particular node, thus marking it, but the leading robot may not find this marked node again in polynomial time. Alternatively, the lagging robot can abandon its current node to catch up with the leader, but then it may not know how to return to that node. In spite of these difficulties, our algorithms successfully employ a lead-lag strategy.

Our work also builds on techniques of Rivest and Schapire [93]. They present an algorithm for a single robot to learn minimal deterministic finite automata. With the help of an equivalence oracle, their algorithm learns a homing sequence, which it uses in place of a reset function. It then runs several copies of Angluin's algorithm [5] for learning DFAs given a reset. Angluin has shown that any algorithm for actively learning DFAs requires an equivalence oracle [4].

In this chapter, we introduce a new type of homing sequence for two robots. Because of the strength of the homing sequence, our algorithm does not require an equivalence oracle. For any graph, the expected running time of our algorithm is $O(d^2n^5)$. In practice, our algorithm can use additional information such as indegree, outdegree, or color of nodes to find better homing sequences and to run faster.

Note that throughout the chapter, the analyses of the algorithms account for only the number of steps that the robots take across edges in the graph. Additional calculations performed between moves are not considered, so long as they are known to take time polynomial in n. In practice, such calculations would not be a noticeable factor in the running time of our algorithms.

Two robots can learn specific classes of directed graphs more quickly, such as the class of graphs with high conductance. Conductance, a measure of the expansion properties of a graph, was introduced by Sinclair and Jerrum [100]. The class of directed graphs with high conductance includes graphs with exponentially-large cover time. We present a randomized algorithm that learns graphs with conductance greater than $n^{-\frac{1}{2}}$ in $O(dn^4 \log n)$ steps with high probability.

3.2 Preliminaries

Let $G = (V, E)$ represent the unknown graph, where G has n nodes, each with outdegree d. An edge from node u to node v with label i is denoted $\langle u, i, v\rangle$. We say that an algorithm *learns* graph G if it outputs a graph isomorphic to G. Our algorithms maintain a graph *map* which represents the subgraph of G learned so far. Included in *map* is an implicit start node u_0. It is worth emphasizing the difference between the target graph G and the graph *map* that the learner constructs. The graph *map* is meant to be a map of the underlying environment, G. However, since the robots do not always know their exact location in G, in some cases *map* may contain errors and therefore may not be isomorphic to any subgraph of G. Much of the notation in this section is needed to specify clearly whether we are referring to a robot's location in the graph G or to its putative location in *map*.

A node u in *map* is called *unfinished* if it has any unexplored outgoing edges. Node u is *map-reachable* from node v if there is a path from v to u containing only edges in *map*. For robot k, the node in *map* corresponding to k's location in G if *map* is correct is denoted $Loc_M(k)$. Robot k's location in G is denoted $Loc_G(k)$.

Let f be an automorphism on the nodes of G such that

$$\forall a, b \in G, \langle a, i, b\rangle \in G \iff \langle f(a), i, f(b)\rangle \in G.$$

We say *nodes* c and d are *equivalent* (written $c \equiv d$) iff there exists such an f where $f(c) = d$.

We now present notation to describe the movements of k robots in a graph. An *action* A_i of the ith robot is either a label of an outgoing edge to explore, or the symbol

r for "rest." A *k-robot sequence of actions* is a sequence of *steps* denoting the actions of the k robots; each step is a *k-tuple* $\langle A_0, \ldots, A_{k-1}\rangle$. For sequences s and t of actions, $s \circ t$ denotes the sequence of actions obtained by concatenating s and t.

A *path* is a sequence of edge labels. Let $|path|$ represent the length of *path*. A robot *follows* a path by traversing the edges in the path in order beginning at a particular start node in *map*. The node in *map* reached by starting at u_0 and following *path* is denoted $final_M(path, u_0)$. Let s be a two-robot sequence of actions such that if both robots start together at any node in any graph and execute s, they follow exactly the same path, although perhaps at different speeds. We call such a sequence a *lead-lag* sequence. Note that if two robots start together and execute a lead-lag sequence, they end together. The node in G reached if both robots start at node a in G and follow lead-lag sequence s is denoted $final_G(s, a)$.

For convenience, we name our robots Lewis and Clark. Whenever Lewis and Clark execute a lead-lag sequence of actions, Lewis leads while Clark lags behind.

3.3 Using a Reset to Learn

Learning a directed graph with indistinguishable nodes is difficult because once both robots have left a known portion of the graph, they do not know how to return. This problem would vanish if there were a reset function that could transport both robots to a particular start node u_0. We describe an algorithm for two robots to learn directed graphs given a reset. Having a reset is not a realistic model, but this algorithm forms the core of later algorithms, which learn without a reset.

Algorithm **Learn-with-Reset** maintains the invariant that if a robot starts at u_0, there is a directed path it can follow that visits every node in *map* at least once. To learn a new edge (one not yet in *map*) using algorithm **Learn-with-Reset**, Lewis crosses the edge and then Clark tours the entire known portion of the map. If they encounter each other, Lewis's position is identified; otherwise Lewis is at a new node. The depth-first strategy employed by **Learn-Edge** is essential in later algorithms. In **Learn-with-Reset**, as in all the procedures in this chapter, variables are passed by reference and are modified destructively.

Lemma 7 *The variable* path *in* **Learn-with-Reset** *denotes a tour of length* $\leq dn^2$ *that starts at* u_0 *and traverses all edges in map.*

Learn-with-Reset( ):
$map := (\{u_0\}, \emptyset)$ { map is the graph consisting of node u_0 and no edges }
$path :=$ empty path { $path$ is the null sequence of edge labels }
$k := 1$ { k counts the number of nodes in map }
while there are unfinished nodes in map
  do Learn-Edge(map,$path$,k)
    Reset
return map

Learn-Edge(map,$path$,k**):** {$path$ = tour through all edges in map }
Lewis follows $path$ to $final_M(path, u_0)$
$u_i :=$ some unfinished node in map map-reachable from Loc_M(Lewis)
Lewis moves to node u_i; append the path taken to $path$
pick an unexplored edge l out of node u_i
Lewis moves along edge l; append edge l to $path$ { Lewis crosses a new edge }
Clark follows $path$ to $final_M(path, u_0)$ { Clark looks for Lewis }
if $\exists j < k$ such that Clark first encountered Lewis at node u_j
  then add edge $\langle u_i, l, u_j \rangle$ to map
  else add new node u_k and edge $\langle u_i, l, u_k \rangle$ to map
    $k := k + 1$

Proof: Every time Lewis crosses an edge, that edge is appended to *path*. Since no edge is added to *map* until Lewis has crossed it, *path* must traverse all edges in *map*. In each call to **Learn-Edge**, at most n edges are added to *path*. The body of the while loop is executed dn times, so $|path| \leq dn^2$. □

Lemma 8 *Map is always a subgraph of* G.

Proof: Initially *map* contains a single node u_0 and no edges. Assume inductively that *map* is a subgraph of G after the cth call to **Learn-Edge** (when *map* has c edges). To learn the next edge, the algorithm chooses a node u_i in *map* and explores a new edge $e = \langle u_i, l, v\rangle$. By Lemma 7 and the inductive hypothesis, if Clark encounters Lewis at u_j then v is identified as u_j. Otherwise v is recognized to be a new node and named u_k. Therefore the updated map is a subgraph of G. □

Lemma 9 *If map contains any unfinished nodes, then there is always some unfinished node in* map *map-reachable from* final$_M$*(*path,u_0*).*

Proof: Suppose this assumption were false. Then there is some unfinished node in *map*, but all nodes of *map* in the strongly-connected component of *final*$_M$(*path*,u_0) are finished. Thus by Lemma 8, there are no additional edges of G leaving that component, so graph G is not strongly connected. □

Theorem 5 *After $O(d^2n^3)$ moves and dn calls to* **Reset**, **Learn-with-Reset** *halts and outputs a graph isomorphic to G.*

Proof: The correctness of the output follows from Lemmas 7 – 9. For each call to **Learn-Edge**, each robot takes *length*(*path*)$\leq dn^2$ steps. The algorithm **Learn-Edge** is executed at most dn times, so the algorithm halts within $O(d^2n^3)$ steps. □

3.4 Homing Sequences

In practice, robots learning a graph do not have access to a reset function. In this section we suggest an alternative technique: we introduce a new type of homing sequence for two robots.

Intuitively, a homing sequence is a sequence of actions whose observed output uniquely determines the final node reached in G. Rivest and Schapire [93] show how a single robot with a teacher can use homing sequences to learn strongly-connected minimal DFAs. The output at each node indicates whether that node is an accepting or rejecting state of the automaton. If the target DFA is not minimal, their algorithm learns the minimal encoding of the DFA. In other words, their algorithm learns the function that the graph computes rather than the structure of the graph.

In unlabeled graphs the nodes do not produce output. However, two robots can generate output indicating when they meet.

Definitions: Each step of a two-robot sequence of actions produces an output symbol T if the robots are together and S if they are separate. An *output sequence* is a string in $\{T, S\}^*$ denoting the observed output of a sequence of actions. Let s be a lead-lag sequence of actions and let a be a node in G. Then *output(s,a)* denotes the output produced by executing the sequence s, given that *both* robots start at a. A lead-lag sequence s of actions is a *two-robot homing sequence* iff $\forall$ nodes $u, v \in G$,

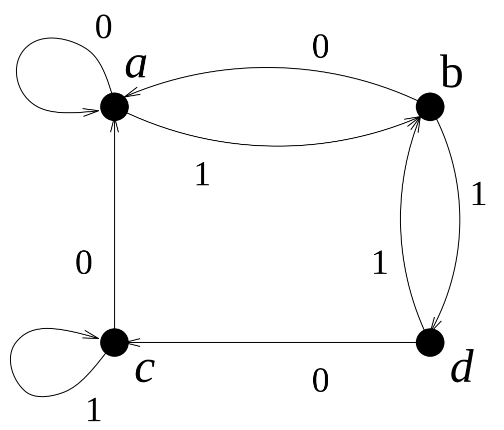
$$output(s,u) = output(s,v) \Rightarrow final_G(s,u) \equiv final_G(s,v).$$

Because the output of a sequence depends on the positions of both robots, it provides information about the underlying structure of the graph. Figure 3.1 illustrates the definition of a two-robot homing sequence. This new type of homing sequence is powerful. Unlike most previous learning results using homing sequences, our algorithms do not require a teacher to provide counterexamples.

In fact, two robots on a graph define a DFA whose states are pairs of nodes in G and whose edges correspond to pairs of actions. Since the automata defined in this way form a restricted class of DFAs, our results are not inconsistent with Angluin's work [4] showing that a teacher is necessary for learning general DFAs.

Theorem 6 *Every strongly-connected directed graph has a two-robot homing sequence.*

Proof: The following algorithm (based on that of Kohavi [69, 93]) constructs a homing sequence: Initially, let h be empty. As long as there are two nodes u and v in G such that *output(h,u)* = *output(h,v)* but *final(h,u)* $\not\equiv$ *final(h,v)*, let x be a lead-lag sequence whose output distinguishes *final(h,u)* from *final(h,v)*. Since *final(h,u)* $\not\equiv$ *final(h,v)* and



$$\text{output} = \begin{cases} T & \text{if Lewis and Clark are together at a node} \\ S & \text{if Lewis and Clark are at separate nodes.} \end{cases}$$

$$\text{homing sequence } h = \begin{cases} \text{Lewis:} & 0r1r \\ \text{Clark:} & r0r1 \end{cases} \qquad \text{sequence } s = \begin{cases} \text{Lewis:} & 0r \\ \text{Clark:} & r0 \end{cases}$$

start node	output	end node
a	$TTST$	b
b	$STST$	b
c	$STST$	b
d	$STTT$	c

start node	output	end node
a	TT	a
b	ST	a
c	ST	a
d	ST	c

Figure 3.1: Illustration of a two-robot homing sequence and a lead-lag sequence. Note that both h and s are lead-lag sequences. However, sequence h is a two-robot homing sequence, because for each output sequence there is a unique end node. (Note that the converse is not true.) Sequence s is not a two-robot homing sequence, because the robots may end at nodes a or c and yet see the same output sequence ST.

G is strongly connected, such an x always exists. Append x to h.

Each time a sequence is appended to h, the number of different outputs of h increases by at least 1. Since G has n nodes, there are at most n possible output sequences. Therefore, after $n-1$ iterations, h is a homing sequence. □

In Section 3.5 we show that it is possible to find a counterexample x efficiently. Given a strongly-connected graph G and a node a in G, a pair of robots can verify whether they are together at a node equivalent to a on some graph isomorphic to G. We describe a verification algorithm **Verify**(a, G) in Section 3.5. The sequence of actions returned by a call of **Verify**(u, G) is always a suitable counterexample x. Using the bound from Corollary 8, we claim that this algorithm produces a homing sequence of length $O(n^4)$ for all graphs. Note that shorter homing sequences exist; the homing sequence produced by algorithm **Learn-Graph** in Section 3.5 has length $O(dn^3)$.

3.4.1 Using a Homing Sequence to Learn

Given a homing sequence h, an algorithm can learn G by maintaining several running copies of **Learn-with-Reset**. Instead of a single start node, there are as many as n possible start nodes, each corresponding to a different output sequence of h. Note that many distinct output sequences may be associated with the same final node in G.

The new algorithm **Learn-with-HS** maintains several copies of *map* and *path*, one for each output sequence of h. Thus, graph map_c denotes the copy of the map associated with output sequence c. Initially, Lewis and Clark are at the same node. Whenever algorithm **Learn-with-Reset** would use a reset, **Learn-with-HS** executes the homing sequence h. If the output of h is c, the algorithm learns a new edge in map_c as if it had been reset to u_0 in map_c (see Figure 3.2). After each execution of h, the algorithm learns a new edge in some map_c. Since there are at most n copies, each with dn edges to learn, eventually one map will be completed. Recall that a homing sequence is a lead-lag sequence. Therefore, at the beginning and end of every homing sequence the two robots are together.

```
Learn-with-HS(h):
done := FALSE
while not done
    do execute h; c := the output sequence produced              { instead of a reset }
       if map_c is undefined
          then map_c := ({u_0}, ∅) { map_c = graph consisting of node u_0 and no edges }
               path_c := empty path         { path_c is the null sequence of edge labels }
               k_c := 1                        { k_c counts the number of nodes in map }
       Learn-Edge(map_c,path_c,k_c)
       if map_c has no unfinished nodes
          then done := TRUE
return map_c
```

Theorem 7 *If* **Learn-with-HS** *is called with a homing sequence h as input, it halts within $O(d^2n^4 + dn^2|h|)$ steps and outputs a graph isomorphic to G.*

Proof: The algorithm **Learn-with-HS** maintains at most n running versions of **Learn-with-Reset**, one for each output of the homing sequence. In particular, whenever the two robots execute a homing sequence and obtain an output c, they have identified their position as the start node u_0 in map_c, and can learn one new edge in map_c before executing another homing sequence.

Eventually, one of the versions halts and outputs a complete map_c. Therefore, the correctness of **Learn-with-HS** follows directly from Theorem 5 and the definition of a two-robot homing sequence.

Let $r = O(d^2n^3)$ be the number of steps taken by **Learn-with-Reset**. Since there are at most n start nodes, **Learn-with-HS** takes at most $nr + dn^2|h|$ steps. □

3.5 Learning a Homing Sequence

Unlike a reset function, a two-robot homing sequence can be learned. The algorithm **Learn-Graph** maintains a candidate homing sequence h and improves h as it learns G.

Definition: Candidate homing sequence h is called a *bad* homing sequence if there exist nodes u, v, $u \neq v$, such that $output(h, u) = output(h, v)$, but $final_G(h, u) \not\equiv final_G(h, v)$.

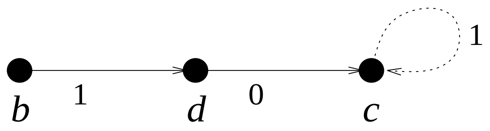
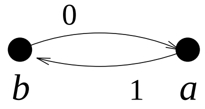
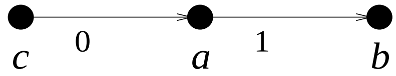
output of homing sequence	starting node of *map*	map
$TTST$	b	
$STST$	b	
$STTT$	c	

Figure 3.2: A possible "snapshot" of the learners' knowledge during an execution of **Learn-with-HS**. The robots are learning the graph G from Figure 3-1 using the two-robot homing sequence h from Figure 3-1. (Node names in maps are *not* known to the learner, but are added for clarity.) The following example demonstrates how the robots learn a new edge using **Learn-with-HS**. Suppose that the robots execute h and see output $TTST$. Then the robots are together at node b. Lewis follows path $1, 0$ to unfinished node c and then crosses the edge labeled 1. Now Clark follows path $1, 0, 1$. Since Clark sees Lewis after 2 steps, the dotted edge is added to map_{TTST}. Next, the robots execute h again and see output $STST$. Thus, they go on to learn some edge in map_{STST}.

Definition: Let a be a node in G. We say that map_c with start node u_0 is a *good representation* of $\langle a, G\rangle$ iff there exists an isomorphism f from the nodes in $map_c = (V^c, E^c)$ to the nodes in a subgraph $G' = (V', E')$ of G, such that $f(u_0) = a$, and

$$\forall u_i, u_j \in V^c, \langle u_i, \ell, u_j\rangle \in E^c \Longleftrightarrow \langle f(u_i), \ell, f(u_j)\rangle \in E'.$$

In algorithms **Learn-with-Reset** and **Learn-with-HS**, the graphs *map* and map_c are *always* good representations of G. In **Learn-Graph** if the candidate homing sequence h is bad, a particular map_c may not be a good representation of G. However, the algorithm *can* test for such maps. Whenever a map_c is shown to be in error, h is improved and all maps are discarded. By the proof of Theorem 6, we know that a candidate homing sequence must be improved at most $n-1$ times. In Section 3.5.1 we explain how to use adaptive homing sequences to discard only one map per improvement.

We now define a test that with probability at least $1/n$ detects an error in map_c if one exists.

Definition: Let $path_c$ be a path such that a robot starting at u_0 and following $path_c$ traverses every edge in $map_c = (V^c, E^c)$. Let $u_0 \ldots u_m$ be the nodes in V^c numbered in order of their first appearance in $path_c$. If both robots are at u_0 then $test_c(u_i)$ denotes the following lead-lag sequence of actions: (1) Lewis follows $path_c$ to the first occurrence of u_i; (2) Clark follows $path_c$ to the first occurrence of u_i; (3) Lewis follows $path_c$ to the end; (4) Clark follows $path_c$ to the end.

Definition: Given map_c and any lead-lag sequence t of actions, define *expected*(t, map_c) to be the expected output if map_c is correct and if both robots start at node u_0 and execute sequence t. We abbreviate *expected*($test_c(u_i)$,map_c) by *expected*($test_c(u_i)$).

Lemma 10 *Suppose Lewis and Clark are both at some node a in G. Let* path_c *be a path such that a robot starting at u_0 and following* path_c *traverses every edge in* map_c*. Then* map_c *is a good representation of $\langle a, G\rangle$ iff* $\forall u_i \in V^c$, $\mathrm{output}(\mathrm{test}_c(u_i)) = \mathrm{expected}(\mathrm{test}_c(u_i))$.

Proof:

($\Longrightarrow$): By definition of good representation and *expected*($\mathrm{test}_c(u_i)$).

($\Longleftarrow$): Suppose that all tests produce the expected output. We define a function f as follows: Let $f(u_0) = a$. Let $p(u_i)$ be the prefix of $path_c$ up to the first occurrence of u_i. Define $f(u_i)$ to be the node in G that a robot reaches if it starts at a and follows $p(u_i)$. Let $G' = (V', E')$ be the image of $f(map_c)$ on (V, E).

We first show that f is an isomorphism from V^c to V'. By definition of G', f must be surjective. To see that f is injective, assume the contrary. Then there exist two nodes $u_i, u_j \in V^c$ such that $i \neq j$ but $f(u_i) = f(u_j)$. But then *output*($\mathrm{test}_c(u_i)$) $\neq$ *expected*($\mathrm{test}_c(u_i)$), which contradicts our assumption that all tests succeed. Next,

```
Learn-Graph():
done := FALSE
h := Λ (empty sequence)
while not done
    do execute h; c := the output sequence produced                    { instead of a reset }
        if map_c is undefined
            then map_c := ({u_0}, ∅)        { map_c is the graph consisting of node u_0 and no edges }
                path_c := empty path                    { path_c is the null sequence of edge labels }
                k_c := 1                                 { k_c counts the number of nodes in map }
        if map_c has no unfinished node map-reachable from final_M(path_c)
            then Lewis and Clark move to final_M(path_c)
                comp := maximal strongly-connected component in map_c containing final_M(path_c)
                h-improve := Verify(final_M(path_c),comp)
                if h-improve = Λ
                    then done := TRUE                                    { map_c is complete }
                    else                                     { h-improve ≠ Λ. error detected }
                        append h-improve to end of h             { improve homing sequence ...}
                        discard all maps and paths      { ...and start learning maps from scratch }
            else v := value of a fair 0/1 coin flip              { learn edges or test for errors? }
                if v = 0                                                     { test for errors }
                    then u_i := a random node in map_c              { randomly pick node to test }
                        h-improve := Test(map_c, path_c, i)
                        if h-improve ≠ Λ                                      { error detected }
                            then append h-improve to end of h    { improve homing sequence ...}
                                discard all maps and paths   { ...start learning maps from scratch }
                    else Learn-Edge(map_c,path_c,k_c)
return map_c
```

```
Test(map_c, path_c, i):   {u_0, u_1, ..., u_k = the nodes in map_c indexed by first appearance in path_c }
h-improve := the following sequence of actions:
    Lewis follows path_c to the first occurrence of u_i in path_c
    Clark follows path_c to the first occurrence of u_i in path_c
    Lewis follows path_c to the end
    Clark follows path_c to the end
if output(h-improve) ≠ expected-output(h-improve)                              { if error detected }
    then return h-improve                                                     { return test_c(u_i) }
    else return Λ                                                        { return empty sequence }
```

```
Verify(v_0, map):             { v_0, v_1, ..., v_k are the nodes in map ordered by first appearance in p}
path := path such that a robot starting at v_0 in map and following path visits all nodes
in map and returns to v_0
for each i, 0 ≤ i < k
    do h-improve := Test(map, path, i)
        if h-improve ≠ Λ
            then return h-improve
return Λ
```

we show that $\langle u_i, \ell, u_j\rangle \in V^c \iff \langle f(u_i), \ell, f(u_j)\rangle \in V'$, proving that map_c is a good representation of $\langle a, G\rangle$.

($\Longleftarrow$): By definition of G', the image of map_c.

($\Longrightarrow$): Inductively assume that $\langle u_i, \ell, u_j\rangle \in V^c \iff \langle f(u_i), \ell, f(u_j)\rangle \in V'$ for the first m edges in $path_c$, and suppose that this prefix of the path visits only nodes $u_0 \dots u_i$. Now consider the $(m+1)$st edge $e = \langle a, \ell, b\rangle$. There are two possibilities. In one case, edge e leads to some new node u_{i+1}. Then by definition $f(u_{i+1})$ is b's image in G, so $\langle f(a), \ell, f(b)\rangle \in G'$. Otherwise e leads to some previously-seen node u_{i-k}. Suppose that $f(u_{i-k})$ is *not* the node reached in G by starting at u_0 and following the first $m+1$ edges in $path_c$. Then *output(test*$_c(u_{i-k})$*)* $\neq$ *expected(test*$_c(u_{i-k})$*)*, so *test*$_c(u_{i-k})$ fails, and we arrive at a contradiction. Therefore $f(b) = f(u_{i-k})$ and $\langle f(a), \ell, f(b)\rangle \in G'$. $\square$

Corollary 8 *Suppose Lewis and Clark are together at u_0 in* map$_c$*. Let* map$_c$ *be strongly connected and have n nodes, $u_0, \dots u_{n-1}$. Then the two robots can verify whether* map$_c$ *is a good representation of $\langle Loc_G(Lewis), G\rangle$ in $O(n^3)$ steps.*

Proof: Since map_c is strongly connected, there exists a path $path_c$ with the following property: a robot starting at u_0 and following $path_c$ visits all nodes in map_c and returns to u_0. Index the remaining nodes in order of their first appearance in $path_c$. The two robots verify whether, for all u_i in order, *output(test*$_c(u_i)$*)* $=$ *expected(test*$_c(u_i)$*)*. Note that Lewis and Clark are together at u_0 after each test. By Lemma 10, this procedure verifies map_c. Since $path_c$ has length $O(n^2)$, each test has length $O(n^2)$, so verification requires $O(n^3)$ steps. $\square$

In **Learn-Graph** after the robots execute a homing sequence, they randomly decide either to learn a new edge or to test a random node in map_c. The following lemma shows that a test that failed can be used to improve the homing sequence.

Lemma 11 *Let h be a candidate homing sequence in* **Learn-Graph***, and let u_k be a node such that* output(test$_c(u_k)$) $\neq$ expected(test$_c(u_k)$)*. Then there are two nodes a, b in G that h does not distinguish but that $h \circ$* test$_c(u_k)$ *does.*

Proof: Let a be a node in G such that when both robots start at a, *output(test*$_c(u_k)$*)* $\neq$ *expected(test*$_c(u_k)$*)*. Suppose that at step i in $test_c(u_k)$, the expected output is T (respectively S), but the actual output is S (resp. T). Each edge in $path_c$ and map_c was learned using **Learn-Edge**. If map_c indicates that the ith node in $path_c$ is u_k, there must be a start node b in G where u_k really is the ith node in $path_c$. Since *output(test*$_c(u_k)$*)* $\neq$ *expected(test*$_c(u_k)$*)*, the sequence $h \circ test_c(u_k)$ distinguishes a from b. □

The algorithm **Learn-Graph** runs until there are no more map-reachable unexplored nodes in some map_c. If map_c is not strongly connected, then it is not a good representation of G. In this case, the representation of the last strongly-connected component on *path* must be incorrect. Thus, calling **Verify** on this component from the last node on *path* returns a sequence that improves h. If map_c is strongly connected, then either **Verify** returns an improvement to the homing sequence, or map_c is a good representation of G.

Before we can prove the correctness of our algorithm, we need one more set of tools. Consider the following statement of Chernoff bounds from Raghavan [91].

Lemma 12 *Let $X_1, \ldots, X_m$ be independent Bernoulli trials with $E[X_j] = p_j$. Let the random variable $X = \sum_{j=1}^{m} X_j$, where $\mu = E[X] \geq 0$. Then for $\delta > 0$,*

$$Pr[X > (1+\delta)\,\mu] < \left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right]^{\mu},$$

and

$$Pr[X < (1-\delta)\,\mu] < e^{-\mu\delta^2/2}.$$

In our analysis in this section and in Section 3.6, the random variables may not be independent. However, the following corollary bounds the conditional probabilities. The proof of this corollary is exactly analogous to that of a similar corollary by Aumann and Rabin [14, Corollary 1].

Corollary 9 *Let $X_1, \ldots, X_m$ be 0/1 random variables (not necessarily independent), and let $b_j \in \{0, 1\}$ for $1 \leq j \leq m$. Let the random variable $X = \sum_{j=1}^{m} X_j$. For any*

$b_1, \ldots, b_{j-1}$ and $\delta > 0$, if $Pr[X_j = 1 | X_1 = b_1, X_2 = b_2, \ldots, X_{j-1} = b_{j-1}] \leq p_j$ and $\mu = \sum_{j=1}^{m} p_j > 0$, then

$$Pr[X > (1+\delta)\,\mu] < \left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right]^{\mu},$$

and for any $b_1, \ldots, b_{j-1}$ and $\delta > 0$, if $Pr[X_j = 1 | X_1 = b_1, X_2 = b_2, \ldots, X_{j-1} = b_{j-1}] \geq p_j$ and $\mu = \sum_{j=1}^{m} p_j > 0$, then

$$Pr[X < (1-\delta)\,\mu] < e^{-\mu\delta^2/2}.$$

Theorem 10 *The algorithm* **Learn-Graph** *always outputs a map isomorphic to G and halts in $O(d^2n^6)$ steps with overwhelming probability ($1 - e^{-cn}$, where constant $c > 0$ can be chosen as needed).*

Proof: Since **Learn-Graph** verifies map_c before finishing, if the algorithm terminates then by Corollary 8 it outputs a map isomorphic to G. It is therefore only necessary to show that the algorithm runs in $O(d^2n^6)$ steps with overwhelming probability.

In each iteration of the while loop in **Learn-Graph**, if there are no map-reachable unfinished nodes, then the algorithm attempts to verify the map. Otherwise, the algorithm decides randomly whether to learn a new edge or to test a random node in the graph. It follows from Lemma 12 that a constant fraction of the random decisions are for learning and a constant fraction are for testing.

By Theorem 5 the total number of steps spent learning edges in each version of *map* is $O(d^2n^3)$. For each candidate homing sequence, there are n versions of *map*, and the candidate homing sequence is improved at most n times. Thus, $O(d^2n^5)$ steps are spent learning nodes and edges.

We consider the number of steps taken testing nodes. Each test requires $|path| = O(dn^2)$ steps. Once a map contains an error, the probability that the robots choose to test a node that is in error is at least $1/n$. A map with more than dn edges must be faulty. Note that the candidate homing sequence is improved at most $n-1$ times. Thus by Corollary 9, with overwhelming probability after $O(n^2)$ tests of maps with at least dn nodes, the candidate sequence h is a homing sequence. Overall, the algorithm has to learn $O(dn^3)$ edges, and therefore it executes $O(dn^3)$ tests. Thus the total number of steps spent testing is $O(d^2n^5)$.

After each test or verification, the algorithm executes a candidate homing sequence. Since there are $O(dn)$ edges in each map, candidate homing sequences are executed $O(dn^3)$ times. Each improvement of the candidate homing sequence extends its length by $|path|$, so the time spent executing homing sequences is $O(d^2n^6)$. Thus, the total running time of the algorithm is $O(d^2n^6)$. □

3.5.1 Improvements to the Algorithm

The running time for **Learn-Graph** can be decreased significantly by using two-robot *adaptive* homing sequences. As in Rivest and Schapire [93], an *adaptive homing sequence* is a decision tree, so the actions in later steps of the sequence depend on the output of earlier steps. With an adaptive homing sequence, only one map_c needs to be discarded each time the homing sequence is improved. Thus the running time of **Learn-Graph** decreases by a factor of n to $O(d^2n^5)$.

Any additional information that distinguishes nodes can be included in the output, so homing sequences can be shortened even more. For example, a robot learning an unfamiliar city could easily count the number of roads leading into and out of intersections. It might also recognize stop signs, traffic lights, railroad tracks, or other common landmarks. Therefore, in any practical application of this algorithm we expect a significantly lower running time than the $O(d^2n^5)$ bound suggests.

Graphs with high conductance can be learned even faster using the algorithm presented in Section 3.6.

3.5.2 Limitations of a Single Robot with Pebbles

We now compare the computational power of two robots to that of one robot with a constant number of pebbles. Note that although **Learn-Graph** runs in time polynomial in n, the algorithm requires no prior knowledge of n. We argue here that a single robot with a constant number of pebbles cannot efficiently learn strongly-connected directed graphs without prior knowledge of n.

As a tool we introduce a family $\mathcal{C} = \cup_n \mathcal{C}_n$ of graphs called *combination locks.* [1] For a graph $C = (V, E)$ in $\mathcal{C}_n$ (the class of n-node combination locks), $V = \{u_0, u_1, \ldots, u_{n-1}\}$ and either $\langle u_i, 0, u_{i+1\, mod\, n}\rangle$ and $\langle u_i, 1, u_0\rangle \in E$, or $\langle u_i, 1, u_{i+1\, mod\, n}\rangle$ and $\langle u_i, 0, u_0\rangle \in E$, for all $i \leq n$ (see Figure 3.3a). In order for a robot to "pick a lock" in $\mathcal{C}_n$ — that is, to reach node u_{n-1} — it must follow the unique n-node simple path from u_0 to u_{n-1}. Thus any algorithm for a single robot with *no* pebbles can expect to take $\Theta(2^n)$ steps to pick a random combination lock in $\mathcal{C}_n$.

We construct a restricted family $\mathcal{R}$ of graphs and consider algorithms for a single robot with a single pebble. For all positive integers n, the class $\mathcal{R}_n$ contains all graphs consisting of a directed ring of $n/2$ nodes with an $n/2$-node combination lock inserted into the ring (as in Figure 3.3b). Let $\mathcal{R} = \cup_{n=1}^{\infty} \mathcal{R}_n$. We claim that there is no probabilistic algorithm for one robot and one pebble that learns arbitrary graphs in $\mathcal{R}$ in polynomial time with high probability.

To see the claim, consider a single robot in node u_0 of a random graph in $\mathcal{R}$. Until the robot drops its pebble for the first time it has no information about the graph. Furthermore, with high probability the robot needs to take $\Theta(2^n)$ steps to emerge from a randomly-chosen n-node combination lock unless it drops a pebble in the lock. But since the size of the graph is unknown, the robot always risks dropping the pebble before entering the lock. If the pebble is dropped outside the lock, the robot will not see the pebble again until it has passed through the lock. A robot that cannot find its pebble

[1] Graphs of this sort have been used in theoretical computer science for many years (see [82], for example). More recently they have reemerged as tools to prove the hardness of learning problems. We are not sure who first coined the term "combination lock."

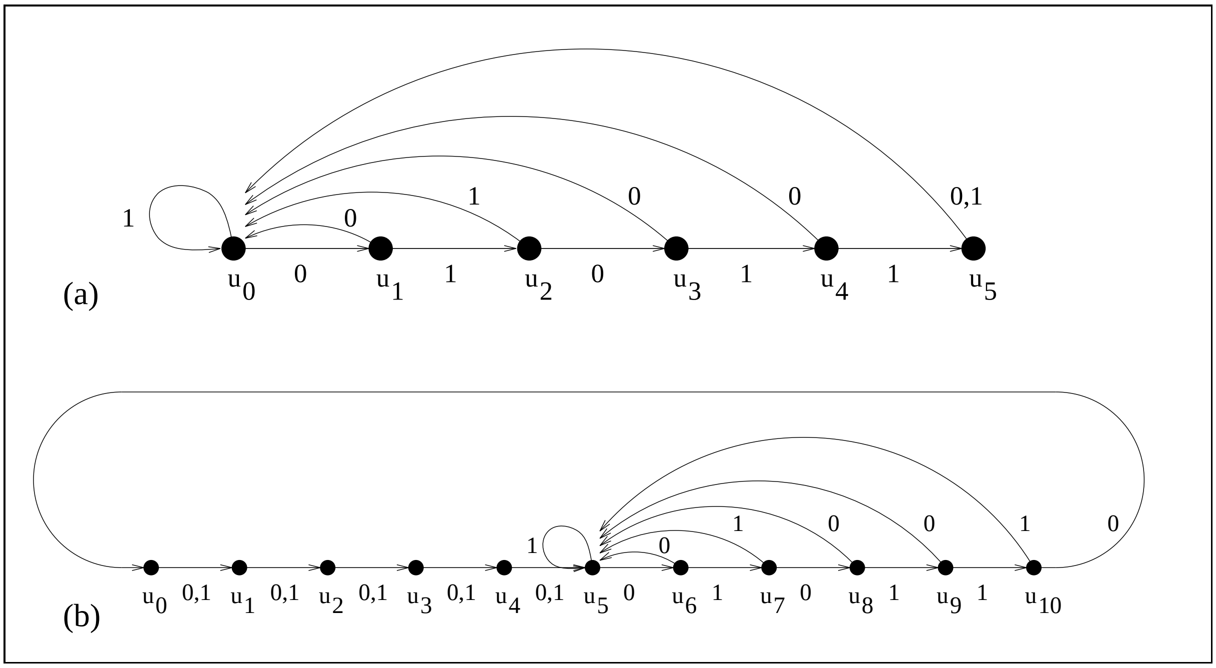


Figure 3.3: (a) A combination-lock, whose combination is $0, 1, 0, 1, 1$. (b) A graph in $\mathcal{R}_{11}$. Graphs in $\mathcal{R} = \cup_{n=1}^{\infty} \mathcal{R}_n$ cannot be learned by one robot with a constant number of pebbles.

has no way of marking nodes and cannot learn.

More formally, suppose that there were some probabilistic algorithm for one robot and a pebble to learn random graphs in $\mathcal{R}$ in polynomial time with probability greater than $1/2$. Then there must be some constant c such that the probability that the robot drops its pebble in its first c steps is greater than $1/2$. (Otherwise, the probability that the algorithm *fails* to learn in time polynomial in n is greater than $1/2$.) Therefore, the probability that the robot loses its pebble and fails to learn a random graph in $\mathcal{R}_{2c}$ efficiently is at least $1/2$.

A similar argument holds for a robot with a constant number of pebbles. We conjecture that even if the algorithm is given n as input, a single robot with a constant number of pebbles cannot learn strongly-connected directed graphs. However, using techniques similar to those in Section 3.6, one robot with a constant number of pebbles and prior knowledge of n *can* learn high-conductance graphs in polynomial time with high probability.

3.6 Learning High Conductance Graphs

For graphs with good expansion, learning by walking randomly is more efficient than learning by using homing sequences. In this section we define *conductance* and present an algorithm that runs more quickly than **Learn-Graph** for graphs with conductance greater than $\sqrt{\log n/dn^2}$.

3.6.1 Conductance

The conductance [100] of a graph characterizes the rate at which a random walk on the graph converges to the stationary distribution π. For a given directed graph $G = (V, E)$, consider a weighted graph $G' = (V, E, W)$ with the same vertices and edges as G, but with edge weights defined as follows. Let $M = \{m_{i,j}\}$ be the transition matrix of a random walk that leaves i by each outgoing edge with probability $1/(2 \cdot \text{degree}(i))$ and remains at node i with probability $1/2$. Let P^0 be an initial distribution on the n nodes in G, and let $P^t = P^0M^t$ be the distribution after t steps of the walk defined by M. (Note that π is a *steady state* distribution if for every node i, $P_i^t = \pi_i \longrightarrow P_i^{t+1} = \pi_i$. For irreducible and aperiodic Markov chains, π exists and is unique.) Then the edge weight $w_{i,j} = \pi_i m_{i,j}$ is proportional to the steady state probability of traversing the edge from i to j. Note that the total weight entering a node is equal to the total weight leaving it; that is, $\sum_j w_{i,j} = \sum_j w_{j,i}$.

Consider a set $S \subseteq V$ which defines a cut $(S, \overline{S})$. For sets of nodes S and T, let $W_{S,T} = \sum_{s \in S, t \in T} w_{s,t}$. We denote $W_{S,V}$ by W_S, so W_V represents the total weight of all edges in the graph. Then the conductance of S is defined as $\phi_S = W_{S,\overline{S}} / \sum_{i \in S} \pi_i = W_{S,\overline{S}}/W_S$.

The conductance of a graph is the least conductance over all cuts whose total weight is at most $W_V/2$: $\phi(G) = \min_S \{\max(\phi_S, \phi_{\overline{S}})\}$. The conductance of a directed graph can be exponentially small.

Mihail [81] shows that after a walk of length $\phi^{-2}\log(2n/\epsilon^2)$, the L_1 norm of the distance between the current distribution P and the stationary distribution π is at most ϵ (i.e. $\sum_i |P_i - \pi_i| \leq \epsilon$). In the rest of this section, a choice of $\epsilon = 1/n^2$ is sufficient, so a random walk of length $\phi^{-2}\log(2n^5)$ is used to approximate the stationary distribution. We call $T = \phi^{-2}\log(2n^5)$ the *approximate mixing time* of a random walk on an n-node graph with conductance ϕ.

3.6.2 An Algorithm for High Conductance Graphs

If a graph has high conductance it can be learned more quickly. In a high-conductance graph, we can estimate the steady state probability of node i by leaving Clark at node i while Lewis takes w random walks of $\phi^{-2}\log(2n^5)$ steps each. Let x be the number of times that Lewis sees Clark at the last step of a walk. If w is large enough, x/w is a good approximation to π_i.

Definitions: Call a node i a *likely* node if $\pi_i \geq 1/2n + 1/n^2$. Note that every graph must have at least one such node. (The $1/n^2$ term appears because of the distance $\epsilon = 1/n^2$ from the stationary distribution. Its inclusion here simplifies the analysis later.) A node that is not likely is called *unlikely*.

Algorithm **Learn-Graph2** uses this estimation technique to find a likely node u_0 and then calls the procedure **Build-Map** to construct a map of G starting from u_0. The procedure **Build-Map** learns at least one new edge each iteration by sending Lewis across an unexplored edge $\langle u, \ell, v\rangle$ of some unfinished node u in *map*. Clark waits at start node u_0 while Lewis walks randomly until he meets Clark. (If u_0 is a likely node, this walk is expected to take $O(Tn)$ steps.) Lewis stores this random walk in the variable *path*. Thus, $path_i$ is the label of the edge traversed at the ith step of the random walk, $path[i \ldots j]$ represents edges $path_i$ to $path_j$, and $|path|$ represents the length of path. We say that *path-step*(Lewis) $= i$ if Lewis has just crossed the ith edge on *path*.

```
Learn-Graph2(w, B, M, T):
done := FALSE
T := $\phi^{-2} \log(2n^5)$                                   { the mixing time }
while (not (done))
    do map := ($\{u_0\}, \emptyset$)        { map is the graph consisting of node $u_0$ and no edges }
        lost := FALSE
        Lewis and Clark together take a random walk of length T
        Lewis takes w random walks of length T    { approx. stationary prob. of $Loc_G$(Clark) }
        x := number of walks where Lewis and Clark are together at the last step
        path := the path Lewis followed since leaving Clark
        if $x/w \leq B$                             { bound $B < 1/n$ chosen for ease of proof}
            then Clark follows path to catch up to Lewis     { not at a frequently-visited node }
            else Lewis moves randomly until he sees Clark    { call node where they meet $u_0$ }
                done := Build-Map(map, M, T)
return map
```

```
Build-Map(map, M, T):
while there are unfinished nodes in map and not lost          { Lewis and Clark both at $u_0$ }
    do $u_i$ := an unfinished node in map
        restart := a minimal path in map from $u_0$ to $u_i$
        m := largest index of the nodes in map
        path := empty path
        Lewis follows restart and traverses unexplored edge $\ell$      { Lewis crosses a new edge }
        while length(path) < MT and robots are not together    { Lewis walks randomly ...}
            do Lewis traverses a random edge $\ell'$ and adds $\ell'$ to end of path
        if robots are together                                 { ...until he sees Clark at $u_0$ }
            then both robots follow restart to $u_i$ and cross edge $\ell$
                path := Compress-Path(path, restart)                 { removes loops from path }
                path, $u_j$ := Truncate-Path-At-Map(path, map)     { shortest path back to map }
                if |path| = 0
                    then add edge $\langle u_i, \ell, u_j \rangle$ to map
                    else add nodes $u_{m+1}, \ldots, u_{m+|path|}$ to map
                        add edges $\langle u_i, \ell, u_{m+1} \rangle$ and $\langle u_{m+|path|}, path_{|path|}, u_j \rangle$ to map
                        $\forall k, 1 \leq k < |path|$ add edges $\langle u_{m+k}, path_k, u_{m+k+1} \rangle$
                both robots move to $u_0$
            else                              { if Lewis walks MT steps without seeing Clark }
                Clark follows restart, $\ell$, path to catch up to Lewis
                lost := TRUE
if lost
    then return FALSE
    else return TRUE
```

```
Compress-Path (path,restart):
while Clark not at end of path                              { Lewis and Clark both at u_0 }
    do while Lewis not at end of path
            do Lewis traverses the next edge of path
                if Lewis and Clark are together       { found a loop in path — remove it }
                    then path :=
                        path[1 ... path-step(Clark) ∘ path[path-step(Lewis) + 1 ... |path|]
        Lewis follows restart and edge ℓ
        Lewis traverses edges path[1 ... path-step(Clark) ]
        both robots are now together and traverse one edge of path
return path                                                 { Lewis and Clark both at u_0 }


Truncate-Path-At-Map(path,map ):
earliest := |path|                  { first position on path that is a node already in map }
earliest-node := u_0                                                { the name of this node }
for each node u_k in map
    Clark moves to u_k
    Lewis follows restart and edge ℓ
    while Lewis not at end of path
        do if Lewis and Clark are together and path-step(Lewis) < earliest
                then earliest := path-step(Lewis)
                    earliest-node := u_k
            Lewis traverses next edge on path
both robots move to u_0
return path [1 ...earliest ], earliest-node
```

The procedure **Compress-Path** returns the shortest subpath of *path* that connects v to u_0. Finally, **Truncate-Path-at-Map** compares nodes on the path with all nodes in *map* and returns the shortest subpath connecting v to some node in *map*. By adding the final path to the map, **Build-Map** connects the new node v to *map*, so *map* always represents a strongly connected subgraph of G. Figure 3.4 illustrates a single iteration of the main loop in **Build-Map**.

Algorithm **Learn-Graph2** takes as input parameters the number of random walks w, a bound B to separate the probability of likely and unlikely nodes (we choose B to be approximately $3/4n$), the mixing time T, and a quantity M. This quantity is chosen so that the probability of a robot's starting at a likely node and walking randomly for MT steps without returning to its start node is very small.

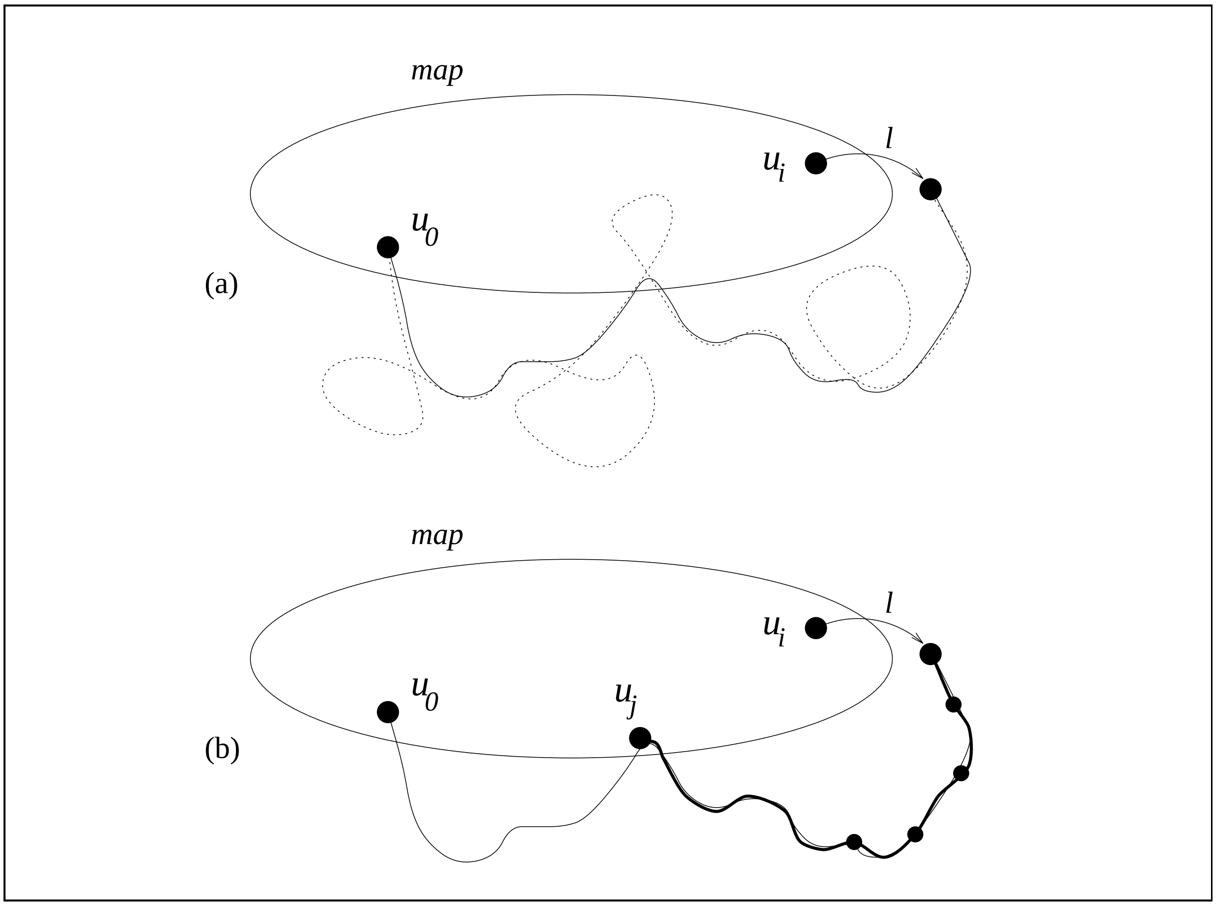


Figure 3.4: Procedure **Build Map** during one execution of the **while** loop. The ovals represent *map*, the portion of graph G learned so far. Note that *map* is strongly connected. Node u_0, the first node added to *map*, is with high probability a node with a large stationary probability (a *likely node*). The robots find u_0 in procedure **Learn-Graph2** using random walks in line 2 of **Build Map**. The robots agree on a node u_i with unexplored outgoing edges (an *unfinished node*). Then Lewis moves to u_i and follows the unexplored edge ℓ, while Clark stays at u_0. Since ℓ is unexplored, Lewis is now at an unknown node. Lewis walks randomly until, visiting u_0, he finds Clark. The dotted line of Figure 4 [a] depicts this random walk, denoted *path*. Random walk *path* may pass through the same node many times. In procedure **Compress-Path**, the robots collectively remove all of the loops from the path (reducing the path to the solid line in [a] and [b]). In procedure **Truncate-Path-at-Map**, the robots find u_j, the first node in *path* already in *map*. All the nodes and edges of *path* until u_j (the bold line in [b]) are added to *map*.

In sections 3.6.3 and 3.6.4 we prove the following theorems.

Theorem 11 *When* **Learn-Graph2** *halts, it outputs a graph isomorphic to G.*

Theorem 12 *Suppose* **Learn-Graph2** *is run on a graph G with $w = \sqrt{(4+\tau)dn^3}$ and $M = (4+\tau)dn^2$ for some constant $\tau > 0$, and $B = \frac{3}{4n}\left(1 + \frac{2}{n}\right)$. Then for sufficiently large n, with probability at least $1-\epsilon$* **Learn-Graph2** *halts within $O((4+\tau)dn^3T)$ steps, where $\epsilon = e^{-\frac{1}{20}\sqrt{(4+\tau)nd}} + e^{-\frac{1}{2}\sqrt{(4+\tau)nd}} + e^{-\frac{dn\tau}{4}}$.*

3.6.3 Correctness of Learn-Graph2

In this section, we prove the correctness of each procedure in **Learn-Graph2**.

Lemma 13 *The procedure* **Compress-Path** *halts in $O(n\,|\mathrm{path}|)$ steps and returns a path in which no node occurs more than once.*

Proof: We prove the following invariant by induction: in **Compress-Path**, whenever Lewis reaches the end of *path*, each node in *path*$[1 \ldots$ *path-step*(Clark)$]$ appears at most once in the entire *path*.

Assume that this claim holds after Clark has crossed the first k edges in *path*. By the inductive hypothesis, we know that when Clark crosses the $(k+1)$st edge, he arrives at some new node not previously encountered along *path*. Now Lewis follows the entire path. Whenever the path loops back to $\mathrm{Loc}_G(\mathrm{Clark})$, the loop is removed from the path. Thus, all repeated occurrences of the new node are removed from the path, proving the inductive step.

Since there are n nodes in the graph, Clark can only make n moves before he returns to u_0. Lewis can move at most $|path|$ steps for every move of Clark's, so the total running time is $O(n\,|path|)$. □

Lemma 14 *The procedure* **Truncate-Path-At-Map** *finds the index of the first path step leading to a node already in* map. *The algorithm runs in $O(n^2)$ steps.*

Proof: For each node u_k in *map*, Lewis traverses the path once while Clark waits at u_k. The procedure keeps track of the earliest node found that is already in *map*, so the procedure's correctness follows. Clark takes at most n steps to reach each node u_k. Lewis needs at most n steps to follow the compressed *path* and n more to return to the start of the path. Thus the algorithm requires no more than $3n^2$ steps. □

The algorithm **Learn-Graph2** halts only when **Build-Map** returns TRUE. The following lemma shows that whenever **Build-Map** returns TRUE, *map* is isomorphic to G.

Lemma 15 *In* **Build-Map***, whenever Clark is at* u_0*,* map *is a good representation of* $\langle Loc_G(Clark), G\rangle$.

Proof: We inductively build a subgraph $G' = (V', E')$ of $G = (V, E)$ and construct an isomorphism f from *map* to G'. Initially, Lewis and Clark are both at u_0 and *map* consists of the single node u_0 and no edges. Let $V' = Loc_G(\text{Clark}), E' = \emptyset$, and $f(u_0) = Loc_G(\text{Clark})$. Then *map* is a good representation of $\langle Loc_G(\text{Clark}),\text{G}\rangle$.

Consider the robots starting an iteration of the first **while** loop in **Build-Map**. Both robots are together at u_0. Inductively assume that *map* is a good representation of $\langle Loc_G(\text{Clark}),\text{G}\rangle$ and that *map* is strongly-connected. Thus, Lewis can always reach an unfinished node in *map* if one exists. Lewis walks to an unfinished node u_i, crosses a new edge ℓ to an unknown node v, and then walks randomly until he returns to u_0, where Clark is waiting. From Lemmas 13 and 14, after the algorithm executes subroutines **Compress-Path** and **Truncate-Path**, $\ell \circ path$ is a path that begins at u_i, crosses edge ℓ, ends at u_j, and whose intermediate nodes are not represented in *map*.

If *path* is empty after **Truncate-Path-At-Map**, then v is node u_j already in *map*. Adding edge $\langle f(u_i), \ell, f(u_j)\rangle$ to E' and $\langle u_i, \ell, u_j\rangle$ to *map* and therefore maintains the invariant that G' is a subgraph of G and preserves the isomorphism between *map* and G'.

If *path* is not empty, then by lemmas 13 and 14 all nodes reached by starting at u_i and following *path* to the end are distinct, and only the last node reached is already in *map*. Let m be the highest index so far of any node in *map*. The algorithm adds new nodes $u_{m+1}, \ldots, u_{m+|path|}$ and new edges $\langle u_{m+k}, path_k, u_{m+k+1}\rangle$ $\forall k, 1 \leq k < |path|$, $\langle u_i, \ell, u_{m+1}\rangle$, and $\langle u_{m+|path|}, path_{|path|}, u_j\rangle$ to *map*. Let $f(u_{m+k})$ be the location of Lewis in G after Lewis has crossed the kth edge of the path. Add the $|path| - 1$ new nodes to V', and edges $\langle f(u_{m+k}), path_k, f(u_{m+k+1})\rangle$ to E'. Then f is an isomorphism from *map* to $G' \subseteq G$, so *map* is a good representation of $\langle Loc_G(\text{Clark}),\text{G}\rangle$. Since *path* connects an unfinished node to another node in *map*, *map* remains strongly-connected. □

When **Build-Map** halts and returns TRUE, there are no unfinished nodes in *map* . Since *map* is isomorphic to $G' \subseteq G$ and has no unfinished nodes, *map* must have the same number of nodes and edges as G. Therefore, *map* is isomorphic to G when **Build-Map** returns TRUE, proving Theorem 11. □

3.6.4 Running Time and Failure Probability of Learn-Graph2

We proved that when the algorithm terminates it is correct. In this section, we prove Theorem 12 by analyzing the probability that the algorithm terminates in a reasonable amount of time. We say the algorithm fails if any of the following cases holds:

1. Algorithm **Learn-Graph2** finds an unlikely node but estimates that it is a likely node.

2. Algorithm **Learn-Graph2** fails to find a likely node in the allotted time. We allow $w = \sqrt{(4+\tau)dn^3}$ iterations, each consisting of w random walks.

3. Algorithm **Learn-Graph2** calls **Build-Map** from a likely node, but **Build-Map** returns FALSE.

In fact, these conditions overestimate the probability that the algorithm fails to run in $O((4+\tau)dn^3T)$ steps. The next three lemmas bound the probabilities of each of the three failure conditions. At the end of the section, we analyze the running time of **Learn-Graph2** when no failure condition occurs.

Lemma 16 (Failure Condition 1) *Suppose that* **Learn-Graph2** *is run with* $w = \sqrt{(4+\tau)dn^3}$ *and* $M = (4+\tau)dn^2$. *Assume that the algorithm estimates node u's steady-state probability to be greater than* $B = \frac{3}{4n}(1+\frac{2}{n})$. *Then the probability that u is not a likely node is at most* $e^{-\frac{1}{20}\sqrt{(4+\tau)nd}}$.

Proof: Call each random walk in **Learn-Graph2** a *phase*. Let X_i be the random variable where

$$X_i = \begin{cases} 1 & \text{if Lewis and Clark are together at the end of phase } i \\ 0 & \text{otherwise.} \end{cases}$$

Then $X = \sum_{i=1}^{w} X_i$ is the number of phases where Lewis and Clark end together. If u is an unlikely node, then $E[X] \leq (w/2n)(1+(2/n))$, because the estimation of π_u could be inaccurate by at most $\epsilon = 1/n^2$. We therefore bound the quantity

$$Pr\left[X > \frac{3w}{4n}\left(1+\frac{2}{n}\right) \;\middle|\; E[X] \leq \frac{w}{2n}\left(1+\frac{2}{n}\right)\right].$$

Using the Chernoff bound from Lemma 9 with $\delta = 1/2$, we get:

$$Pr\left[X \geq \frac{3w}{4n}\left(1+\frac{2}{n}\right)\right] \leq \left[\frac{e^{\frac{1}{2}}}{(\frac{3}{2})^{\frac{3}{2}}}\right]^{\frac{w}{2n}\left(1+\frac{2}{n}\right)}$$

$$\leq \left(\sqrt{\frac{8e}{27}}\right)^{\frac{w}{2n}} \leq e^{\ln\left(\sqrt{\frac{8e}{27}}\right)^{\frac{\sqrt{(4+\tau)dn}}{2}}} \leq e^{\frac{-\sqrt{(4+\tau)dn}}{2}\ln\left(\sqrt{\frac{27}{8e}}\right)} \leq e^{\frac{-\sqrt{(4+\tau)dn}}{20}}.$$

$\square$

Lemma 17 (Failure Condition 2) *The probability that* **Learn-Graph2** *fails to find and recognize a likely node after* $\sqrt{(4+\tau)dn^3}$ *iterations is at most* $e^{-\frac{1}{2}\sqrt{(4+\tau)dn}}$ *for sufficiently large n.*

Proof: Define a *good* node to be a node with steady state probability at least $1/n$. (Note that every graph has at least one good node.) We can bound the probability that we fail to find and recognize a likely node by the probability that we fail to identify a good node within w iterations.

First, we bound the probability that the algorithm fails to recognize a good node when testing one. Random variables X_i and X are defined as in Lemma 16. Then, since the stationary probability of a good node is greater than $1/n$,

$$E(X) \geq w\left(\frac{1}{n} - \frac{1}{n^2}\right).$$

To simplify the math, note that for sufficiently large n,

$$w\left(\frac{1}{n} - \frac{1}{n^2}\right) \geq \frac{15w}{16n}\left(1 + \frac{2}{n}\right).$$

Then by the Chernoff corollary in Lemma 9, for $\delta = 1/5$,

$$Pr\left[X < (1-\delta)\frac{15w}{16n}\right] = Pr\left[X < \frac{3w}{4n}\right] \leq e^{-\frac{1}{2}\frac{15}{16}\frac{1}{25}\frac{w}{n}} \leq e^{-\frac{3}{160}\sqrt{(4+\tau)dn}}.$$

Define γ to be the quantity

$$\left(1 - \frac{1}{n}\right)\left(1 - e^{-\frac{3}{160}\sqrt{(4+\tau)dn^3}}\right).$$

The probability that the node reached at the end of a random walk of both robots is a good node and is identified as such is at least

$$\left(\frac{1}{n} - \frac{1}{n^2}\right)\left(1 - e^{-\frac{3}{160}\sqrt{(4+\tau)dn^3}}\right) = \frac{\gamma}{n}.$$

Therefore the probability that after w trials, no likely node is found and recognized is at most

$$\left(1-\frac{\gamma}{n}\right)^{\sqrt{(4+\tau)dn^3}} = \left(1-\frac{\gamma}{n}\right)^{\frac{-n}{\gamma}\left(-\gamma\sqrt{(4+\tau)dn}\right)} = e^{-\gamma\sqrt{(4+\tau)dn}}.$$

Note that γ approaches 1 as n increases. For sufficiently large $n, \gamma > 1/2$, so the probability that the robots fail to find a likely node after w trials is at most $e^{-\frac{1}{2}\sqrt{(4+\tau)dn}}$.
□

Now we analyze the running time of **Build-Map**. The procedure **Build-Map** executes the main while loop at most once for each of the dn edges in the graph. Let k_i be the length of the random walk in the ith iteration of the while loop. Then let $K = \sum_{i=1}^{dn} k_i$ be the total length of all the random walks in the algorithm. Since by Lemmas 13 and 14 **Compress-Path** runs in $O(nk_i)$ steps and **Truncate-Path-At-Map** runs in $O(n^2)$ steps, the ith iteration takes $O(n + k_i + nk_i + n^2)$ steps. Therefore the total running time of the algorithm is $O(dn^3 + nK)$.

Lemma 18 (Failure Condition 3) *If u_0 is a likely node, then with probability at least $1 - e^{-\frac{\tau dn}{4}}$, $K \leq (4+\tau)dn^2T$.*

Proof: We use an amortized analysis to prove the bound on K. First we subdivide all of Lewis' random walks during **Build-Map** into periods of $T = \phi^{-2}\log 2n/\epsilon^2$ steps each, where T is the approximate mixing time. Recall that if Lewis starts from any node, after T steps Lewis is at node u_k with probability between $\pi_k - 1/n^2$ and $\pi_k + 1/n^2$. Thus, Lewis' position after the kth period is almost independent of Lewis' position after the $(k+1)$st period.

We associate a 0/1-valued random variable, X_k, with the kth period of Lewis' random walk.

$$X_k = \begin{cases} 1 & \text{if Lewis is at node } u_0 \text{ at the end of the } k\text{th period} \\ 0 & \text{otherwise.} \end{cases}$$

Since u_0 is a likely node, $X_k = 1$ with probability at least $1/2n$. Let $X = \sum_{k=1}^{(4+\tau)dn^2} X_k$ be the number of times Lewis returns to u_0 in $(4+\tau)dn^2$ periods. Note that $E(X)$ is at least $(4+\tau)dn/2$.

Using Chernoff bounds with $\delta = 1 - (2/(4+\tau))$ we find:

$$Pr[X < (1-\delta)E[X]] < Pr[X < dn] < e^{\frac{-(4+\tau)dn}{4}\left(1-\frac{2}{4+\tau}\right)^2}$$

$$< e^{\frac{-(4+\tau)dn}{4}\left(1-\frac{4}{4+\tau}+\frac{4}{(4+\tau)^2}\right)} < e^{\frac{-(4+\tau)dn}{4}+dn} < e^{\frac{-dn\tau}{4}}.$$

□

Thus, with high probability **Build-Map** runs in $O(dn^3 + dn^3(4+\tau)T)$ steps. Suppose none of the failure conditions occurs in a run of **Learn-Graph2**. Then the execution never calls **Build-Map** on an unlikely node, *does* call **Build-Map** on a likely node, and when it does, **Build-Map** returns TRUE. Therefore **Learn-Graph2** makes at most w steady-state probability estimates, each taking $O(wT)$ steps, before calling **Build-Map** once. Therefore, the running time is $O((4+\tau)dn^3T)$, proving Theorem 12.
□

3.6.5 Exploring Without Prior Knowledge

Prior knowledge of n is used in two ways in **Learn-Graph2**: to estimate the stationary probability of a node and to compute the mixing time T. If T is known but n is not, the algorithm can forego estimating π_i entirely and simply run **Build-Map** after step 6. The removal of lines 7 – 12 from **Learn-Graph2** yields a new algorithm whose expected running time is polynomially slower than the original. If we know neither n nor T, we can run this new algorithm using standard doubling to estimate the quantity MT. This quantity can be used in line 6 of **Learn-Graph2** and also in line 7 of **Build-Map** as an upper bound on the length of the random walks. Thus no prior knowledge of the graph is necessary.

3.7 Conclusions and Open Problems

Note that with high probability, a single robot with a pebble can simulate algorithm **Learn-Graph2** with a substantial but polynomial slowdown. However, **Learn-Graph2** does not run in polynomial expected time on graphs with exponentially-small conductance. An open problem is to establish tight bounds on the running time of an algorithm that uses one robot and a constant number of pebbles to learn an n-node graph G. We conjecture that the lower bound will be a function of $\phi(G)$, but there may be other graph characteristics (*e.g.*, cover time) which yield better bounds. It would also be interesting to establish tight bounds on the number of pebbles a single robot needs to learn graphs in polynomial time.

Another direction for future work is to find other special classes of graphs that two robots can learn substantially more quickly than general directed graphs, and to find efficient algorithms for these cases.

Several additional questions concern the practical application of our results in the area of robot navigation. Suppose that a robot can recognize a node correctly $x\%$ of the time and that it uses this information to augment its candidate homing sequence. How does our learning algorithm speed up as x increases? What happens if one robot recognizes the node but the other does not? If landmark recognition may actually be *incorrect* some fraction of the time, how can the techniques described in this chapter help to correct these mistakes? Clearly, the robots could ignore *all* distinguishing information and use our algorithm for completely indistinguishable nodes. However, if most landmarks are recognized correctly, there may be hybrid schemes that allow a team of robots to learn accurately and quickly.

CHAPTER 4

Building Human Genome Maps with Radiation Hybrids

4.1 Introduction

In this chapter we investigate the computational problem of constructing accurate physical maps of the human genome. This case study illustrates the practical challenges of drawing accurate conclusions from noisy data. We introduce a theoretical model of the noise that helps us solve the problem, and we explore the point at which the model breaks down. We are able to compensate for the limits of our model by incorporating additional information into our search techniques. This integrated strategy yields excellent results: our team has assembled the first radiation hybrid map of the entire human genome. Our work illustrates that even when it is difficult to model a noise source exactly, we *can* learn from imperfect data in practice.

This chapter is self-contained; no genetics background is assumed. However, the reader unfamiliar with biology may find it helpful to refer to the glossary at the end of the chapter.

4.1.1 Mapping the Genome: What and Why?

The *human genome* refers to the entire complement of human genetic material (DNA); a copy of the genome is present in each cell in the body. One can think of a DNA molecule as a long string over the four-letter alphabet $\{A, C, G, T\}$. Each of these letters is called a *nucleotide* or a *base*. A gene is a substring of a DNA strand; genes range from a few hundred to many thousand bases in length. The human genome is about 3 billion bases long and is divided into 23 pairs of chromosomes. These include the *autosomes*, 22 pairs of homologous chromosomes (numbered 1..22) that are present in all cells, and the *sex chromosomes* X and Y: females have two copies of the X chromosome; males have one X and one Y chromosome.

Each gene encodes a protein that has a specific function in the body. In addition to genes controlling physical traits such as eye color or more complex behavioral traits, there are genes governing almost all cell functions: they regulate cell growth and reproduction, control the transport of materials across cell membranes, and catalyze chemical reactions. Thus, a malfunctioning gene may have a noticeable effect in the body and may even cause disease.

For example, consider a gene whose normal function tells cells when to stop reproduction. If that gene is deleted or fails to function normally in some cell, the cell might divide continually. All of the cell's offspring would also replicate themselves without check. One can imagine how such a malfunction could lead to tumor growth and cancer.

This does not imply that all cancers are caused by an inherited genetic flaw. However, the treatment of all cancers may benefit from genetics research. By finding the genes responsible for certain cancers and studying how those genes function in the body, we gain understanding of the mechanism of the disease. This information may be used to develop new treatments and therapies. Thus, the field of genetics plays an essential role in modern medical research.

An intense effort by many teams of scientists worldwide is currently underway to determine the location, DNA sequence and function of human genes [38, 55]. Physical maps are an important part of this process. A *physical map* of a chromosome shows the relative locations and estimated distances between known *markers* along the chromosome. The markers may be genes or simply arbitrary DNA substrings that appear only once in the genome. (A sample map is shown in Figure 4.12.) While our maps indicate the location of each marker as a point on a line representing the chromosome, in fact each marker may be several thousand bases long. The U.S. Human Genome Project's current goal is to construct a physical map of 30,000 markers spanning the entire genome, with an average spacing of 100 kilobases (kb) between markers, by the end of 1998.

There are two main reasons for building physical maps. The first is that genome maps are essential tools for finding new genes. For a researcher to clone a novel gene causing a certain disease, the gene must first be localized to a specific region of a chromosome. For example, the researcher might notice that several patients with the disease are missing a small piece of Chromosome 14, suggesting that some gene in that region of Chromosome 14 is related to the disease. However, there may be a thousand genes in that part of the chromosome. The process of finding all genes in the region and examining each of them would take many years. Using a physical map, the researcher can locate the target gene with respect to the map markers. This process can narrow down the search to a region containing only 10 or 20 genes, each of which can then be tested individually.

In the past, if the chromosomal region of interest had not already been mapped, researchers looking for genes had to build a map of the region before continuing their search. Having a map of the entire genome will eliminate this time-consuming work from future gene-finding projects.

The other reason for building physical maps is that they form a scaffold for sequencing. The task of "sequencing the genome," reading the sequence of all 3 billion bases of human DNA, is the next phase of the Human Genome Project. While the DNA sequences of different people are not all identical, the differences between two individuals generally account for less than 0.1% of the genome. Thus one can propose to sequence the 99.9% of the DNA that all individuals share and to determine which of the remaining differences between individuals are critical and in what ways.

Current technology allows us to sequence only a few hundred bases at a time. From these short reads we can assemble the sequence of regions about 100 kb in length.[1] If we could build a map with markers spaced 100 kb apart, we could then use the map to help assemble the final sequence. Thus timely completion of a physical map is a crucial part of the Genome Project. The project's ultimate goal is to analyze the sequenced genome, find genes, and study their function in the body. This work is expected to yield a wealth of information that will revolutionize medical and biological research.

Physical mapping is also an excellent case study in handling noisy or imperfect data. Physical map data can be derived from a variety of experimental methods; *radiation hybrid mapping* is one such mapping technique [43]. The project described in this chapter involves a great deal of data developed in different labs and with different experimental methods. Each type of data is subject to different sorts of noise and corruption. Failures in the lab, unclear test results, and human error all contribute to the noise problem. Furthermore, while this noise is not malicious, it is also not uniformly random, but simply follows some unknown pattern. Thus any theoretical model of this noise is bound to be incorrect in some cases.

At first, one might think that finding order in a large data set subject to unknown patterns of error is an impossible task. However, we have developed a suite of programs that allows us to construct good radiation hybrid maps efficiently. While there is no "correct map" against which to test our results, we have evidence that our maps are both well-supported by the data and very close to the truth.

[1]The problem of assembling the correct sequence from a number of short, overlapping subsequences containing various sorts of errors is not a trivial one. It has been the subject of a great deal of research that is beyond the scope of this thesis; see [67] for a summary of current sequence assembly methods.

Chapter Overview

This chapter describes the practical problems of building genome-wide radiation hybrid maps from noisy data. The work is part of a collaborative effort with the physical mapping group at the Whitehead Institute Center for Genome Research. We have constructed a radiation hybrid map containing over 11,000 unique markers with estimated coverage of about 99% of the human genome. We have also released a software package incorporating the algorithms used to build and debug our map.

Section 4.2 introduces the technique of radiation hybrid mapping. It places the problem in the context of previous mapping work, describes the experimental method and the computational issues of mapping, and outlines our approach to map construction. Section 4.3 defines a hidden Markov model with which we represent our data and shows how we use the model to detect and correct errors.

The subsequent three sections are concerned with algorithms for finding good maps from an exponentially-large space of possible maps. Section 4.4 describes some early experiments with basic combinatorial algorithms. In the past, such algorithms have been used to solve many small mapping problems. However, we have had difficulty finding efficient implementations suitable for our large-scale genome-wide mapping project. Understanding why our preliminary experiments failed yields some insight into the complexities of the problem. Section 4.5 defines an effective greedy approach to map construction and presents the results of experiments testing this method. Section 4.6 introduces the notions of *framework* and *placement* maps, maps that indicate a degree of confidence in the position of each marker. Such information is of great value to the research community using our maps. This section includes the algorithms that we ultimately used to produce our published maps.

Finally, in Section 4.7 we present our results. We first introduce RHMAPPER, a software package that incorporates our methods of map construction. Next we discuss the practical problems encountered in building genome-wide maps, describe the maps produced, show a sample map, and discuss some of the interesting biological implications of our work. We then describe how a preliminary version of our radiation hybrid maps was combined with other types of mapping data to produce an integrated genome-wide map of over 15,000 markers. This map is about half the density of the map specified as the Human Genome Project's 1998 goal. We expect that this goal of a 30,000-marker map will be reached well ahead of schedule, in part due to the work described in this chapter.

4.2 Radiation Hybrid Mapping

Radiation hybrid mapping [43] is a technique that has been used for small-scale mapping since 1990 [61, 1, 62]. The experimental method involves exposing human cells to gamma radiation, which breaks each chromosome into random fragments. The DNA fragments are then "rescued" by fusion with healthy hamster cells that incorporate or *retain* a random subset of the human DNA fragments. Each resulting *hybrid* cell, which includes both hamster and human DNA, can be cloned to form a *hybrid cell line* of cells all containing the same random subset of the human genome. A *radiation hybrid panel* consists of a number of different hybrid cell lines (sometimes just called "hybrids").

To understand the computational problems of map construction, we consider a single chromosome with four markers, A, B, C, and D, as shown in Figure 4.1. Our goal is to determine the correct order of the markers along the chromosome and the approximate distances between adjacent markers. We assume that breaks occur uniformly at random across the chromosome. The crucial observation is that for a pair of markers such as A and B that are near each other on the chromosome, the probability that the radiation induces a break between the two markers is quite small. If there is no break between the two markers, they are *co-retained*; that is, the hybrid contains either both or neither of the markers. In contrast, for markers that are far apart like B and C, there is a high probability that at least one break occurs between the two. Thus, the fragments containing markers B and C are retained independently by most hybrid cell lines.

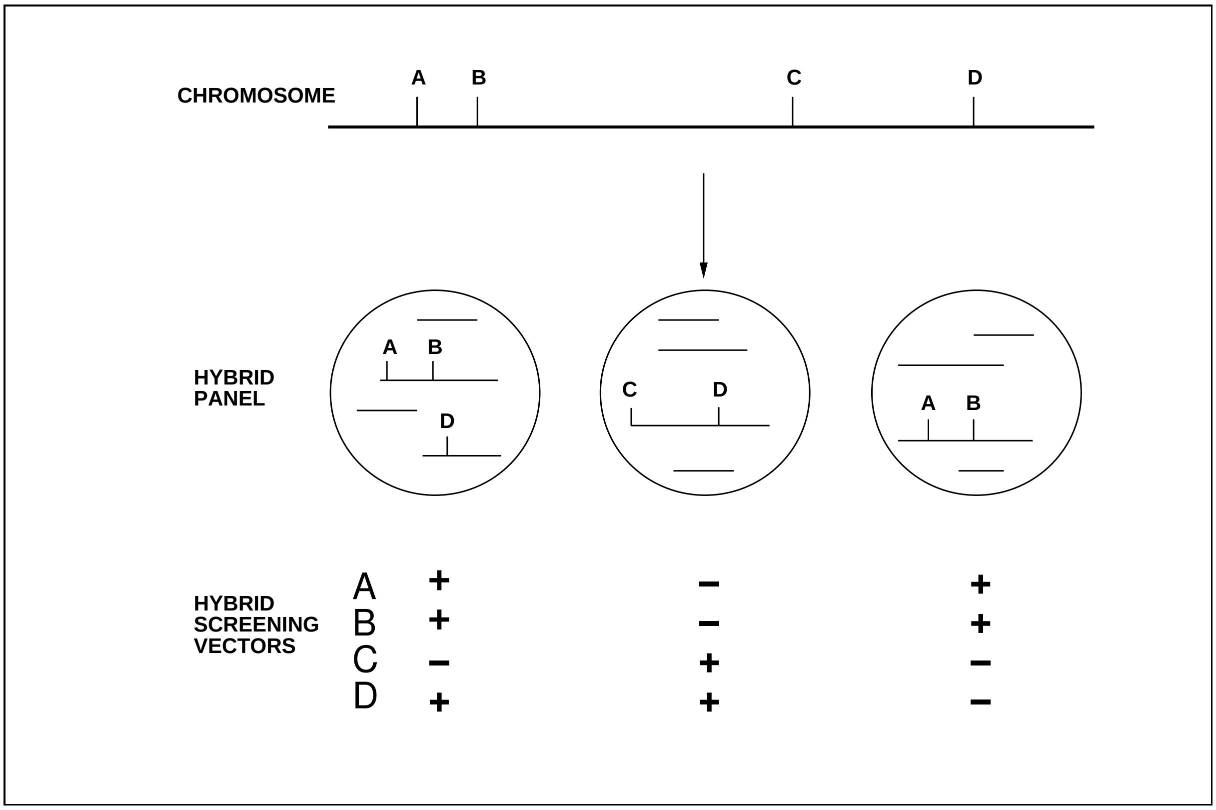


Figure 4.1: A DNA strand with markers A through D in a radiation hybrid panel (made from many cells containing copies of the DNA). Gamma irradiation breaks the DNA in each cell in a random fashion. The fragmented DNA is cloned into several hybrid cells, each of which retains a random subset of the fragments. Each hybrid can then be screened against each marker. In the resulting matrix, a "+" indicates that the marker in question is retained by the hybrid, while a "–" indicates that it is not. The computational problem is to use this matrix to reconstruct the correct order of and distances between the markers.

To test for co-retention, we screen the DNA in each hybrid in the panel against each marker. In the absence of errors, a positive result (represented by a "+") indicates that the hybrid in question retained the marker, while a negative result (represented by a "–") indicates that the marker was not retained. Thus, the data form a matrix of pluses and minuses, one bit for each marker/hybrid pair. Our task is to reconstruct the markers' positions on the original chromosome given only their retention patterns in this matrix.

This reconstruction process is complicated by several factors. The first problem is that the screening is subject to several types of noise. Thus there is some probability (generally less than 2%) of seeing a false-positive or false-negative result. To decrease the error rate, we perform each experiment twice. If the two results are discrepant, we record a "?" instead of a "–" or a "+" in the input matrix, representing the fact that we do not know whether or not the marker was retained by the hybrid.

Another issue to consider in determining co-retention is that humans are *diploid* organisms, meaning that each cell contains two copies of each chromosome. (In contrast, each cell in a *haploid* organism contains only a single copy of each chromosome.) All that we can determine by our screening methods, however, is whether or not *at least one copy* of a marker is retained by a hybrid. We cannot distinguish the exact number of copies retained. We must account for this limitation when we attempt to determine accurate distances between adjacent markers in our maps.

The unit of distance in radiation hybrid maps is the Ray [31, 43]. The distance is calculated with the Haldane [56] formula: $-\log(1-\theta)$, where θ is the probability of a break between two markers. One centiRay (cR) corresponds roughly to a 1% chance of a break. Each centiRay also corresponds to a rough physical distance (in Mb) based on the radiation dosage used in creating the hybrid panel. Since there is a direct correspondence between distances and break probabilities, we sometimes refer to break probabilities as if they were distances.

There are two key computational issues in radiation hybrid mapping. The first is that of determining how accurate a map is. We need a way to evaluate each map quantitatively to assess how well it is supported by the observed data. To determine the correct order and spacing of markers we may need to compare maps repeatedly, so the evaluation function must be easily and efficiently computable.

The second issue is that of efficiently searching the space of all possible maps to find the best maps. For a set of n markers, there are $n!$ possible marker orders. With just 20 markers there are several quadrillion possible orders; we want to manage data sets of hundreds or even thousands of markers on each chromosome. Thus, we need an efficient method of choosing candidate maps so that with high probability we find the true order of markers along the chromosome (or something very close to it) in a reasonable amount of time.

Previous Work on Radiation Hybrid Mapping

Analytical methods for constructing radiation hybrid maps have been published by Boehnke, *et al.* [30, 31, 73, 76]. Their software, RHMAP, was originally designed for building maps of small chromosomal regions near disease genes. The hybrid panels used for this purpose are derived from somatic cell hybrids containing only a single copy of the human chromosome of interest. Thus, while software for handling the haploid case has been widely available for a number of years [32], software that handles the diploid panels used in genome-wide mapping has only recently become available [30, 73].

Boehnke solves the first computational problem, that of determining what makes a good map, by representing the data with a Markov model. Under this probabilistic model one can compute the *likelihood* of each map. The likelihood is the probability of seeing the observed data given the map; this criterion measures how well the map fits the observed data.

The second problem, that of finding the right map efficiently, is more difficult. Boehnke's group has tried several approaches to this problem. One of their search methods is a greedy algorithm similar to those we describe in Section 4.5. We favorably compare the efficiency of our greedy approach to theirs. Another option in Boehnke's software performs a branch-and-bound exhaustive search that finds the best overall order. This method, however, is extremely slow; although it prunes the search tree substantially, its running time is exponential in the number of markers. A third RHMAP option uses simulated annealing to find a maximum-likelihood solution. In Section 4.4 we describe our preliminary attempts to apply similar techniques to large-scale mapping.

Radiation hybrid mapping has been used successfully to create maps of human chromosomes 14 [111], 11 [63], 4 and 12 [42]. However, building maps of entire chromosomes using RHMAP is a slow and painful process. Since the software takes several hours to order a group of just 20 markers using the fastest method (see Table 4.1), mapping several hundred markers requires breaking the markers up into groups of about 20 markers, finding the best maps of each, and then merging the results. The process demands a good deal of human intervention at every step. Thus we determined that these methods were not suitable for whole-genome mapping.

Another key challenge in genome-wide mapping projects is dealing with the inevitable experimental errors. Laboratory errors in the characterization of markers on radiation hybrid panels lead to false breakage events, creating regions of map expansion and interfering with the correct ordering of markers. While errors create difficulties in monochromosomal mapping as well, the problem is even more pervasive in large-scale mapping efforts. RHMAP does not account for errors in the data at all. Lunetta, *et al.* [76] analyze the impact of such errors on map construction and conclude that even low error rates can significantly confound mapping efforts. To be practical, map software should accommodate noisy data and flag suspected laboratory errors for experimental verification.

Our Approach

We use a hidden Markov model to represent our data. While our approach is an extension of Boehnke's maximum-likelihood method, the differences are significant. The "hidden" states of the model allow us to represent uncertainty in the data. Thus, the model accounts for missing data, errors, and either diploid or haploid cell lines. Using experimentally-determined error rates, our software predicts which marker/hybrid assays are likely to be errors. These data are then flagged for laboratory verification and the corrected data are incorporated into new maps. The hidden Markov model is described in detail in Section 4.3.

We experimented with several algorithms for finding maximum-likelihood maps under our model. While basic optimization techniques such as simulated annealing are effective for smaller problems, our preliminary efforts in applying these techniques to large-scale mapping have been disappointing. However, we describe a number of greedy strategies that are fast and successful at finding good maps.

Our software thus has several advantages over RHMAP: it handles both haploid and diploid data, it performs error-detection and flags putative errors for verification, and it runs much more quickly while delivering good results. We have validated our approach by using it to construct genome-wide radiation hybrid maps.

Other Physical Mapping Strategies

We also experimented briefly with algorithmic methods that have been applied successfully to other physical mapping problems. The earliest physical maps consisted of overlapping clone coverage of a region [84, 41, 68]. The clones most suitable for large-scale mapping are *yeast artificial chromosomes* (YACs). Each YAC incorporates an approximately 1 megabase (Mb) DNA fragment from a random part of the human genome. Recently, a collaboration between CEPH, Genethon, and the Whitehead has produced a clone-based map that is estimated to cover 75% of the genome with YAC clones [37].

Clone-based maps rely heavily on the accuracy of the cloning technology. However, most cloning techniques are prone to a variety of errors. There are many types of errors common to YACs: a piece of the human DNA may be deleted in the clone, or the DNA may contain inversions or other rearrangements. Repeated DNA regions across the genome can also cause difficulty in map construction. Perhaps the most problematic and most common errors are *chimeric* clones. A clone is said to be *chimeric* if it picks up two pieces of DNA from different parts of the genome. The danger of chimeric clones is that the two different DNA pieces appear as though they were adjacent in the cell. Thus, chimeric clones can cause substantial trouble in map construction. As many as 50% of the YAC clones used in mapping may be chimeric, so any mapping software must be able to handle chimeras.

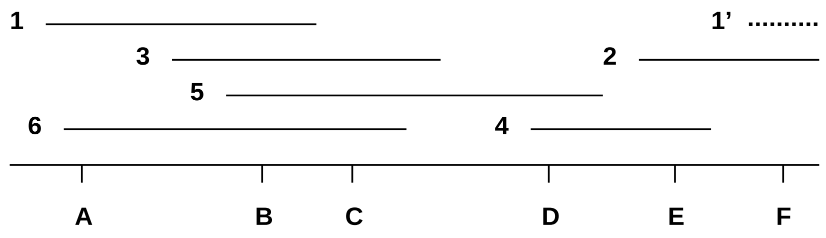
Another approach to physical mapping is to build a marker-based map by STS-content mapping on YACs. An STS, or *sequence-tagged site*, is a DNA marker with known flanking sequences that can be used for cloning. Given a set of STSs and a number of YACs, we can test to see which YAC clones contain each STS. If two YACs hit several of the same markers, they are probably derived from overlapping regions of DNA.

The computational problem of STS-content mapping looks a bit like that of radiation hybrid mapping. The data consist of a matrix of "–"s and "+"s, where each YAC clone represents a column and each STS represents a row. If the rows are arranged in the order in which the STSs appear along the chromosome, and if there are no errors in the data, then the matrix would have the *consecutive ones property*: all the positive results (sometimes represented as ones in a zero/one matrix) in each column would appear in consecutive rows, as in Figure 4.2b. Thus the goal is to find a permutation of the rows of the input matrix with the consecutive ones property.

This problem has a polynomial-time solution in the error-free case, due to Booth and Lueker[33]. However, as in any mapping problem, there are several types of errors in the data: deletions, insertions, false-positive or false-negative experimental results, and chimeras. Thus, the real matrix might look more like that in Figure 4.2c. Solving the "almost-consecutive-ones" problem that arises in the error-prone case is much harder.

In fact, Alizadeh, Karp, Newberg and Weisser prove that a formalization of the almost-consecutive-ones problem is NP-complete [2]. They reduce the problem to a variant of the traveling salesman problem. The key concept behind their approach is the notion of *gap minimization.* A *gap* in a column of the data matrix is defined to be a consecutive run of some number of "–"s flanked by a "+" on either side. Each false-positive, false-negative, or chimeric clone adds a gap to the matrix (as can be seen in Figure 4.2c). Thus, it seems reasonable that the correct permutation is one that minimizes the number of gaps in the matrix.

Karp's group has implemented a traveling salesman approximation algorithm as a method of approximate gap-minimization [113]. Their methods are extremely sensitive a)



b)

	Clones					
STSs	1	2	3	4	5	6
A	+	−	−	−	−	+
B	+	−	+	−	+	+
C	−	−	+	−	+	+
D	−	−	−	+	+	−
E	−	+	−	+	−	−
F	−	+	−	−	−	−

c)

	Clones					
STSs	1	2	3	4	5	6
A	+	−	−	+	−	+
B	+	−	+	−	+	−
C	−	−	+	−	+	+
D	−	−	−	+	+	−
E	−	+	−	+	−	−
F	+	+	−	−	−	−

Figure 4.2: **a)** A map showing YAC clones 1 through 6 and STSs A through F. **b)** The error-free data matrix for STS-content mapping on YACs for the region mapped in a). **c)** The same matrix with some errors. There is a false positive hit between clone 4 and STS A and a false negative between clone 6 and STS B. Clone 1 has become *chimeric*, retaining two fragments: 1 and 1’. Note how the number of *gaps* (consecutive runs of “–”s flanked by “+”s) in each column increases when errors are introduced.

to false-positive errors but can tolerate a 20-30% false-negative error rate and a 25% chimerism rate. They obtain fairly accurate maps for simulated data with a false-negative rate of 0.1% [3]. However, building large-scale maps by STS-content mapping alone has proven to be difficult in practice because of the high rate of false positives and the frequency of repeated DNA.

One key difference between the computational problems of STS-content mapping and radiation hybrid mapping is that even in the error-free case, the correct permutation of the input matrix for radiation hybrid mapping would not have the consecutive ones property. Since many different human DNA fragments are retained by each hybrid cell line, the columns of the matrix would contain many groups of consecutive ones. Despite this difference, we attempted to build radiation hybrid maps using the gap-minimization code that Karp's group wrote for STS-content mapping [113]; we describe the results in Section 4.4.

4.3 The Hidden Markov Model

In this section we present an answer to the first fundamental question in mapping, "how good is a map?" Our approach compares different maps by constructing a probabilistic model of the data. Given any map consisting of an ordered list of markers and the distances between them, one can compute the probability under the model that the map produced the observed data. This probability is known as the *likelihood* of the map.

The ratio of the likelihoods of two different maps containing the same markers provides a quantitative method of comparison between the two. Since the likelihood of any particular map is extremely small, we measure the difference between the (base 10) logarithms of the likelihoods instead of the direct likelihood ratio. Borrowing terminology from genetic linkage mapping, we refer to the difference of the log likelihoods of two specific maps as the *lod score*.

The *pairwise lod score* of two markers compares the log likelihoods of two maps: the most-likely map of the two markers, and the map placing the two markers infinitely far apart ($\theta = 1$). This comparison effectively measures the likelihood that the two markers are linked.

One advantage of the hidden Markov model is that it allows us to account for errors and diploid data in a straightforward manner. The model also offers an efficient method for finding the best map distances associated with a given marker order; these distances are an important feature of our maps. To determine the likelihood of a particular order we use the estimation-maximization (EM) algorithm, which efficiently estimates the most-likely distances between adjacent markers in the given order. (Lander and Green [71] describe a similar process for building maps from genetic linkage data.)

Our hidden Markov model relies on several assumptions. (For a basic tutorial on hidden Markov models, see Rabiner [90].) We assume that the radiation-induced breaks occur randomly along a chromosome as a Poisson process, and that different fragments are retained independently in a given hybrid. The retention rate is taken to be a constant for each hybrid, but different hybrids may have different retention rates. The Markovian assumption is that the retention of a marker depends only on the retention of the previous marker in the order and the chance of a break between the two markers.

4.3.1 Likelihood Calculations

The likelihood of a map is simply Pr(Data|Map), the probability of seeing the observed data given the map order and break probabilities. To describe the likelihood computation we first need a few definitions.

Let N be the number of markers in the map. A separate hidden Markov model is defined for each hybrid in the following way. For each marker i, the hidden Markov model has a set S_i of states. The state the model is in at a given marker corresponds to the number of copies of that marker actually retained by the given hybrid. Thus in the haploid case, $S_i = \{0, 1\}$, while in the diploid case, $S_i = \{0, 1, 2\}$, for all i between 1 and N.

The observed data O_i for the ith marker at a particular hybrid may be either +, −, or ?, representing positive, negative, or uncertain results of the marker/hybrid assay. The transition and output probabilities are described below. Figure 4.3 shows the hidden Markov model for one hybrid in the haploid case. In the diploid case there would be an additional state labeled "2" for each marker, and the edges connecting the states of two consecutive markers would form a complete bipartite graph with all edges directed towards the higher-lettered marker.

For example, suppose the observed result of the assay for marker A in hybrid i is negative. If this observation is correct, marker A is truly not retained by hybrid i. This case corresponds to being in state 0 of the model at marker A. Suppose that α and β are the average false-positive and false-negative error rates, respectively. Then the probability of seeing the negative result if the model is truly in state 0 at marker A is $1 - \alpha$. The other possibility is that the observed result is a false-negative error; the marker A really is retained by hybrid i, but the observed result is incorrect. This case corresponds to being in state 1 at marker A. The probability of seeing the incorrect observation, given that the model is in state 1, is exactly the false-negative probability β.

Since the hybrid cell lines are independent of one another, the total likelihood is the product of the likelihoods for the individual hybrids. Hence we describe the method for computing the likelihood of a map for a single hybrid.

Our goal is to determine Pr(Data|Map). We compute this quantity inductively using Baum's forward-backward algorithm (see Rabiner [90] for an introduction to this and other basic HMM algorithms). Let $Pr^L(i,j)$, the left-conditioned probability of state j at marker i, represent the probability of being in state j at marker i and seeing the observed data at the first i markers, given the map. The left-conditioned probabilities can be computed inductively using the following equations:

$$Pr^L(1,j) = Prior(j)Obs(O_1 \mid j), \ \ \forall j \in S_1, \text{ and}$$

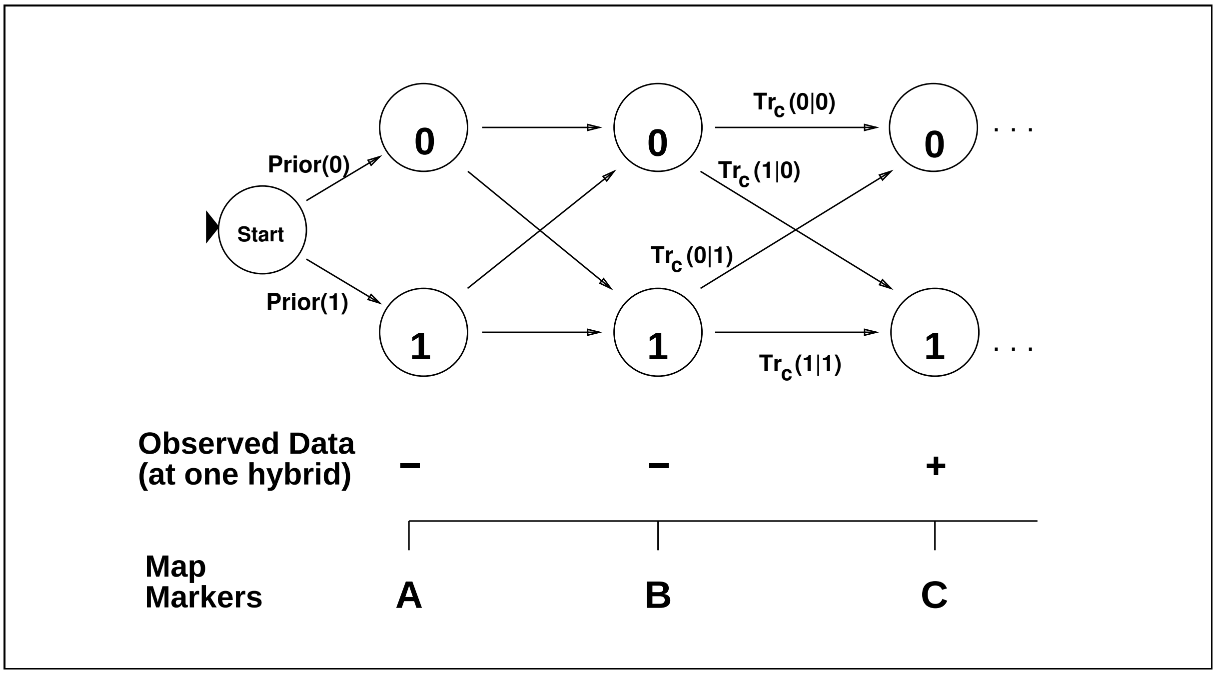


Figure 4.3: The hidden Markov model for the haploid case. For the map consisting of ordered markers A,B,C,..., this model represents the true state of a particular hybrid cell line. At each marker the system may be in one of two states, corresponding to whether or not the marker is retained by the hybrid in question. For example, the rightmost state labeled "0" represents the case in which marker C is not retained by the hybrid. One can then calculate the probability of seeing the observed data for that marker (in this case, a "+"), given that the marker is not retained; this is just the false-positive rate α. In this way one can determine the probability of seeing all the observed data given the map.

$$Pr^L(i,j) = Obs(O_i \mid j) \left(\sum_{k \in S_{i-1}} Pr^L(i-1,k)\, Tr_i(j \mid k) \right), \quad \forall j \in S_i,$$

where $Obs(O_i \mid j)$ is the probability of seeing the observed data for marker i if the model is in state j at the ith marker, $Tr_i(j \mid k)$ represents the transition probability of moving from state k at the $(i-1)$st marker into state j at the ith marker, and $Prior(j)$ is the prior probability of starting in state j. The prior probabilities can be experimentally determined by measuring the *retention frequency*, the percentage of the DNA fragments that are retained by a hybrid. For example, in the haploid case for a hybrid with retention frequency r, $Prior(0) = 1 - r$ and $Prior(1) = r$.

If the screens are carried out without error on haploid hybrid lines and no data are missing then the observed and true states are in one-to-one correspondence, so $Obs(- \mid 0) = 1$, and $Obs(+ \mid 1) = 1$. Accounting for errors changes these probabilities. Recall that the states of the model indicate how many copies of the marker are actually retained in the given hybrid cell line. Suppose α is the average false positive rate and β is the average false negative rate. Then

$$\begin{aligned} Obs(- \mid 0) &= 1 - \alpha, \\ Obs(+ \mid 0) &= \alpha, \\ Obs(+ \mid 1) &= 1 - \beta, \text{ and} \\ Obs(- \mid 1) &= \beta. \end{aligned}$$

Diploid and polyploid cases are treated by allowing all states other than 0 (those states in which some copy of the marker is retained) to correspond to a positive assay result ($O_i = +$). Thus in the diploid case, we add $Obs(+ \mid 2) = 1 - \beta$ and $Obs(- \mid 2) = \beta$ to the above equations.

We further extend the model to account for missing data. Let γ represent the probability that an individual marker/hybrid assay result is missing or inconclusive (such a result is represented by a "?" in the matrix of observed data). Let $Obs'(O_i \mid j)$ represent the new probability of seeing the observed data for marker i when the model is in state j. Then

$$Obs'(O_i \mid j) = \begin{cases} \gamma & \text{if } O_i = ? \\ (1-\gamma)\, Obs(O_i \mid j) & \text{otherwise} \end{cases}$$

The transition probabilities for markers $i-1$ and i separated by break probability θ can be computed as follows. For the haploid case one can verify that $Tr_i(1 \mid 0) = \theta r$ (a break occurs and the second marker is retained), $Tr_i(0 \mid 0) = (1-\theta) + \theta(1-r)$ (either no break occurs, or a break does occur and the second marker is not retained), $Tr_i(0 \mid 1) = \theta(1-r)$ (a break occurs and the second marker is not retained), and

$Tr_i(1 \mid 1) = (1-\theta) + \theta r$ (either no break occurs, or a break does occur and the second marker is retained.

If we designate these probabilities as t_{10}, t_{00}, t_{01}, and t_{11} respectively, then in the general n-ploid case, the transition probability from the state with k retained copies to the state with ℓ retained copies is given by the equation:

$$T_{lk} = \sum_{m=0}^{n-k} \binom{n-k}{m} \binom{k}{n-\ell-m} t_{00}^{m} t_{10}^{n-k-m} t_{01}^{n-\ell-m} t_{11}^{k+\ell+m-n}.$$

In this equation we rely on the convention that $\binom{s}{t}$ is equal to 0 if $t > s$ or $t < 0$. The index m counts the number of copies that are not retained in either state.

The transition matrix for the diploid case is easily derived from this equation or by considering the fates of the individual fragments:

	k		
ℓ	0	1	2
0	$(t_{00})^2$	$t_{00}t_{01}$	$(t_{01})^2$
1	$2(t_{10}t_{00})$	$t_{00}t_{11}+ t_{01}t_{10}$	$2(t_{11}t_{01})$
2	$(t_{10})^2$	$t_{11}t_{00}$	$(t_{11})^2$

The matrix entries show the probability of moving from a state with k retained copies of a marker to one with ℓ retained copies of the next marker. For example, in the diploid case T_{00} represents the probability of moving from the state with neither copy of the first marker retained to the state in which no copy of the second marker is retained. Thus, each (haploid) strand of the DNA must move from state 0 to state 0; the probability that both strands do this is just $(t_{00})^2$.

The likelihood for an entire hybrid can be computed by summing the left-conditioned probabilities over all states for the rightmost marker:

$$L = \sum_{k \in S_N} Pr^L(N, k).$$

If we were only interested in determining the likelihood of each hybrid, this result would be sufficient. However, to perform error detection we need to know the probability of being in a given state conditioned on all the data, so the intermediate probabilities $Pr^L(i,j)$ become important.

4.3.2 Error Detection

The error-detection algorithm determines the ratio of the likelihood that the experimental result observed at a marker is in error to the likelihood that the result is correct, conditioned on all the observed data for the hybrid. For example, consider the haploid case with no missing data, where the possible states correspond to having either 0 or 1 copy retained and where there are only two possible observations, $-$ or $+$. We write $Pr(\text{state at } i = O_i)$ to represent the probability that the state at marker i corresponds to a state in which one would see the observed output O_i if no error occurred. Then the probability that the observed result at that point is correct is just $Pr(\text{state at } i = O_i \mid \text{data})$, while the probability that it is incorrect is $Pr(\text{state at } i \neq O_i \mid \text{data})$.

These probabilities depend upon *all* the observed data for the hybrid, so the left-conditioned probabilities $Pr^L(i,j)$ solve only half the problem. Analogously, one can define the right-conditioned probability $Pr^R(i,j)$, the probability of the observed data at markers $i+1$ through N, given the map and the fact that the model is in state j at marker i:

$$Pr^R(N,j) = 1, \quad \forall j \in S_N, \text{and}$$

$$Pr^R(i,j) = \sum_{k \in S_{i+1}} Obs(O_{i+1} \mid k)\, Tr_i(k \mid j)\, Pr^R(i+1,k), \quad \forall j \in S_i,$$

Then to find $Pr(\text{state at } i = O_i \mid \text{data})$ in the haploid case, we only need the product $Pr^L(i, \text{state at } i = O_i) \cdot Pr^R(i, \text{state at } i = O_i)$. (In fact, this quantity should be normalized by the probability of the data given the map. However, since we ultimately want the ratio of two such likelihoods, these normalization factors cancel each other out. Thus, they may safely be ignored in our calculations.) In the more general polyploid case, the lod score in favor of an error is

$$\frac{\sum_{\text{state at } j \neq O_i} Pr^L(i,j) \cdot Pr^R(i,j)}{\sum_{\text{state at } j = O_i} Pr^L(i,j) \cdot Pr^R(i,j)}.$$

One can calculate the likelihood of a hybrid inductively in either direction (i.e., using either $Pr^L(i,j)$ or $Pr^R(i,j)$). Thus performing both calculations only doubles the work, and it allows us to determine the probability of error at each point with only a few arithmetic operations.

4.3.3 The EM Algorithm

Given an ordered set of markers, we employ the EM algorithm to find the most likely map distances associated with that order. The algorithm requires choosing an arbitrary set of break probabilities as a starting point; we assume initially that all markers are evenly spaced. (The initial break probabilities do not influence the result, but they may affect how long the algorithm takes to converge.) Using the method described above, the algorithm evaluates the likelihood of the map with these specific break probabilities. This likelihood calculation also yields the state probabilities $Pr(\text{state } i \text{ at marker } j \mid \text{data})$ for each state $i \in S_j$. The algorithm then re-estimates the distances between markers by counting the expected number of breaks using the state probabilities. The new break probabilities correspond to new distance estimates. This process is repeated (using the new distance estimates to calculate the likelihood of the next map) until it converges. The procedure is guaranteed to converge to a local maximum; in practice, it converges rapidly to the globally-optimal map.

Finally, we describe how the EM algorithm re-estimates the break probabilities to calculate new map distances. Again, we first consider the haploid case. If two adjacent markers are in different states (0,1, or 1,0), the probability of a break having occurred between the markers is 1. If the markers are in the same state (1,1 or 0,0, indicating that both are retained or both are not retained), a break may or may not have occurred. The corresponding break probabilities, denoted b_{11} and b_{00}, are:

$$b_{11} = \frac{\theta r^2}{\theta r^2 + (1-\theta)r}, \text{ and}$$

$$b_{00} = \frac{\theta(1-r)^2}{\theta(1-r)^2 + (1-\theta)(1-r)}.$$

For the diploid case, the expected number of breaks between states with k and ℓ copies retained can be derived from these expressions. The diploid matrix of the expected number of breaks is shown below.

ℓ \ k	0	1	2
0	$2\ b_{00}$	$b_{00}+1$	2
1	$b_{00}+1$	*cis* $(b_{00}+b_{11}) + 2$ *trans*	$b_{11}+1$
2	2	$b_{11}+1$	$2\ b_{11}$

The only case that requires additional explanation is the middle square, representing the expected number of breaks when moving from state 1 at the first marker to state 1 at the second. In this case, there are two possibilities; either both markers retained are on the same strand, or the retained markers are on different strands. These probabilities are respectively denoted by their biological terms, *cis* and *trans*. Let p_1 be the probability that one strand moves from state 0 to state 0 and the other moves from state 1 to state 1;

$$\begin{aligned} p_1 &= [(1-\theta) + \theta(1-r)][(1-\theta) + \theta r] \\ &= (1-\theta) + \theta^2 r(1-r). \end{aligned}$$

Similarly, the probability p_2 that one strand moves from state 0 to state 1 and the other moves from state 1 to state 0 is $p_2 = \theta^2 r(1-r)$. Then $cis = p_1/(p_1+p_2)$, while $trans = p_2/(p_1+p_2)$.

4.4 Initial Experiments in Map Construction

Simulated Annealing and Genetic Algorithms

In this section we describe our preliminary attempts to solve the second challenge in physical map construction, that of finding a good map efficiently. The methods described in this section are often the first approaches suggested for the problem of finding good maps, although we have had only limited success applying them to our large-scale mapping problem. Furthermore, it is instructive to explore the questions of when and how each technique fails, and how one might hope to overcome these problems in future experiments.

A natural approach is to use standard combinatorial optimization techniques with map likelihood as the objective function. We tried two such methods: genetic algorithms and simulated annealing. For both techniques, the search space is simply the space of all possible *orders* of the n markers. For each marker order, we use the EM algorithm (as described in Section 4.3.3) to determine the maximum-likelihood map distances associated with that marker order, and we evaluate only this map in our comparisons.

For our genetic algorithm[2] we initialized the population with a set of random orders. In each new generation we created a "child" by combining the orders of two "parents" chosen at random, with a mild bias in favor of more likely orders. We then removed either the child or one of the parents from the population, again chosen at random with probabilities dependent on the likelihoods of the orders. We repeated this process until the set of the ten most-likely elements in the population remained stable for a large number of generations (generally at least 100).

Our simulated annealing algorithm was based on the code in *Numerical Recipes in C* [88]. (The relevant chapter, by Press, *et al.*, also contains a good overview of simulated annealing.) New orders were derived from old ones by either swapping or reversing random substrings. We used the standard Metropolis algorithm to determine which

[2]See Goldberg [51] for a general discussion of genetic algorithms.

candidate orders to accept. We experimented with several different cooling schemes and initial conditions, but in all of our attempts we sought maximum-likelihood permutations of the markers.

Our first observation is that for small enough data sets (containing fewer than 10 markers), most optimization methods produce roughly equivalent, correct maps. For example, on a simulated data set of seven markers, the simulated-annealing algorithm successfully found the maximum-likelihood permutation. However, for so small a data set, it is possible to explore the entire space of possible orders! Near-exhaustive search becomes impossible as the number of markers increases, and once this happens our pure maximum-likelihood search methods begin to lose ground.

For groups of more than 20 or 30 markers, the genetic algorithm may converge to any of a number of likely orderings that bear little resemblance to the true order. Since each population is initialized at random, one can frequently evaluate the output by comparing the results of two or three runs on the same data and iterating for a longer time if the resulting maps are not similar. However, for groups of about 60 markers, we found that the output orders differed dramatically even when each experiment was allowed to run for several hours.

Figure 4.4 shows the results of running a genetic algorithm on a 200-marker simulated data set broken up into linkage groups of 10–75 markers each. (The simulator and the data are described in Section 4.5.4.) The graph plots the correct order of the markers (according to the simulated chromosome) along the x-axis. The y-axis shows the same markers in their position according to the best map found by the genetic algorithm. If the reconstructed map were completely correct, the graph would appear as a straight diagonal line (on either axis). Diagonal lines in the plot represent regions that were mapped correctly; scattered clusters of markers represent linked groups of markers that were not ordered well.

This figure shows a problem common to both the genetic algorithm and simulated annealing approaches: it is hard to tell whether the algorithm has converged to a correct or an incorrect order. The algorithm has used the same termination criteria for all of the linkage groups shown in the figure; some have converged to relatively good orders while others have not. Some small groups are incorrectly-ordered while other larger groups are ordered well, so group size does not seem to predict whether or not the algorithm will converge to a good map. Thus, it is difficult to choose adequate criteria for accepting the output of such algorithms unless information about the true order is already available.

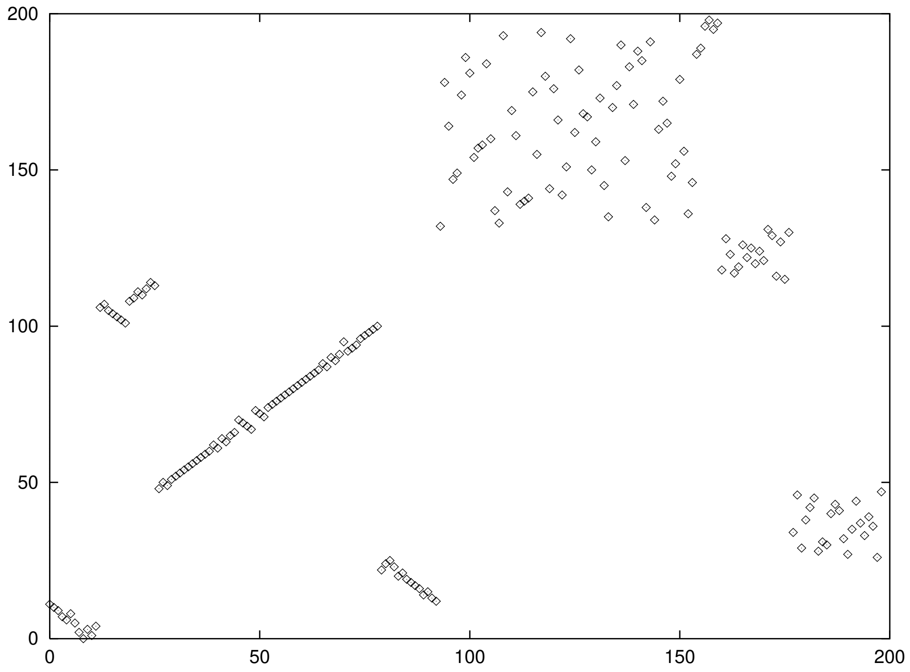


Figure 4.4: Genetic-algorithm order of a 200-marker simulated data set compared to the true order. The markers are first divided into groups with strong pairwise linkage. Straight diagonal lines indicate groups that are ordered correctly; scattered clusters of dots are groups that are not ordered well. All groups used the same criteria for declaring convergence.

Our experiments with simulated annealing yielded similar results but ran considerably more slowly. In one test, hoping to increase reliability, we chose a fairly slow cooling scheme and initialized the algorithm to a nearly-correct order of a 200-marker simulated data set. After running for four days, the algorithm had converged to an order with a log likelihood score of over 100 worse than that of the initial permutation (i.e., the ordering was 10^{100} times less likely to have produced the data)!

There are several possible explanations of why our large-scale mapping experiments with these methods have been so disappointing. The first is that the moves we used to derive new orders from old ones include swaps and reversals of substrings. However, it is possible that this set of moves is insufficient for solving large mapping problems efficiently. We might speed up the convergence process by including a move that intersperses two correctly-ordered substrings covering the same map region; for example, a single move that could change order 1357902468 into order 0123456789. It would be interesting to conduct additional experiments with simulated annealing methods that employ different sets of moves.

Another reason for the failure of our early experiments might be the nature of the multi-dimensional likelihood function that we are exploring. All known mapping techniques have a limited range of distances over which one can accurately order markers. For the data described in Section 4.7, we are unable to distinguish confidently two markers that are closer than about 1 Mb apart. We also have difficulty seeing accurate linkage between two markers that are more than 6 Mb apart, so different map orders with very different permutations of distant markers might have nearly identical likelihoods. Thus not only must we search a vast space of possible orders, but our optimization function has a great many local maxima consisting of marker orders that may bear no resemblance to the true order.

A third possible reason for our difficulties with pure maximum-likelihood searches may be due to the limits of the model. Our model makes very specific assumptions about the probabilities of breaks and errors. While we can estimate the overall error rates and break probabilities for the entire data set, these estimates may be inaccurate for specific regions of the data or for individual markers. Thus in some cases our algorithms find orders that are slightly *more* likely under our model than the true marker order!

However, our goal is to use the likelihood function not as an end in itself, but as a tool for discovering the truth. Thus we can bias our search algorithms by incorporating additional information about the true order into the objective function governing the search. This technique works quite well in practice; the algorithms described in the next two sections are examples of such approaches.

Gap Minimization

Finally, we reinforce the conclusions of Boehnke [29] that gap-minimization produces worse results than maximum-likelihood methods for radiation hybrid mapping. We used the gap-minimization software that Karp's group has written and successfully applied to STS-content mapping [3]. However, gap-minimization is particularly sensitive to false-positive errors, since each such error produces an additional gap. The STS-content data in Karp's project were carefully filtered to remove as many false-positive errors as possible, at the expense of adding some false-negative errors. While our group at the Genome Center performs every experiment twice to obtain relatively clean radiation hybrid data, there are still a number of errors present in our data sets.

Furthermore, radiation hybrid mapping differs from STS-content mapping in that there are a large number of inherent gaps. Each hybrid cell line retains many human DNA fragments, so even in the error-free case there are a large number of gaps in the matrix. Thus, perhaps it is not surprising that in our experiments for ordering groups of fifty or more markers, the maps produced by gap-minimization appeared indistinguishable from random permutations of the markers! We therefore agree with Boehnke's assessment that this technique is unlikely to be useful for developing good radiation hybrid maps.

4.5 Greedy Algorithms for Ordering Markers

In this section we describe a greedy approach to the problem of finding good marker orders. We first define a simple greedy algorithm and then show how to use the basic method as a subroutine for a faster, more accurate algorithm.

4.5.1 The Basic-Greedy Algorithm

The Basic-Greedy algorithm inserts markers one by one into a growing map. The initial map consists of a pair of markers. At each step, the next marker to insert may be chosen at random or may depend on linkage to other previously-placed markers. The algorithm computes the likelihood of the current order with the new marker inserted in each possible interval. The marker is then permanently inserted into the position yielding the new order with the highest likelihood.

```
Basic-Greedy(M):                                  {M = { all markers to be mapped } }
map = Initialize-Map(m1, m2)                      { start with any 2 markers in map }
M := M\{m1, m2}                                   { remove m1, m2 from M }
for each m ∈ M
    for i = 0 to sizeof(map)                      {for each interval in map }
        map_i = map with m inserted in interval i
    map = map_i with maximum likelihood
return map
```

For example, if the map starts with markers A and B, there are three intervals into which a third marker C could be inserted, producing the three maps CAB, ACB, and ABC. (Note that we are unable to distinguish map ABC from map CBA, since they have the same likelihood. Thus we need only consider three possible permutations of three markers, not six.) The basic greedy algorithm would evaluate all three maps using the most-likely map distances as determined by the EM algorithm. It would choose the most likely of these as the current map and continue adding new markers into this new map.

This approach works well on small groups of markers where all markers are within about 10 Mb of one another. Over larger distances, the limits of the mapping technique reduce our confidence in the orders produced.

One general problem with the basic greedy approach, however, is that if a mistake is made early in the process, it can cause a lot of trouble later on. Mistakes may be due to several factors. If we try to place a marker that belongs too far away from any markers already in the map, the algorithm will place the marker off one end of the map, but it might be the wrong end. If we try to add a marker too close to one already in the map, there will be two positions that are nearly-equally good, one on either side of the previously-placed marker. And if we try to place a marker with many errors, it may be advantageous for the algorithm to place the marker in an interval where it doesn't really belong.

We can improve our results somewhat by adding markers in order of strongest pairwise linkage to markers already mapped. This process avoids the problem of placing markers too far from previously-mapped ones. Another improvement, which handles markers too close to those already mapped, is to defer the addition of a marker if there are two or more intervals for it that are of nearly-identical likelihood. We reserve all markers with this property and insert them last, so that their addition into an incorrect position won't cause later markers to be placed incorrectly.

Even with these improvements, the basic greedy method runs into trouble when asked to map several hundred markers. At first glance the algorithm appears to have an $O(n^2)$ running time, since it requires $O(n^2)$ likelihood calculations of maps. However, this analysis ignores the fact that evaluating a map is not a constant-time operation. The time for the EM algorithm to converge is proportional to the size of the map as well. Thus it pays to break the problem into smaller groups for two reasons: our results are more accurate for groups of markers that are near each other in the true map, and working with smaller groups of markers dramatically decreases the running time.

4.5.2 The Parallel-Greedy Algorithm

We have implemented a new algorithm that forms greedily-ordered subgroups and then attempts to join the subgroups together correctly. While it is not truly a parallel algorithm, it is called Parallel-Greedy since all the subgroups are grown at the same time. The advantage of this new method is that it biases the search for good maps by ensuring that tightly-linked pairs of markers remain near each other in the final order. This constraint helps combat the problems of the methods described in Section 4.4 and of the Basic-Greedy algorithm.

Parallel-Greedy(M): $\{M = \{$ all markers to be mapped$\}$ $\}$
$S :=$ **Initialize-Subgroups**(M)
$A := \cup_{s \in S} s$ $\{$ A = all markers in any subgroup in S $\}$
$M := M \backslash A$ $\{$ M = remaining markers $\}$
while M is not empty
  do $m :=$ marker in M most closely linked to any $m' \in A$
    $s :=$ subgroup of S containing m'
    Insert m into s with **Basic-Greedy**
    $M := M \backslash \{m\}$
    Try to merge s with all other subgroups using **Group-Merge**
Merge all remaining subgroups using **Group-Merge** at a lower lod threshold
return final merged group

Initialize-Subgroups(M):
$L :=$ list of all pairs of markers in M, sorted by decreasing pairwise lod score
$k := 1$
while L is not empty
  do $\langle i, j \rangle :=$ the first pair in L $\{$ the remaining pair with highest lod score $\}$
    if pairwise-lod$(i, j) <$ *threshold* $\{$ all pairs left have lod...
      then return $S = \{s_\ell \mid 1 \leq \ell \leq k-1\}$ ...scores below threshold $\}$
    else $s_k :=$ map of markers i and j at distance θ_{opt}
      $k := k + 1$
      remove all pairs containing i or j from L
return $S = \{s_k\}$

For the new algorithm we first initialize the subgroups to contain only pairs of tightly-linked markers. At each step, we greedily add a marker to its most tightly-linked subgroup. Each time a new marker is inserted into a subgroup, there is a chance that the evidence for linkage between the two subgroups has increased to the point where it is statistically significant. Thus, after each insertion, we try to join the newly augmented subgroup with the others to see if there is now strong enough evidence for linkage. We also defer insertion of any marker if there are two nearly-equally-likely positions for the marker. At the end, we greedily insert all deferred markers into their best positions.

Our algorithm often needs to merge a number of groups of already-ordered markers.

To do this we use the procedure Group-Merge, which calculates multipoint lod scores for pairs of groups. The groups m_1 and m_2 are merged end to end in each of the four possible combinations:

$$\begin{array}{rl} m_1 & m_2 \\ m_1 & \text{reverse}(m_2) \\ \text{reverse}(m_1) & m_2 \\ \text{reverse}(m_1) & \text{reverse}(m_2) \end{array}$$

For each combination, the likelihood score for the two groups linked at the optimal distance is compared to the score when the groups are separated by an infinite distance ($\theta = 1.0$). If the lod score comparing these two maps exceeds a certain threshold, the two groups are joined; otherwise they remain as separate groups.

Group-Merge($group_1, group_2$):
for each orientation (m_1, m_2) of the two groups
    linked := map joining m_1 and m_2 at optimal distance
    unlinked := map joining m_1 and m_2 at infinite distance ($\theta = 1.0$)
    if lod score (*linked* vs. *unlinked*) $>$ *threshold*
        then return *linked*
return $group_1, group_2$

4.5.3 A Greedy Strategy for Large-Scale Mapping

We have incorporated the Parallel-Greedy algorithm into a general method for mapping large data sets. At the top level, our strategy consists of four steps. The first step involves breaking the data set into strong linkage groups of fewer than 100 markers each. Linkage groups are formed by computing the transitive closure of pairwise linkage at a given lod threshold. Markers that show high pairwise linkage to one another are likely to be near each other on the chromosome; the algorithm that follows places all such markers together in the final map.

Next, we greedily order the markers within each linkage group using either the Basic-Greedy algorithm or the Parallel-Greedy algorithm. In the third step, we attempt to improve the order of each linkage group using a local permutation method (the "ripple" algorithm described in Section 4.6.2). Finally, the ordered groups are linked together with the Group-Merge procedure to form the final map. At this stage, the lod threshold for merging can be decreased gradually until all groups are merged together.

4.5.4 Results

We tested our greedy strategy on both real and simulated data. In this section we describe the results.

Chromosome 4

We obtained haploid data for 234 markers on Chromosome 4, courtesy of David Cox. The markers were screened by PCR assay against the Stanford radiation hybrid panel of 85 hybrid cell lines. All assays were duplicated and any discrepancies were treated as missing data, so the error rate for this data set is rather low. Cox used version 1.0 of RHMAP [32] to generate a maximum-likelihood order of the markers.

Using the same data, we generated a map of their markers blindly (without reference to Cox's map) and then compared our ordering to theirs. It took our software about 3 hours to obtain the final order.

A graph plotting our marker order against Cox's is shown in Figure 4.5a. A straight diagonal line would represent complete agreement between our orders. When markers whose relative position cannot be determined from the data are placed into bins, the two orderings become even more similar (as shown in Figure 4.5b). Under our likelihood model, our map beats Cox's map by a lod score of 0.7.

While the maps produced are quite similar, there are a few discrepancies beyond those caused by the arbitrary ordering of markers in the same bins. These discrepancies are most likely due to several differences in our models. One difference is that our method used a diploid model to evaluate the haploid data, since these tests were performed before we added the haploid option to our software.

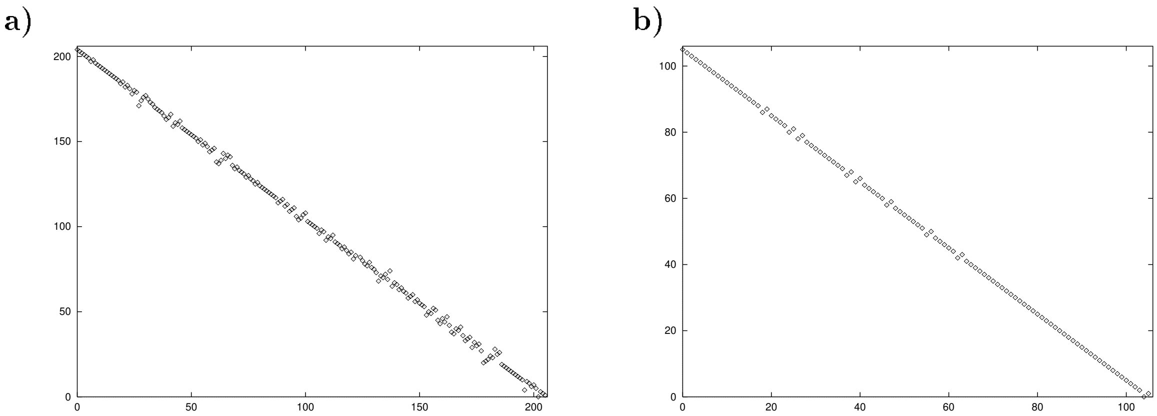


Figure 4.5: a) Comparison of our greedy order of 234 markers on Chromosome 4 to David Cox's order of the same data. b) The same data, with markers that are essentially indistinguishable – i.e., the estimated distance between them is 0 cR – placed into bins. This comparison of our bins to Cox's bins shows that the two orders agree very closely.

Another difference is the presence of errors in the data. RHMAP does not account for errors at all. We estimated the average false positive rate to be about 1% and the false negative rate to be about 3%. We also assumed that if the results of a marker/hybrid assay were "missing" (i.e., the duplicated experiments yielded discrepant results), there was a 50% chance that the true result was positive. We have since discovered that the vast majority of the missing data in these experiments are actually weak positives, so this probability is closer to 95%. Our map is likely to be slightly worse because of this incorrect assumption.

In constructing our order, we assumed that the retention frequency was constant across all hybrids. This assumption is probably incorrect, but it is one that Cox made as well. Our software can handle variable retention frequencies, but we used a constant retention frequency to correspond more closely to the RHMAP model.

Simulated Diploid Data

To further test our algorithms, we generated a sample diploid data set using a program called Groupsim. Groupsim generates linkage groups and simulated radiation hybrid data by placing markers according to a Poisson distribution. The program includes random errors in its simulated data according to user-defined error rates. Other user-adjustable parameters include the retention frequency (constant or per-hybrid), the mean distance between markers and the mean fragment length.

Our test data set contained 200 markers in a single linkage group, screened against a hypothetical panel of 85 hybrids. We assumed a mean distance of 500kb between the markers and a mean fragment length of 3kb. These values correspond roughly to the mean fragment length and spacing of the Chromosome 4 data. The simulated data had a retention frequency of about 15%, which is typical of the Chromosome 4 data as well. (In contrast, the retention frequency for the data used to build the maps described in Section 4.7 is about 32%.) We used a 2% false-positive rate and 10% false negative rate, much higher error rates than we would expect for duplicated assays on real data.

Our algorithm ran to completion in 1 hour, 41 minutes on a Dec Alpha 3000. The final ordering we obtained is compared to the "true" order (that of the simulated input data) in Figure 4.6. A straight diagonal line would correspond to a perfect ordering of the data. Almost all markers were placed at most three steps away from their position in the true order. The greedy order had a lod score of 3.4 less than the true order.

In a set of 200 markers with only 85 hybrids and a substantial noise rate, we would expect to see about one marker that, by chance, looks more like some other region of the map than like its neighbors. As expected, a single marker appears in the wrong section of the map. Such deceptive markers would be observed in real data, but their presence could be detected by other means (i.e., strong linkage to two distinct regions of the map; assignment to a different chromosome than other markers in the same linkage group, etc.).

Comparison with RHMAP

We also compared the efficiency of our algorithm with that of version 1 of RHMAP [32]. (Tests with version 2, performed later, yielded similar maps but ran even more slowly.) We used the maximum-likelihood option and compared our results to RHMAP's stepwise

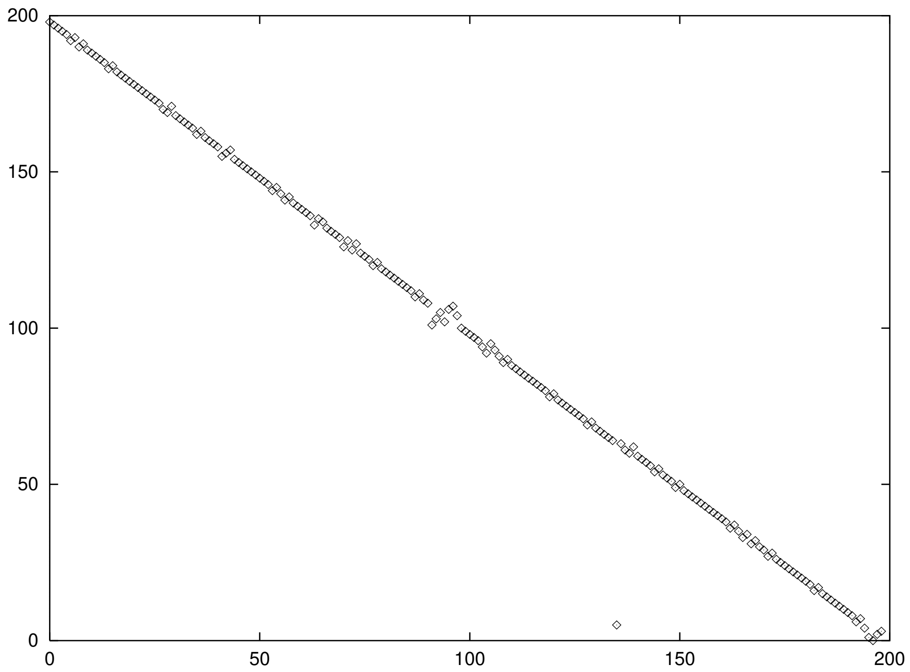


Figure 4.6: Test of our greedy mapping strategy on a simulated chromosome of 200 markers. The x-axis represents markers in the order output by the greedy algorithm; the y-axis lists them in their correct order.

ordering method (the fastest option, and the one most like our greedy approach). We assumed equal retention probabilities for all fragments.

Due to RHMAP's computational limitations, we were only able to obtain results for small linkage groups. We tested the algorithm on two linkage groups from our simulated diploid data set; one of 17 markers and one of 22. The maps produced by RHMAP for these groups were remarkably similar to those produced by our greedy algorithm, which ran in a fraction of the time taken by RHMAP. Table 4.1 compares the running time of our method to that of RHMAP.

The maps for the 22 marker linkage group are shown in Figure 4.7. The best locus orders produced by the two algorithms are virtually identical; the difference in each case is only a single pair of swapped markers producing a negligible difference in likelihood under our model. The only substantial differences between the two maps, aside from the running time of the algorithms, are the estimated map distances. Our algorithm seems

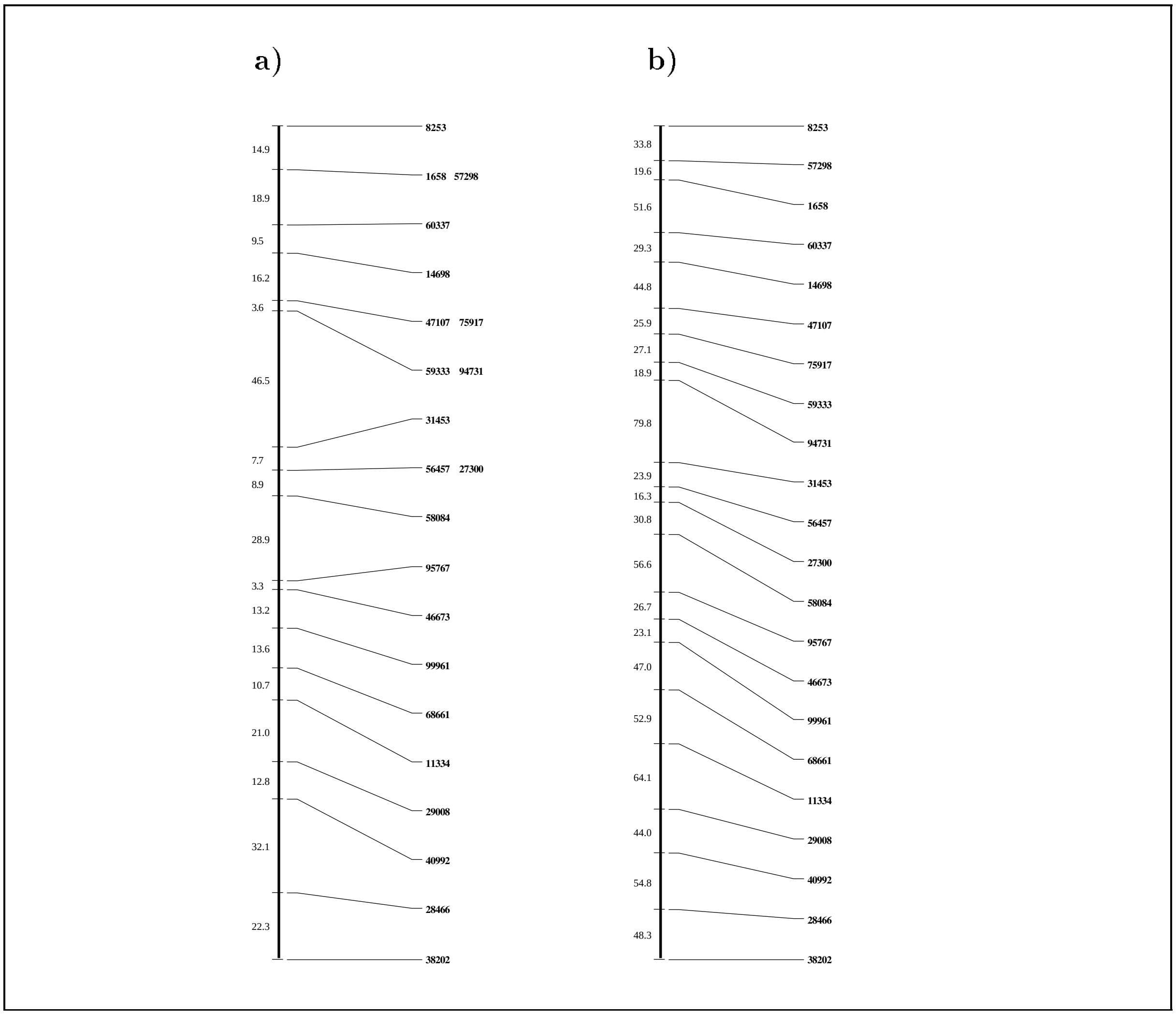


Figure 4.7: Radiation hybrid maps (with distances in cR) of 22 linked markers from a 200-marker simulated data set. a) The map produced by our greedy algorithm. b) The map produced by RHMAP. For groups this small, the two marker orders are quite similar. However, the distances between markers differ, perhaps because RHMAP does not model errors or diploid data.

Running Times

number of markers	RHMAP	Greedy
17	4 hours, 54 mins	1 min, 37 sec
22	8 hours, 50 mins	2 mins, 7 sec
63	$>$ 10 days	1 hour, 3 mins
200	??	1 hour, 41 mins

Table 4.1: This table compares the running times of RHMAP to those of our greedy mapping strategy for several different-sized groups of markers. All benchmarks were run on a Dec Alpha 3000 workstation. RHMAP is written in optimized FORTRAN 77, while our software is written in a combination of Perl and optimized C.

to underestimate the map size somewhat, while RHMAP overestimates it. This makes some sense, since RHMAP does not model errors in the data.

We suspect that due to our error model, there would be more noticeable differences between our maps and RHMAP's on larger data sets. In an attempt to test this hypothesis we ran RHMAP on a group of 63 markers, but we gave up after it ran for over 10 days without halting. It would be interesting to continue trying to test this hypothesis by performing additional experiments on intermediate-sized data sets.

4.6 Framework and Placement Maps

One disadvantage of all the algorithms described above is that they provide no indication of the degree of confidence in the placement of each individual marker. Figure 4.6 illustrates the need for such confidence estimates. While the map shown in the figure is generally quite accurate, one marker is placed in a grossly incorrect position. A researcher using the map, however, would have no indication of which markers are placed with high confidence and which might be placed incorrectly. The method described in this section builds maps with inherent confidence estimates.

Our maps consist of two sets of markers, indicating different degrees of confidence in the markers' map positions. A *framework map* is a set of markers whose relative order is known with reasonable certainty. We add as many markers as possible to the framework maps. Those markers that cannot be mapped with sufficient confidence, perhaps because they are too close to markers already in the framework, are placed into bins relative to the framework markers. These binned markers comprise a *placement map*. Placed markers cannot be ordered confidently relative to one another using the available radiation hybrid data. Instead, placement maps indicate the markers' approximate locations with respect to the framework and list all alternative placements that are nearly equally likely.

To build framework maps we start with a sparse, correct framework that spans the chromosome to be mapped. Building this framework is non-trivial; we discuss its construction from radiation hybrid data in Section 4.6.1. However, one could imagine finding a sparse framework of 10 or 20 correctly-ordered markers on a chromosome from another source such as genetic linkage maps. Given a good initial framework map, we incrementally add selected markers to nearly-proper places in the framework using a simple greedy algorithm to be described below. We can then use a local permutation algorithm (such as the "ripple" algorithm described in Section 4.6.2) to improve the map and to test its accuracy. Markers that fail to satisfy the criteria for admission to the framework map are then added to the placement map.

4.6.1 Building Initial Frameworks

Our algorithms for finding initial frameworks begin by examining all triples of markers. Three markers in their most likely permutation form a *strongly-ordered triple* if their order is more likely than that of any other permutation by a fixed lod threshold, and if the estimated distances between the three markers are within a certain range. (We generally look for triples with a lod score of at least 3, and with inter-marker distances ranging from 5 to 20 cR.) The markers in a strongly-ordered triple are quite likely to be in their correct map order with respect to one another. Thus it may be possible to integrate several strongly-ordered triples to form longer, correctly-ordered sparse framework maps.

We have designed algorithms that search for the longest framework orders supported by the triples. To see why we want the longest possible order, consider the case of an initial framework that spans only half of a chromosome. Many markers to be mapped might belong to the part of the chromosome not covered by the framework. These markers would be poorly linked to all of the framework markers. Such markers would be placed off some end of the framework map, but not necessarily in their correct location. Any errors introduced at this early stage are likely to propagate throughout the map construction process. Thus it is critical to start with a framework map that spans the entire region.

There are two obstacles to assembling frameworks from strongly-ordered triples. First, a small but significant fraction of the triples (generally less than 5% for a lod threshold above 3.0) may be ordered incorrectly. Second, the orientation of the triples (A-B-C vs. C-B-A) is unknown, increasing the computational difficulty of the problem. Without some knowledge of which triples are incorrect, we cannot guarantee finding a correct initial framework. However, we have developed two algorithms that generate good candidate frameworks.

Each of these algorithms creates a directed acyclic graph (DAG) based on the triples and then finds the longest path in that DAG. Finding the longest path in a DAG with V vertices and E edges requires $O(V + E)$ steps [40]; in our case this is $O(n)$ steps, where n is the number of good triples. In contrast, finding the longest path in a general graph is NP-complete.

The first algorithm relies on all partial order information available from the triples to assemble a path. For example, the triples A-B-C and A-C-D would be combined into A-B-C-D by this algorithm. The vertices of the graph correspond to all the marker names in the list of good triples (each marker gets one vertex, no matter how many triples it appears in). The edges impose a partial order on the markers defined by the strongly-ordered triples. If all the triples were listed in their correct forward/reverse orientation, it would be possible to place all edges on the graph in just one pass through the data. However, since some triples are reversed in orientation with respect to others, we need to be a bit more careful.

The edges for the first triple may be added in arbitrary direction. For example, if the first triple is A-B-C, we add the edges $A \rightarrow B$ and $B \rightarrow C$ to the graph. The algorithm then makes up to n passes through the list of remaining triples. For each triple, the algorithm adds edges for that triple if there is exactly one consistent orientation for that triple in the current graph. This means that in one orientation, adding the edges corresponding to the triple would cause a cycle in the graph, while in the other (correct) orientation it would not. If both orientations are consistent, the triple is deferred until the next pass. If neither is consistent, some triple must be incorrectly ordered. The algorithm then prints a warning message and discards the current triple. After all triples have been added to the graph (or discarded), the algorithm simply finds the longest path in the directed acyclic graph. Figure 4.8 illustrates a sample run of this algorithm.

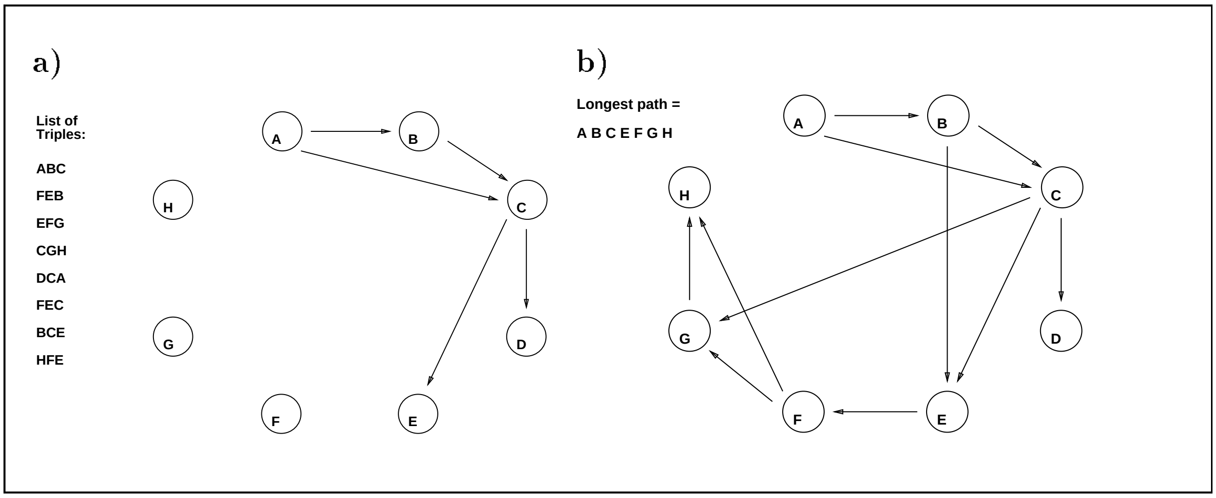


Figure 4.8: The first algorithm for finding a long path from strongly-ordered triples. The triples are considered in the order listed. a) The graph initially contains edges $A \rightarrow B$ and $B \rightarrow C$. In the first round, the edges for triples DCA and BCE can be added, since there is only one orientation for each triple consistent with the edges already in the graph. b) In the second round, the edges corresponding to the rest of the triples are added in the order in which they appear in the list. The longest path in the resulting graph is a candidate framework map.

This algorithm uses all partial ordering information available; consequently, the paths output may contain pairs of adjacent markers that are adjacent in only one triple in the input data. These markers may actually be too close together for both to appear in the same framework map. Thus, we sometimes prefer using a stricter algorithm for finding long paths of framework markers.

This second, more selective method seeks the longest path of overlapping triples such that, for all adjacent triples in the path, the last two markers of the left triple are the first two markers of the right triple. For example, triples A-B-C and B-C-D would be merged to form A-B-C-D, but triples A-B-C and A-C-D would not. Because this criterion is very stringent, the paths found using this method tend to be shorter and more reliable than paths found using the first method.

To implement this method, we build a graph whose vertices correspond to triples rather than individual markers. We first assign undirected edges between any two triples whose end-most markers overlap. Next, the algorithm assigns an arbitrary direction to one of the graph edges. It then performs a breadth-first search of the undirected graph to determine the directions of the remaining graph edges (so that they are consistent with the initial edge, and so that the graph remains acyclic). Edges that form cycles are discarded. Finally, the algorithm seeks the longest path through the directed graph of triples. This algorithm is illustrated in figure 4.9.

The danger of both of these methods is that if an incorrectly-ordered triple is added to the map at an inconvenient time, the resulting path could be incorrectly-ordered. Our solution to this problem is to run the algorithm several times on different random permutations of the input file. We have found that in most cases, the incorrectly-ordered triples appear inconsistent with the rest of the graph and are discarded. Occasionally, a bad triple is added to the graph early; in that case, many other triples are discarded and the resulting path tends to be very short. Thus, we have generally been able to find good frameworks using these methods.

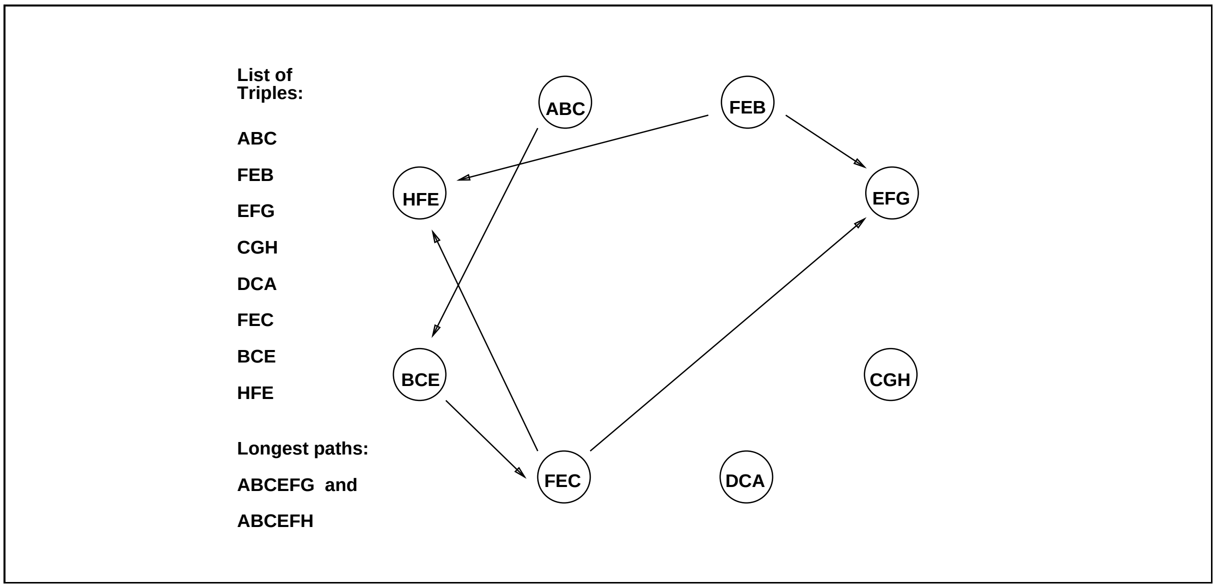


Figure 4.9: The second, more stringent path-finding algorithm. The figure shows the final graph for the same list of triples as in Figure 4-8. Triples are only linked if they share two adjacent markers in a way that allows for path extension. (For example, FEB and EFG share an edge, but EFG and HFE do not, since the latter two cannot be combined to form a longer path.) Note that the resulting candidate frameworks differ from the one produced by the previous algorithm.

4.6.2 Testing Initial Frameworks

The candidate frameworks produced by these algorithms are subjected to a great deal of testing before they are accepted as initial frameworks. First, the most-likely map distances are determined for the given candidate marker order. The maps are examined to ensure that no pair of adjacent markers are too close together or too far apart. The raw data vectors for the map are examined as well, to see if any obvious gaps or out-of-place markers appear.

The candidate is then subjected to the “ripple” test. In this test, a sliding window k markers wide (for some constant k) is moved along the map, and the map's likelihood is evaluated for each possible permutation of the markers in the window, with the rest of the map remaining constant. If the initial framework order is substantially more likely than any other permutation found this way then the candidate passes the test. If a candidate fails the ripple test, it may contain two markers that are too close together or too far apart to be ordered reliably or a small set of markers that are ordered incorrectly. The candidate is then adjusted by hand to fix the problem, if possible.

Candidate frameworks are also subjected to the "pull-out" test, in which one marker at a time is removed from the map. The test attempts to insert the one removed marker into each possible interval in the remaining map. If the best place for the marker is in some interval other than the one it came from, or if the lod score favoring the correct interval over the next-best one isn't high enough, the candidate fails the test.

Finally we test the maps by trying to add markers to the frameworks greedily, using the methods described in the next section. Any incorrect ordering in an initial framework is likely to produce a map interval that rejects the placement of future markers. These gaps are obvious upon inspection of the expanded maps. During our genome-wide mapping efforts we detected several errors in this fashion, fixed the initial frameworks as necessary, and re-built the maps. Since constructing a correct initial framework is by far the most time-intensive part of the map-building process, this sort of correction is not too expensive.

4.6.3 Growing LOD-k Frameworks

Once a good initial framework has been found, reliably adding markers to it is easy. We call the resulting framework a LOD-k framework map, for some constant k. We consider as a candidate any marker in the database that shows strong pairwise linkage to some marker already in the framework. We then subject each candidate marker to two tests. First, we use the Basic-Greedy algorithm (from Section 4.5) to find the most-likely interval for the new marker. If the lod score comparing the map with the new marker in its best interval to the map with the new marker in the next-best interval is at least k, then the best map is considered as a tentative new framework. This map is then subjected to the ripple test. If it passes the test at a lod threshold of k, the tentative order is promoted to become the new framework, and the process continues with the next marker.

Adding new markers to a framework changes the likelihoods for inserting other markers. Therefore, the process of growing a framework may be repeated with markers rejected in a previous iteration until no new markers are accepted.

4.6.4 Placement Maps

In any mapping project some markers will not be added to the framework, either because they are too close to markers already in the framework or because they have an above average noise rate. These markers are binned in a *placement map*. A placement map consists of a framework map and a set of markers positioned relative to the framework. Each placed marker may map into several framework intervals with nearly-equal likelihood; all such possibilities are represented in the placement map. Markers are added to the placement map with the same greedy test used to choose candidate framework markers. However, the algorithm not only records the best placement interval but also all placement intervals within lod 3.0 of the best one. If the acceptable placement intervals are not all adjacent in the framework (for example, if a marker's best placement is off one end of the map but its next-best placement is off the other end), the marker is likely to have a high error rate.

A marker may be rejected from the placement map if it does not exhibit strong pairwise linkage to any framework marker, or if its best placement is too far from any neighboring framework marker. The first criterion removes markers that are not truly linked to the chromosome being mapped. The second criterion is designed to prevent error-prone markers from sliding to the end of the chromosome. This artifact of our technique occurs when a marker has enough errors that placing the marker in its correct interval implies a large number of obligate breaks. If the marker's correct position is near the end of a chromosome, there may be a higher likelihood score for placing the marker off the end of the map or into the large gap that often occurs at the centromere. However, since the marker does not really belong off the end of the map, the optimal distance between the new marker and the rest of the framework is quite large. Thus we reject all such placements, since the majority of them are incorrect.

This criterion points out the importance of having a framework map that spans the entire chromosome. If one end of the chromosome is not represented, many markers will place too far off the end for acceptance to the map. For example, at one point we noticed a lack of framework markers on the top of Chromosome 21 when a number of markers had placed off the end there; adding a new framework marker fixed the problem.

Figure 4.10 shows part of a placement map and data for several markers on Chromosome 16. In this figure, the first column consists of marker names and the second column lists the estimated distances (in cR) between adjacent markers. Two markers with virtually identical data vectors, such as EST157352 and MR14121, are placed at a distance of zero. An “F” in the third column indicates that the marker in question is part of the framework map, while a “P” indicates that it is a placement marker. Each placed marker is listed in its most likely position, and the number to the right of the “P” indicates the lod score between the best and next-best placements for that marker. Marker AFM340YE5 is placed at a very low lod score, because its data vector is nearly identical to that of the adjacent framework marker, and thus it could in fact belong on either side of that marker. However, EST151329 is placed with a lod score of greater than 3.0, indicating that the marker is at least $10^3 = 1000$ times more likely to belong in the interval between AFM214ZG5 and MR7804 than in any other framework interval. Such a marker is a candidate for promotion to the framework.

4.6.5 Tests with Simulated Data

We tested our algorithms for building framework and placement maps on simulated data comparable to the real data used to construct our genome-wide maps.

The data were generated by the simulator Groupsim, which is described in Section 4.5. Our data set consisted of 200 markers on a single chromosome, with an average spacing of 500 kb between markers. The mean fragment size was 10 Mb (corresponding to the

MR10702	2.20	F	1000100100010000011000100100100000120010
EST156791	2.30	P0.29	1000100100010000011000100100100000110010
UTR-05569	1.82	F	1000100101010000011000100100100000110010
EST157352	0.00	P1.26	1100100101010000011000100100100000110010
MR14121	1.82	P1.33	1100100102010000011000100100100000110010
UTR-04543	0.26	P>3.00	1100100101010000011000100100100000110010
AFM214ZG5	2.67	F	1100100101010000011000100100000000110010
EST333861	0.52	P1.34	1100100101000000011000000100000000110010
EST181036	0.97	P2.73	1100100101010000011000000100000000210010
EST151329	0.07	P>3.00	1100100101010100111010000100010000110010
A002K14	1.02	P1.05	1100101101010000111000000100000000210010
MR7804	1.81	F	1100100101010000111000000100000000110010
AFM112XH2	4.09	P1.74	1100100101000000111010000100000000110010
AFM340YE5	0.00	P0.03	1100100101010000111000000100000000100010
MH1082	0.00	F	1100100101010000111000000100000000110010

Figure 4.10: A placement map of several markers on Chromosome 16. The first column lists the marker names, the second column lists distances between adjacent markers, and the third column designates markers as framework or placement markers, listing the confidence (lod score) for each placement. The data vectors for each marker are shown as well.

estimated mean fragment size of the Genebridge 4 radiation hybrid panel); the average retention frequency was 30.3%. The average false-positive and false-negative error rates were .4% and .2% respectively, corresponding to the expected error rates for real data in which each PCR assay has been performed twice, and any discrepant results recorded as missing data.

Of the 200 markers, we chose a random subset of 100 and computed the likelihoods of all triples of these markers in about 1.5 hours. The more conservative initial-framework algorithm found a framework of 33 markers after running for 30 minutes. After some brief testing and visual examination, this framework was expanded at lod 3.0 using the greedy method described in Section 4.6.3. This phase, in which all 200 markers were considered as possible framework markers, ran for approximately 2.5 hours at a lowered priority (+10, on a scale of 0 (highest) to +20 (lowest)). The resulting framework map was examined and tested with the pull-out and ripple tests, but was not altered in any way. This map contained 89 markers that were in perfect order with respect to the "true" order of the simulated data; a graph comparing the framework order and the true order of the same markers is shown in Figure 4.11a.

a) b)

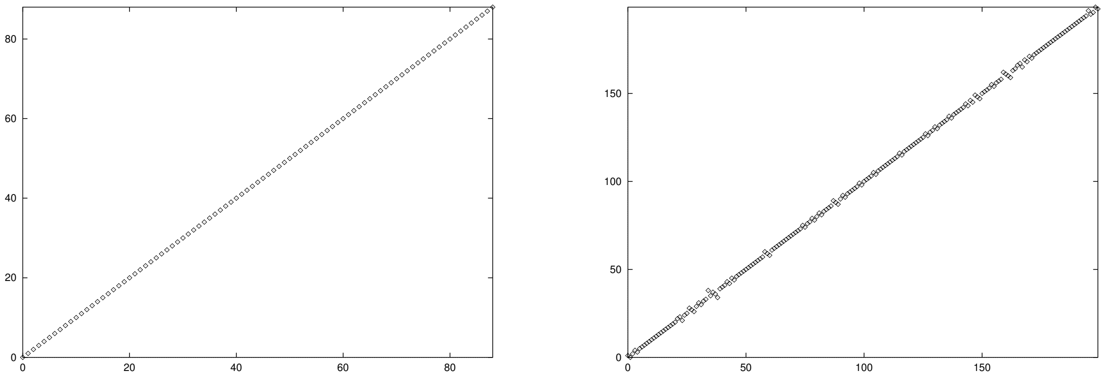


Figure 4.11: Test of the framework and placement map construction algorithms on a 200-marker simulated data set. a) Comparison of our 89-marker framework map to the correct order of the same markers. b) Comparison of the 200-marker placement map (including the framework map from a) to the "true" order of the markers along the simulated chromosome. Each placement marker is shown in its most-likely map position; other positions may be nearly-equally likely.

Finally, we created a placement map from the 89-marker framework by attempting to place all the remaining markers. All of them were accepted into the placement map; this procedure ran for about 45 minutes. Figure 4.11b compares the placement map to the correct order of the markers. Since markers in a placement map are not ordered with respect to each other but are simply placed into bins, their map order may appear incorrect.

Thus, within five and a half hours we were able to complete the entire process of creating an accurate map of a chromosome containing 200 markers. The map construction was remarkably easy; while we examined the map output at each stage for problematic regions, we found none. In practice, map construction is rarely so simple, since real data do not always exactly follow the assumptions of our model. Our practical experiments in map construction are described in the next section.

4.7 Results

4.7.1 RHMAPPER: Interactive Map Construction Software

We have written an interactive software package called RHMAPPER [104, 106] that incorporates the algorithms described here for the construction of genome-wide radiation hybrid maps. The package includes facilities to automatically detect and flag errors in the data, allowing error-correction during map assembly. In addition to the framework-construction and testing algorithms described above, the software includes several tools for evaluating and manipulating groups of markers, and displaying, testing, and editing maps.

RHMAPPER is written in C and Perl for the UNIX operating system. The package is built on a client/server architecture, allowing the user to modify the map-building routines in an incremental and experimental manner. The server is written in optimized C and is responsible for the computationally intensive maximum likelihood calculations. The remainder of the software consists of a series of Perl scripts tied together by a front end that is essentially a Perl interpreter. Thus one can type in commands to be evaluated immediately or create new subroutines and macros using any of Perl's control constructs and functions. RHMAPPER can also run in batch mode.

The software and documentation are available free of charge, under a license that allows unlimited redistribution, at `http://www-genome.wi.mit.edu/ftp/pub/software/rhmapper`, or via anonymous ftp from `ftp-genome.wi.mit.edu`, in directory `/pub/software/rhmapper`.

4.7.2 A Genome-Wide Radiation Hybrid Map

Practical Map Construction Methods

RHMAPPER has been tested extensively at the Whitehead Institute Center for Genome Research, where we have used it to build the first human genome-wide radiation hybrid maps. The laboratory work for this project represents the combined effort of a large team of biologists. To build the map, our group initially screened a total of 6,795 STSs against the Genebridge 4 radiation hybrid panel, which consists of 91 human/hamster hybrid cell lines. The average fragment length in the panel is about 10 Mb and the average retention rate per hybrid is about 32%. All assays were performed twice to improve accuracy; discrepant results were marked as unknown for mapping purposes. Chromosomal assignments for 5134 markers were determined by somatic-cell hybridization or genetic map position.

Once we had obtained data for roughly the first two thousand markers, we segregated the assigned markers by chromosome and built initial framework maps for each chromosome using only markers known to be on the chromosome. Although this approach excludes many potentially good framework markers, it has two advantages. First, it narrows down the number of markers considerably. This is crucial, since to find strongly-ordered triples our algorithms must examine nearly all combinations of three markers in the data set. This cubic algorithm can take a number of hours for groups of several hundred markers (running on a DEC Alpha 3000 workstation). Second, this criterion increases the confidence we have in the resulting framework maps, since each marker in a framework not only exhibits strong radiation-hybrid linkage to other framework markers, but is placed on the correct chromosome by independent experimental evidence.

We built the initial framework maps using the methods described in Section 4.6.1. During this process, we noticed that adjacent markers on opposite sides of the centromere show very low pairwise linkage. Therefore, we often constructed separate framework maps for each chromosome arm. We then relied on the genetic map positions of markers as well as on radiation-hybrid linkage to correctly order the two arms of each framework.

When constructing candidate initial framework maps, we occasionally encountered "circular chromosomes" – candidate frameworks whose ends were linked together. While it has been suggested that these cycles were caused by the prevalence of repeated DNA, we found that increasing the lod-threshold criterion for strongly-ordered triples eliminated the problem.

Once the initial frameworks were formed and tested we greedily added markers to them, again restricting our search to chromosomally-assigned markers. At first, we required that all frameworks pass the ripple test at a lod threshold of 3.0. In a later pass, we reduced this threshold to 2.5 to increase the number of framework markers. Further attempts at lod-threshold reductions resulted in a marked decrease in map quality.

Next, we assigned all the remaining markers to placement maps. For the 1,661 markers whose chromosomal assignment was still unknown, we used YAC contig information and RH linkage to determine a unique chromosomal assignment whenever possible. There are 187 markers in the database for which we were unable to determine a unique chromosomal assignment by any method. (Many of these have shown linkage to more than one chromosome.)

We then engaged in a round of error-detection and correction. We generated a list of the marker/hybrid assays most likely to be incorrect and repeated these experiments in the laboratory. We then incorporated the corrected data into the database. Since some frameworks failed to satisfy the ripple test using the new data, we adjusted those framework maps by hand, generally by removing one or two improperly-placed markers. We then re-built the placement maps using the corrected data.

Finally, we screened an additional 5,693 markers, most of these expressed sequences whose genomic location was unknown. We expanded and tested the framework maps again, and we added a total of 5,164 new markers to the placement maps.

Description of the Map

Our maps consist of 11,357 markers, 1,638 of them on lod-2.5 framework maps, spanning chromosomes 1..22 and X.[3] The markers are spaced at an average distance of 264 kb. Table 4.2 lists the number of mapped markers as of May 6, 1996 and the map sizes for each chromosome.

A sample map is shown in Figure 4.12. The markers named in bold are framework markers; placement map markers are shown in their most likely positions. All of our maps are available on the Genome Center web server, at

`http://www-genome.wi.mit.edu/cgi-bin/contig/phys_map.`

This page includes a confidential facility for electronically submitting markers to be located on our radiation hybrid maps. The page also displays our latest RH maps by chromosome and allows the user to click on individual markers to learn more about the data placing them onto the map.

Coverage and Accuracy

We estimate that the maps cover 99% of the (female) genome. To derive estimates of coverage, we attempted to place 100 random STSs onto our maps. Using RH linkage alone, we assigned all 100 markers to the correct chromosome. We were able to position all 100 on the map with pairwise linkage to at least two framework markers at a lod of greater than 8.0. Since 94 of the markers also fell into YAC contigs, we were able to verify that the RH placement was consistent with the YAC linkage in all 94 cases. That is, all of the RH-mapped markers were placed within 15cR (about 4.2 Mb) of another RH-mapped marker in the contig in question.

We also note that there are no gaps larger than 30 cR in the framework map of any chromosome arm. Since the Genebridge 4 Hybrid Panel allows us to detect RH linkage between markers 30 cR apart, we can conclude that there are no substantial

[3] Chromosome Y was not mapped because a detailed physical map of Chromosome Y had already been published by a collaborating Whitehead team [47].

Chr	Framework Markers	Total Markers	Length (cR)	Physical Length (Mb)	RH vs. Physical Length (cR/Mb)
1	139	1066	911	248	3.7
2	139	961	999	240	4.2
3	117	863	819	202	4.1
4	109	608	674	191	3.5
5	77	626	485	183	2.7
6	117	678	763	173	4.4
7	76	643	606	161	3.8
8	87	495	652	146	4.5
9	78	488	428	137	3.1
10	91	528	612	136	4.5
11	76	636	543	136	4.0
12	75	552	556	135	4.1
13	51	287	289	92	3.1
14	43	407	338	88	3.8
15	50	381	376	84	4.5
16	38	343	245	92	2.7
17	42	316	346	87	4.0
18	55	277	441	80	5.5
19	30	273	308	63	4.9
20	41	278	273	68	4.0
21	25	98	213	37	5.8
22	22	182	184	41	4.5
X	60	371	697	155	4.5
Total	1638	11357	11362	2975	3.8

Table 4.2: Breakdown showing the number of markers on the radiation hybrid framework and placement maps, and the estimated physical size of a centiRay, by chromosome. Physical lengths are calculated as proportional fractions of a 3,000 Mb genome (minus 25 Mb for the Y chromosome). The RH map lengths exclude the size of large intervals at the centromere.

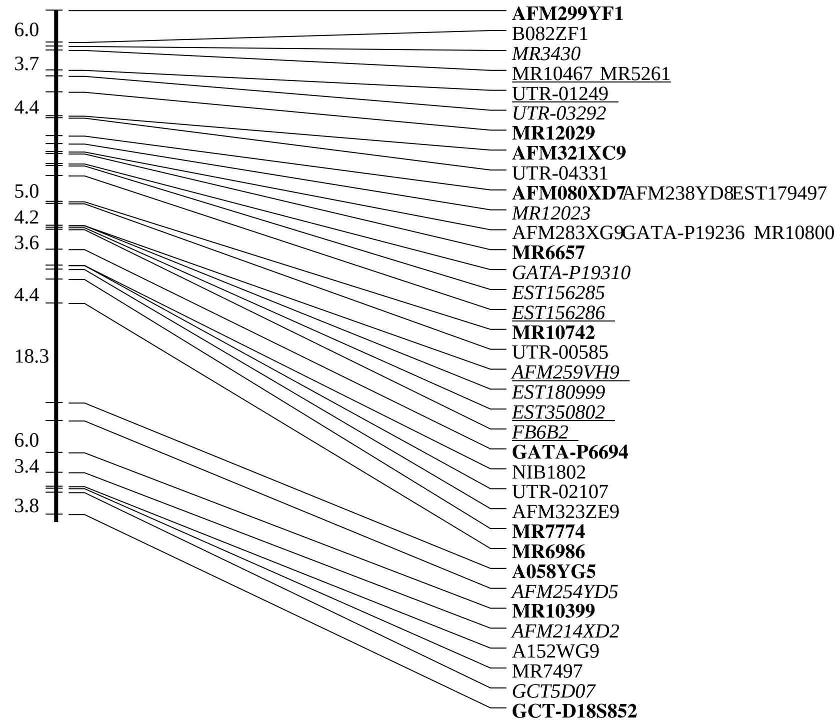


Figure 4.12: A radiation hybrid map of the p-arm of chromosome 18. All distances are measured in centiRays. Marker names in bold are part of the lod-2.5 framework map. Placed markers are shown in their most likely position; the lod score of the placement represents the difference in log likelihoods between the best and next-best placement of the marker. The strength of a placement is reflected by the type style: markers placed at a lod between 1.0 and 2.0 are in italics, those placed at a lod between 2.0 and 3.0 are underlined, and markers placed at a lod of greater than 3.0 are both underlined and italicized.

holes in any of the framework maps. (However, it is possible that the frameworks might have incomplete coverage of the centromeres or telomeres. The issue of gaps at the centromeres is of independent interest and is discussed below.)

Measuring the accuracy of the maps is difficult, since there is no known correct map to which we can appeal. However, we have validated our maps and our methods in several ways. First we verified that our map construction methods can be used successfully on simulated data, as reported Section 4.6.5. One way of estimating accuracy is to compare our maps to different maps of the same regions. Constructing the integrated map described below facilitated such comparisons. Across the entire genome we found only four conflicts between the initial RH framework and the Genethon genetic linkage map; each conflict consists of two markers that are at most 3 Mb apart.

Furthermore, chromosomal assignment by RH linkage appears to be extremely accurate. Of the 5,134 markers also assigned by other methods, we found only 31 discrepancies between the assignment due to RH linkage and that derived from another method. In most of these cases, experimental verification confirmed the RH linkage assignment. In the remaining few cases, we suspect that DNA repeats are to blame.

Centromeric Retention

We have noticed two phenomena that occur at the centromeres of most chromosomes. First, markers near the centromeres tend to have a higher retention rate than markers from elsewhere along the chromosome. This finding has previously been noted by other researchers [63, 111]. Second, we see very low pairwise linkage between markers on opposite sides of the centromere whose genetic map positions are only a few cM apart (so that we would expect to see significant RH linkage).

We believe that the high retention rate at the centromeres is related to the fact that fragments can be retained by the hybrid cell in two different ways. Walter, *et al.* [111] have observed by fluorescence *in situ* hybridization (FISH) that some human DNA fragments become integrated into existing hamster chromosomes in the hybrid cells, while other fragments join together to form entirely new chromosomes. Our hypothesis is that centromeric fragments are more likely than other fragments to be retained in this latter form, since the centromere confers stability on the nascent chromosome. However, it is possible for a centromeric fragment to be incorporated into a hybrid chromosome as well, provided that the centromeric sequence is inactivated so that it no longer functions as a centromere. (A chromosome with two centromeres cannot reproduce properly during mitosis and thus would be unstable.) One could use FISH to flag markers near centromeres and determine whether they are retained exclusively as independent chromosomes, or whether they also appear in hybrid human/hamster chromosomes.

The phenomenon of low linkage across centromeres may be explained in several ways. One is that the probability of a radiation-induced break might be extremely high at the centromere. If this were the case, nearby markers on opposite sides of the centromere would be retained independently in almost all hybrids, yielding the low pairwise linkage that we see.

Another possibility is that the gaps at the centromeres of our radiation hybrid maps might be real. It is known [78] that there is a low rate of recombination near the centromeres, so the genetic map distances in the region may correspond to much larger physical distances than expected. Thus two markers near the centromere that are 3 cM apart on the genetic map might be be separated by 10 Mb on the physical map, so the pairwise RH linkage between them would be rather low. It is also known that the centromeric region contains relatively few genes. The region may be similarly biased against the selection of random markers. Thus our difficulty in seeing linkage across the centromere might be exacerbated by the fact that we rarely find usable markers within a few megabases of the centromere.

One way to test this hypothesis is to try specifically to generate several markers in the centromere itself, as James' group did with microsatellite markers on Chromosome 11 [63]. One could then test to see whether satellite markers from opposite sides of the centromere show linkage to one another or not. If the centromere is just a region with low density of the sort of markers used in our map, it ought to be possible to build a map of tightly-linked microsatellite markers spanning the centromere. However, if there is some particular point in the centromere with an unusually high break probability, that break point would appear even in a map of intra-centromeric markers.

A final possibility has to do with high centromeric retention rates. James' study of chromosome 11 [63] showed that the retention frequencies of several markers near the centromere were higher than 60%. It is possible that for markers right near the centromere, the retention rates are so high that the hybrid screening vectors are rejected by our database software. (Very few markers with retention rates greater than 65% are accepted, since such retention patterns can be indicative of DNA repeats or of indiscriminate probes that hybridize to nearly everything.) To test this theory, one would find markers known (by other mapping methods) to be near the centromere of a specific chromosome, screen them against the hybrid panel, and examine their retention frequencies. If the retention frequencies are extremely high, one could then try to build a map spanning the centromeric region by adjusting the assumed retention frequency of the model.

It is likely that some combination of high retention rates and low marker density near the centromeres is to blame, and that we will be able to map the centromeric regions by compensating for these problems as described above. We hope to perform experiments to resolve this issue shortly.

Efficiency

A key issue in large-scale map construction is that of computational efficiency. RHMAPPER is designed for use on large data sets. The central function that evaluates the likelihood of an order is highly optimized and scales linearly with the number of markers. The computation time required to evaluate a single group becomes noticeable only for fairly large groups (about 1 second for 70-80 markers on our DEC Alpha 3000 workstation). However, for procedures like the ripple test that repeatedly evaluate orders, such delays do add up. To build the initial 6,193-marker map, we attempted to place 6,608 chromosomally-assigned markers onto a 1,339-marker framework spanning 23 chromosomes. This task required about 8 hours of computation time. We ran such processes as batch jobs overnight and kept a log of their actions.

Error Detection and Correction

Finally, we have successfully implemented error detection and correction. We selected about 2,500 of the most egregious errors from a preliminary version of our genome-wide map[4]. Upon laboratory verification of these assays, we found that 65.6% of RHMAPPER's error predictions were confirmed. Thus, it appears that despite some known flaws in the model, the software can correctly identify a large proportion of laboratory errors. Such iterative error-detection and verification is a crucial step towards refining the data and ultimately perfecting the maps.

Integrated Map

The resolution of radiation hybrid mapping can be controlled by adjusting the amount of radiation used to create the hybrid panel. In particular, the method can generate maps intermediate in resolution between genetic linkage maps and fine-grain STS content maps on YACs [35, 60]. Thus, radiation hybrid maps may provide the cohesion necessary for integrating several different types of mapping data.

The radiation hybrid maps described here formed the basis for the construction of a 15,086-marker integrated map described in Hudson, *et al.* [60], which has since been expanded to contain 20,186 markers. The integrated map shows the consistency of the RH map with other types of map data. For example, there are only four minor conflicts between the RH framework and the Genethon genetic linkage map; the placement map is overwhelmingly consistent with the genetic linkage map as well. The RH map

[4] The 2,500 assays to verify were selected from about 500,000 assays used in that stage of map construction, representing about 0.5% of the total.

provides a scaffold for incorporating the fine-ordered STS-content map data on YACs while highlighting many of the YAC errors. For example, chimeric clones are quite visible when the YACs are anchored to a substantially-correct large-scale map. Thus, the RH map has enabled the construction of a large-scale integrated map with estimated YAC coverage of 94% of the genome. Based on the low frequency of significant conflicts between maps, we estimate that perhaps 0.5% of the loci on the integrated map may be significantly misplaced.

Part of the integrated map of chromosome 18q is shown in Figure 4.13. The long vertical lines represent the different types of maps; the STS-content map is represented twice (on the outside columns) and is compared with the Genethon genetic linkage map and with our radiation hybrid map. Note that while there are a few conflicts between the genetic and radiation hybrid maps (seen as crossed lines in the middle of the five columns connecting the maps), there are no crosses involving two framework markers (whose names appear in boldface). The three maps appear overwhelmingly consistent.

4.8 Conclusions

In this chapter we have chronicled our experiences in building the first human genome-wide radiation hybrid maps. In addition to providing a clear contribution to the field of genetics, this project has been a successful experiment in drawing conclusions from imperfect data. The hidden Markov model, a theoretical model representing our uncertainty about the data, played an essential role in the mapping process. However, the model alone was insufficient for finding correct maps quickly. Ultimately, employing a combination of theoretical modeling and practical engineering techniques enabled us to achieve our goals. This union of the theoretical and the practical is likely to be an effective paradigm for solving future problems of learning from imperfect data.

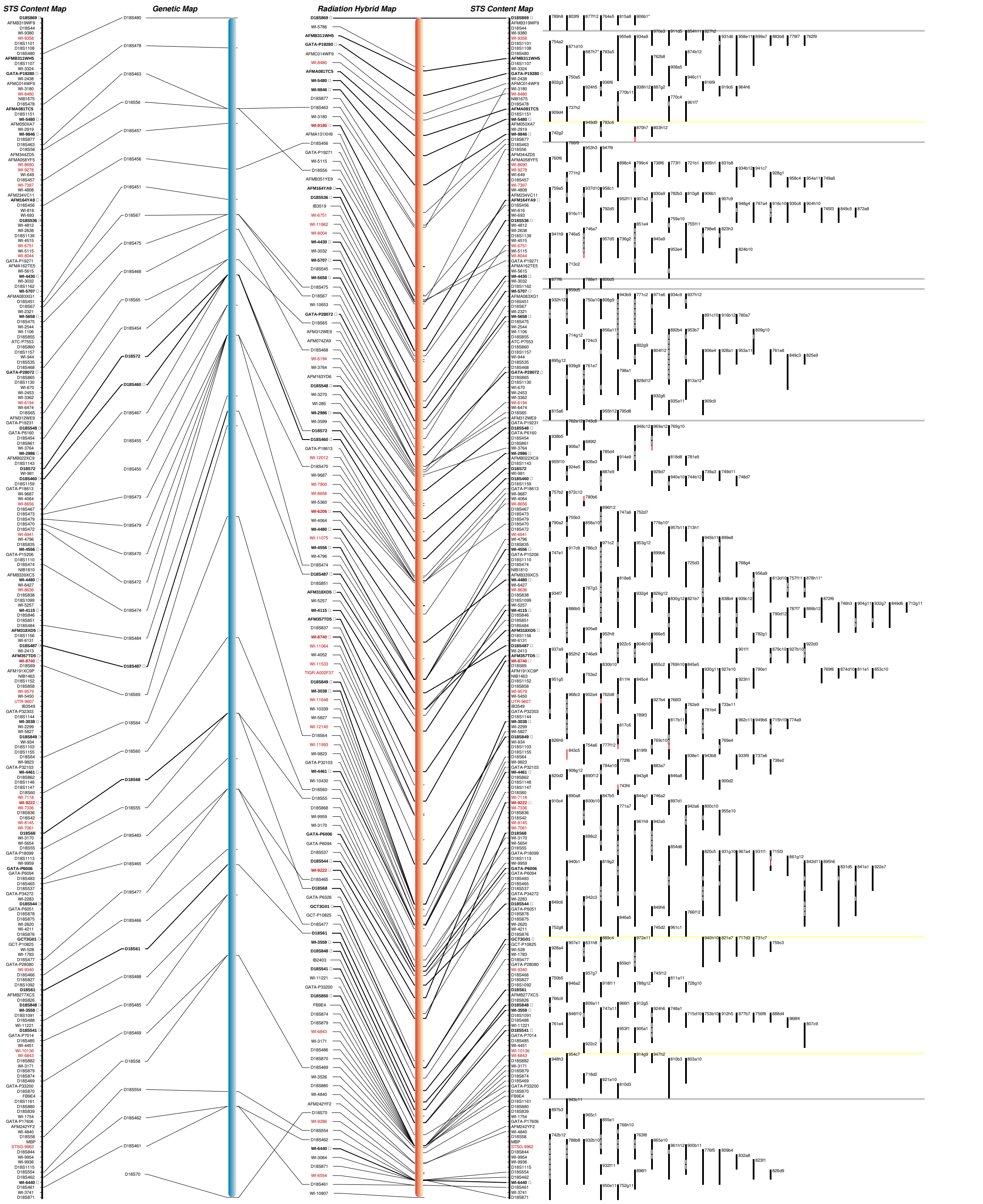


Figure 4.13: An integrated map of chromosome 18q. The STS-content map order of markers is represented twice (on the outside columns), to show the three-way comparison between the STS-content map, the genetic linkage map, and the radiation hybrid map. Lines connect the same markers on different maps. The vertical lines in the right hand columns show the positions of the YAC clones used for STS-content mapping.

Glossary

base One of the individual components (A,C,T,G) of a DNA strand. Also used as a measure of physical distance; larger distances are measured in kilobases (kb) or megabases (Mb). See page 104.

centromere A unique region of each chromosome (often found in the center) crucial to accurate DNA replication and separation in cell division. See page 146.

chimeric clone A clone that picks up DNA from two or more regions of the genome, instead of only one. See page 113.

chromosome A linear arrangement of DNA, found inside a cell, that carries genetic information. See page 104.

co-retention The correlation between the retention patterns of two markers. See page 108.

diploid An organism having two copies of each chromosome in each cell. See page 110.

DNA *Deoxyribonucleic acid.* The basic genetic material. Long DNA molecules, formed of two complementary strands twisted together in a double helix, encode genes and other regulatory information. See page 104.

framework map A sparse, accurate map in which all markers are placed with high confidence. See page 107.

gene A substring of a DNA strand that encodes hereditary information. See page 104.

genetic algorithm A combinatorial optimization technique meant to simulate natural selection in evolution. A "population" of hypotheses is chosen, recombined in some way to produce "offspring," and then the population in each generation is whittled down to contain only the hypotheses that are most "fit" according to some selection criterion (the optimization function). In our case, each hypothesis is an order of the markers, and the optimization function is the likelihood of the best map corresponding to that marker order. See page 125.

genome The entire genetic complement of an organism (e.g., human). See page 104.

haploid An organism having one copy of each chromosome in each cell. See page 110.

hybrid One hybrid cell line in a radiation hybrid panel. See page 108.

likelihood The probability of seeing the observed data, given a map consisting of an ordered list of markers and the distances between them. See page 116.

lod score The log of the likelihood ratio between two maps. For example, the lod score comparing map M_1 to map M_2 is $\log_{10}[Pr(D|M_1)/Pr(D|M_2)]$. See page 116.

marker A landmark in a genome map. Markers may be genes, expressed regions of genes, or random sequenced strings of DNA. See page 105.

nucleotide One of the individual letters (A,C,T,G) of a DNA strand; a molecule comprised of one of four possible nitrogen bases attached to a sugar-phosphate group. See page 104.

pairwise lod score The lod score comparing two maps, each consisting of two markers: one with the two separated by the optimal distance θ_{opt}, and the other placing the two markers infinitely far apart ($\theta = 1$). See page 116.

physical map A map showing the relative locations of and distances between markers along a chromosome. See page 105.

placement map A map with markers placed into bins or intervals in a framework map. See page 107.

radiation hybrid panel A collection of several radiation hybrid cell lines. Each hybrid contains a different random subset of human DNA fragments. See page 108.

retention frequency The probability that a DNA fragment is retained by a hybrid. See page 119.

simulated annealing A combinatorial optimization technique based on random fluctuations. A "temperature" variable controls the probability of moving to a less-optimal state with respect to the optimization function. At high temperatures the system moves around randomly and may frequently move to less-optimal states; at low temperatures, few moves are made unless they improve the value of the optimization function. If the system starts at a high enough temperature and is "cooled" slowly enough, it is guaranteed to converge to the global optimum. For many applications it is impractical to cool slowly enough, so the method often converges to local optima. See page 125.

STS *Sequence-tagged site.* A DNA marker used as a landmark in map construction. See page 114.

STS-content mapping A marker-based mapping technique in which STSs (markers) are screened against YAC clones. See page 114.

yeast-artificial chromosome (YAC) A type of clone suitable for large-scale mapping. See page 113.

Bibliography

[1] K.J. Abel, *et al.* A radiation hybrid map of the BRCA1 region of chromosome 17q12–q21. *Genomics*, 17:632–641, 1993.

[2] Farid Alizadeh, Richard M. Karp, Lee A. Newberg, and Deborah K. Weisser. Physical mapping of chromosomes: A combinatorial problem in molecular biology. In *Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms*, pages 371–381, 1993.

[3] Farid Alizadeh, Richard M. Karp, Deborah K. Weisser, and Geoffrey Zweig. Physical mapping of chromosomes using unique probes. *Journal of Computational Biology*, 2(2):159–184, 1995.

[4] Dana Angluin. A note on the number of queries needed to identify regular languages. *Information and Control*, 51:76–87, 1981.

[5] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, November 1987.

[6] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[7] Dana Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990.

[8] Dana Angluin. Exact learning of μ-DNF formulas with malicious membership queries. Technical Report YALEU/DCS/TR-1020, Yale University, March 1994.

[9] Dana Angluin and Michael Kharitonov. When won't membership queries help? In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, pages 444–454, New Orleans, Louisiana, May 1991.

[10] Dana Angluin and Mārtiņš Kriķis. Learning with malicious membership queries and exceptions. In *Proceedings of the 7th Annual ACM Workshop on Computational Learning Theory (COLT '94)*, pages 57–66. ACM Press, New York, NY, 1994.

[11] Dana Angluin and Philip Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[12] Dana Angluin and Donna K. Slonim. Learning monotone DNF with an incomplete membership oracle. In *Proceedings of COLT '91*, pages 139–146. Morgan Kaufmann, 1991.

[13] Dana Angluin and Donna K. Slonim. Randomly fallible teachers: Learning monotone DNF with an incomplete membership oracle. *Machine Learning*, 14(1):7–26, January 1994.

[14] Yonatan Aumann and Michael O. Rabin. Clock construction in fully asynchronous parallel systems and PRAM simulation. *Theoretical Computer Science*, 128:3–30, 1994.

[15] Baruch Awerbuch, Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal graph exploration by a mobile robot. In *Proceedings of COLT '95*, pages 321–328. ACM, 1995.

[16] Eric Baum. Polynomial time algorithms for learning neural nets. In *Proceedings of the 3rd Annual Workshop on Computational Learning Theory*, pages 258–272, San Mateo, CA, 1990. Morgan Kaufmann.

[17] Eric Baum. Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Transactions on Neural Networks*, 2:5–19, 1991.

[18] Paul Beame, Allan Borodin, Prabhakar Raghavan, Walter Ruzzo, and Martin Tompa. Time-space tradeoffs for undirected graph traversal. In *Proceedings of the 31st Annual Symposium on the Foundations of Computer Science*, pages 429–438. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[19] Michael A. Bender and Donna K. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In *Proceedings of the Thirty-Fifth Annual Symposium on Foundations of Computer Science*, pages 75–85, November 1994.

[20] Margrit Betke. Algorithms for exploring an unknown graph. Master's thesis, MIT Department of Electrical Engineering and Computer Science, February 1992. (Published as MIT Laboratory for Computer Science Technical Report MIT/LCS/TR-536, March, 1992).

[21] Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal learning of an unknown environment. In *Proceedings of the 1993 Conference on Computational Learning Theory*, pages 277–286, Santa Cruz, CA, July 1993. (Published as MIT AI-Memo 1474, CBCL-Memo 93; and in *Machine Learning*, Feb/Mar 1995.).

[22] Avrim Blum. Separating distribution-free and mistake-bound learning models over the Boolean domain. *SIAM Journal on Computing*, 23(5):990–1000, Oct. 1994.

[23] Avrim Blum, Prasad Chalasani, Sally Goldman, and Donna Slonim. Learning with unreliable boundary queries. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, pages 98–107, July 1995. Submitted by invitation to *Journal of Computer and System Sciences*.

[24] Avrim Blum and Ronald L. Rivest. Training a 3-node neural net is NP-Complete. In *Advances in Neural Information Processing Systems I*, pages 494–501. Morgan Kaufmann, 1989.

[25] M. Blum and W. J. Sakoda. On the capability of finite automata in 2 and 3 dimensional space. In *18th Annual Symposium on Foundations of Computer Science*, pages 147–161. IEEE, 1977.

[26] Manuel Blum and Dexter Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *19th Annual Symposium on Foundations of Computer Science*, pages 132–142. IEEE, 1978.

[27] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, April 1987.

[28] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[29] Michael Boehnke. Radiation hybrid mapping by minimization of the number of obligate chromosome breaks: Genetic analysis workshop 7. *Cytogenetic Cell Genetics*, 59:96–98, 1992.

[30] Michael Boehnke, Elizabeth Hauser, Kenneth Lange, Kathryn L. Lunetta, Justine Uro, and Jill Vanderstoep. RHMAP version 2.01: Statistical package for multipoint radiation hybrid mapping, 1994. Software and documentation available from `http://www.sph.umich.edu/group/statgen/software`.

[31] Michael Boehnke, Kenneth Lange, and David Cox. Statistical methods for multipoint radiation hybrid mapping. *American Journal of Human Genetics*, 49:1174–1188, 1991.

[32] Michael Boehnke, *et al.* RHMAP version 1.01: Statistical package for multipoint radiation hybrid mapping, 1991. Software and documentation received by personal communication.

[33] K.S. Booth and G.S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.

[34] Rodney Brooks, Pattie Maes, Maja Mataric, and Grinell More. Lunar base construction robots. In *Proceedings of the IEEE International Workshop on Robots and Systems*, pages 389–392. IEEE Press, 1990.

[35] D. T. Burke, G. F. Carle, and M. V. Olson. Cloning of large exogenous DNA into yeast by means of artificial chromosomes. *Science*, 236:806–812, 1987.

[36] N. Cesa-Bianchi, Y. Freund, D. Helmbold, D. Haussler, R. Schapire, and M. Warmuth. How to use expert advice. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 382–391, San Diego, CA, May 1993.

[37] I. Chumakov, *et al.* A YAC contig map of the human genome. *Nature*, 377:S175–S297, 1995.

[38] Francis S. Collins. Ahead of schedule and under budget: The genome project passes its fifth birthday. *Proceedings of the National Academy of Sciences*, 92:10821–10823, November 1995.

[39] Stephen A. Cook and Charles W. Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM Journal on Computing*, 9:636–652, 1980.

[40] Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[41] A. Coulson, *et al.* Toward a physical map of the genome of the nematode C. elegans. *Proceedings of the National Academy of Sciences, USA*, 83(20):7821–7826, 1986.

[42] D. Cox and R. Myers. Chromosome 4 and 12 radiation hybrid data realease, electronically published at `http://shgc.stanford.edu/rhmap.html`, 1996.

[43] D. R. Cox, M. Burmeister, E. R. Price, S. Kim, and R. M. Myers. Radiation hybrid mapping: A somatic cell genetic method for constructing high-resolution maps of mammalian chromosomes. *Science*, 250:245–250, 1990.

[44] T. Dean, D. Angluin, K. Basye, S. Engelson, L. Kaelbling, E. Kokkevis, and O. Maron. Inferring finite automata with stochastic output functions and an application to map learning. In *Proceedings of AAAI-92*, pages 208–214. AAAI, 1992.

[45] Xiaotie Deng and Christos H. Papadimitriou. Exploring an unknown graph. In *Proceedings of the 31st Symposium on Foundations of Computer Science*, volume I, pages 355–361, 1990.

[46] Jeff Edmonds. Time-space trade-offs for undirected *st*-connectivity on a JAG. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 718–727, May 1993.

[47] S. Foote, D. Vollrath, A. Hilton, and D. C. Page. The human Y chromosome: Overlapping DNA clones spanning the euchromatic region. *Science*, 258:60–66, 1992.

[48] Michael Frazier, Sally Goldman, Nina Mishra, and Leonard Pitt. Learning from a consistently ignorant teacher. In *Proceedings of the 7th Annual ACM Workshop on Computational Learning Theory*, pages 328–339. ACM Press, New York, NY, 1994.

[49] Michael Frazier and Leonard Pitt. CLASSIC learning. In *Proceedings of the 7th Annual ACM Workshop on Computational Learning Theory*, pages 23–34. ACM Press, New York, NY, 1994.

[50] Yoav Freund, Michael Kearns, Dana Ron, Ronitt Rubenfeld, Robert Schapire, and Linda Sellie. Efficient learning of typical finite automata from random walks. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 315–324, May 1993.

[51] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[52] Sally Goldman and Robert Sloan. Can PAC learning algorithms tolerate random noise. *Algorithmica*, 14(1):70–84, July 1995.

[53] Sally A. Goldman, Michael J. Kearns, and Robert E. Schapire. Exact identification of circuits using fixed points of amplification functions. *SIAM Journal on Computing*, 22(4):705–726, August 1993.

[54] Sally A. Goldman and H. David Mathias. Learning k-term DNF formulas with an incomplete membership oracle. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 77–84, Pittsburgh, PA, 1992. ACM Press.

[55] Mark S. Guyer and Francis S. Collins. How is the human genome project doing, and what have we learned so far? *Proceedings of the National Academy of Sciences*, 92:10841–10848, November 1995.

[56] J.B.S. Haldane. The combination of linkage values, and the calculation of distance between the loci of linked factors. *J. Genet.*, 8:299–309, 1919.

[57] David Haussler, Michael Kearns, Nick Littlestone, and Manfred K. Warmuth. Equivalence of models for polynomial learnability. *Information and Computation*, 95(2):129–161, December 1991.

[58] David Haussler, Nick Littlestone, and Manfred K. Warmuth. Predicting {0,1} functions on randomly drawn points. *Information and Computation*, 115(2):284–293, 1994.

[59] Ian Horswill. *Specialization of Perceptual Processes.* PhD dissertation, MIT Artificial Intelligence Laboratory, September 1994. Available as Technical Report MIT-AITR-1511.

[60] T. Hudson, *et al.* An STS-based map of the human genome. *Science*, 270:1945–1954, Dec. 1995.

[61] C.W. Richard III, *et al.* A radiation hybrid map of the proximal long arm of human chromosome 11, containing the multiple endocrine neoplasia type 1 (MEN-1) and bcl-1 disease loci. *Am. J. hum. Gen.*, 49:1189–1196, 1991.

[62] C.W. Richard III, *et al.* A radiation hybrid map of the distal short arm of human chromosome 11, containing the beckwith-weidemann and associated embryonal tumor disease loci. *Am. J. hum. Gen.*, 52:915–921, 1993.

[63] M.R. James, *et al.* A radiation hybrid map of 506 STS markers spanning human chromosome 11. *Nature Genetics*, 8:70–76, 1994.

[64] Michael Kearns and Ming Li. Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22:807–837, 1993.

[65] Michael J. Kearns and Robert E. Schapire. Efficient distribution-free learning of probablistic concepts. In *Proceedings of the Thirty-First Annual Symposium on Foundations of Computer Science*, volume I, pages 382–391. IEEE, 1990.

[66] Michael J. Kearns and H. Sebastian Seung. Learning from a population of hypotheses. In *Proceedings of COLT '93*, pages 101–110, 1993.

[67] John Kececioglu and Eugene Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, 13:7–51, 1995.

[68] Y. Kohara, K. Akiyama, and K. Isono. The physical map of the whole E. coli chromosome: application of a new strategy for rapid analysis and sorting of a large genomic library. *Cell*, 50(3):495–508, 1987.

[69] Zvi Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, second edition, 1978.

[70] Philip D. Laird. *Learning from Good and Bad Data.* Kluwer international series in engineering and computer science. Kluwer Academic Publishers, Boston, 1988.

[71] Eric S. Lander and Philip Green. Construction of multilocus genetic linkage maps in humans. *Proceedings of the National Academy of Sciences, USA*, 84:2363–2367, 1987.

[72] K.J. Lang and E.B. Baum. Query learning can work poorly when a human oracle is used. In *Proceedings of International Joint Conference on Neural Networks.* IEEE, 1992.

[73] Kenneth Lange, Michael Boehnke, David Cox, and Kathryn Lunetta. Statistical methods for polyploid radiation hybrid mapping. *Genome Research*, 5:136–150, 1995.

[74] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[75] Nick Littlestone. Redundant noisy attributes, attribute errors, and linear threshold learning using Winnow. In *Proceedings of the 4th Annual Workshop on Computational Learning Theory*, pages 147–156, San Mateo, CA, 1991. Morgan Kaufmann.

[76] Kathryn Lunetta, Michael Boehnke, Kenneth Lange, and David Cox. Experimental design and error detection for polyploid radiation hybrid mapping. *Genome Research*, 5:151–163, 1995.

[77] Wolfgang Maass. On-line learning with an oblivious environment and the power of randomization. In *Proceedings of the 4th Annual Workshop on Computational Learning Theory*, pages 167–175, San Mateo, CA, 1991. Morgan Kaufmann.

[78] Melanie Mahtani. Personal communication, 1996.

[79] Maja Mataric. A distributed model for mobile robot environment-learning and navigation. Master's thesis, MIT Artificial Intelligence Laboratory, May 1990. Available as Technical Report MIT-AITR-1228.

[80] Maja Mataric. *Interaction and Intelligent Behavior.* PhD dissertation, MIT Artificial Intelligence Laboratory, August 1994. Available as Technical Report MIT-AITR-1495.

[81] Milena Mihail. Conductance and convergence of Markov chains – a combinatorial treatment of expanders. In *Proceedings of the 30th Annual Symposium on the Foundations of Computer Science*, pages 526–531. IEEE Press, 1989.

[82] Edward F. Moore. *Gedanken-Experiments on Sequential Machines*, pages 129–153. Princeton University Press, 1956. Edited by C. E. Shannon and J. McCarthy.

[83] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, New York, 1995.

[84] M. Olson, *et al.* Random-clone strategy for genomic restriction mapping in yeast. *Proceedings of the National Academy of Sciences, USA*, 83(20):7826–7830, 1986.

[85] Lynne E. Parker. *Heterogeneous Multi-Robot Cooperation.* PhD dissertation, MIT, Artificial Intelligence Laboratory, February 1994. Available as Technical Report MIT-AITR-1465.

[86] Leonard Pitt and Manfred K. Warmuth. Prediction preserving reducibility. *Journal of Computer and System Sciences*, 41(3):430–467, December 1990. Special issue on the *Third Annual Conference of Structure in Complexity Theory* (Washington, DC., June 88).

[87] C. K. Poon. Space bounds for graph connectivity problems on node-named JAGs and node-ordered JAGs. In *Proceedings of the 33rd Annual Symposium on the Foundations of Computer Science*, pages 218–227. IEEE Press, 1993.

[88] W.H. Press, S. A. Teukolshy, W. T. Vettering, and B. P. Flannery. Minimization or maximization of functions. In *Numerical Recipes in C, 2nd Edition*, pages 394–455, Cambridge, UK, 1988. Cambridge University Press.

[89] Michael O. Rabin. Maze threading automata. Seminar talk presented at the University of California at Berkeley, October 1967.

[90] Lawrence Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, 1989.

[91] Prabhakar Raghavan. Lecture notes on randomized algorithms. Yale University Course *CS661*, Fall, 1989.

[92] Ronald L. Rivest and Robert E. Schapire. Diversity-based inference of finite automata. In *Proceeding of the Twenty-Eighth Annual Symposium on Foundations of Computer Science*, pages 78–87, Los Angeles, California, October 1987.

[93] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, April 1993.

[94] Dana Ron and Ronitt Rubinfeld. Exactly learning automata with small cover time. In *Proceedings 8th Annual Conference on Computational Learning Theory*, pages 427–436. ACM Press, New York, NY, 1995.

[95] Dana Ron and Ronitt Rubinfeld. Learning fallible deterministic finite automata. *Machine Learning*, 18(2/3):149–185, 1995.

[96] Daniela Rus, Bruce Donald, and Jim Jennings. Moving furniture with teams of autonomous robots. In *Proceedings of the IEEE International Workshop on Robots and Systems*. IEEE Press, 1995.

[97] Yasubumi Sakakibara. On learning from queries and counterexamples in the presence of noise. *Information Processing Letters*, 37(5):279–284, March 1991.

[98] Walter J. Savitch. Maze recognizing automata and nondeterministic tape complexity. *Journal of Computer and System Sciences*, 7:389–403, 1972.

[99] George Shackelford and Dennis Volper. Learning k-DNF with noise in the attributes. In *First Workshop on Computatinal Learning Theory*, pages 97–103, Cambridge, Mass. August 1988. Morgan Kaufmann.

[100] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing Markov chains. *Information and Computation*, 82(1):93–133, July 1989.

[101] Robert H. Sloan. Types of noise in data for concept learning. In *First Workshop on Computational Learning Theory*, pages 91–96. Morgan Kaufmann, August 1988.

[102] Robert H. Sloan. *Computational Learning Theory: New Models and Algorithms*. PhD thesis, MIT EECS Department, May 1989. (Published as MIT/LCS/TR-448.).

[103] Robert H. Sloan and György Turán. Learning with queries but incomplete information. In *Proceedings of the 7th Annual ACM Workshop on Computational Learning Theory*, pages 237–245. ACM Press, New York, NY, 1994.

[104] Donna Slonim, Lincoln Stein, Leonid Kruglyak, and Eric Lander. RHMAPPER: An interactive computer package for constructing radiation hybrid maps. Unpublished manuscript, 1996.

[105] Carl H. Smith. Three decades of team learning. In *Algorithmic Learning Theory*. Springer Verlag, 1994. To appear.

[106] Lincoln Stein, Leonid Kruglyak, Donna Slonim, and Eric Lander. RHMAPPER software package, 1995. Available at `http://www-genome.wi.mit.edu/ftp/pub/software/rhmapper/`, and via anonymous ftp from ftp-genome.wi.mit.edu, directory `/pub/software/rhmapper`.

[107] Richard Sutton, editor. Special issue on reinforcement learning. *Machine Learning*, 8(3/4), May 1992.

[108] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[109] Leslie G. Valiant. Learning disjunctions of conjunctions. In *Proceedings IJCAI-85*, pages 560–566. International Joint Committee for Artificial Intelligence, Morgan Kaufmann, August 1985.

[110] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[111] M. A. Walter, D. J. Spillett, P. Thomas, J. Weissenbach, and P. N. Goodfellow. A method for constructing radiation hybrid maps of whole genomes. *Nature Genetics*, 7:22–28, 1994.

[112] Holly Yanco and Lynn Stein. An adaptive communication protocol for cooperating mobile robots. In *From Animals to Animats 2: Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior*. MIT Press, 1993.

[113] Geoffrey Zweig. HAMTSP, unpublished software for solving the Hamming distance traveleing salesman problem, 1994. Received through personal communication.