Automating Computational Molecular Genetics:

Solving the Microsatellite Genotyping Problem

See-Kiong Ng

January 23, 1998
CMU-CS-98-105

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:
Mark W. Perlin, Chair
Scott E. Fahlman
James H. Morris
Robert E. Ferrell, Department of Human Genetics, University of Pittsburgh

This research was sponsored in part by the NIH National Institute of Neurological Disorders and Stroke (MWP) under Grant No. R01 NS32084.

Keywords: Artificial intelligence, automation software, biotechnology, computational biology, molecular genetics, microsatellite genotyping, pattern matching, FAST-MAP.

ABSTRACT

The Human Genome Project has extended the reach of modern genetics by providing an infrastructure of high-resolution genetic maps. Scientists can now find genes using these maps by genotyping — experimentally assaying the genome at mapped genetic markers. To track the inheritance patterns of a genetic disorder, individual genomes are genotyped at high resolution using densely distributed genetic markers, such as the microsatellites. However, because of the complexity associated with the inheritance patterns of most common human genetic diseases, hundreds of thousands of genotyping experiments are typically required to genetically localize even one disorder on the genome.

The full automation of microsatellite-based genotyping is currently limited by the human scoring bottleneck: every experiment must be viewed by a human eye. The intricate genotyping data, densely multiplexed for throughput, is confounded with intrinsic data artifacts such as PCR stuttering. Human experts are required to visually decipher the highly complex data patterns that resulted. It is estimated that over half the cost of microsatellite-based genotyping is due to this human scoring effort.

We have developed and implemented novel computer-based analysis methods that computationally solve the various problems associated with the microsatellite scoring bottleneck. Our system, FAST-MAP, is a platform-independent fully automated genotyping system that accurately calls alleles from quantitative microsatellite data. FAST-MAP has been extensively tested and used by scientists worldwide to generate genotypes with high accuracy from real data generated in high throughput genetic laboratories. With FAST-MAP, we have shown that by appropriately modeling and representing genotype data, powerful computational strategies can overcome key molecular biology bottlenecks and significantly advance the rapid localization of genes across the whole human genome.

ACKNOWLEDGMENTS

Many people helped greatly with this thesis. My advisor, Mark Perlin, adopted me as his student when my former advisor left CMU, and introduced me to the fascinating field of molecular genetics. Mark's enthusiasm must have been contagious as I immediately fell in love with the field, even though my knowledge of biology was limited. Since then, Mark has taught me biology, genetics, biotechnology, computer science and even lessons in life. I was also fortunate to have a very supportive thesis committee: Jim, Scott, and Bob shared my belief in the value of building a working system that solves a real problem; I am grateful to them for constantly encouraging me.

FAST-MAP was by no means a one-man effort. For example, Nandita Mukhopadhyay created the sophisticated graphical interface that made FAST-MAP more user-friendly. Lillian Bloch, our in-house "alpha-tester", tirelessly tested the initial versions of FAST-MAP. Most amazing, however, was our FAST-MAP users: they not only generously contributed their data for system testing and development, but also tolerated the software bugs that I had overlooked, and even fixed some for me. I am particularly indebted to Gordon Bentley (gene/Networks), Dr. Vicki Magnuson (NIH/FUSION), Dr. Mike Gorin (University of Pittsburgh), Frosti Palsson (deCODE Genetics, Iceland), and Dr. Richard Mott (Smithkline Beecham, UK) for their invaluable contributions to the project.

I am grateful to the School of Computer Science both for its financial support and for maintaining a most stimulating research culture. I thank Prof. M. Tomita (Tommy), whom I have known since I was an undergraduate at CMU, and who became my first graduate student advisor. He inspired me by showing me the wonders of Computer Science.

Getting a Ph.D. in computer science has been a long and arduous process — but it was never unenjoyable. Among the rewards are the many great friendships that I was blessed with along the way. I would like to thank in particular Jürgen Dingel and Denis Dancanet for providing the athletic aspect of my student life, Jeff Holland for opening his house to me whenever I felt the urge to get out of Pittsburgh, C.B. Chan for going to countless operas at the Met with me, and Dr. D. A. Wilkinson, whom I met at a time when I was considering quitting. He showed me that I could get what I want if I would give myself a chance.

Finally, I would like to dedicate this thesis to my loving parents and family, who believe in me much more than I do myself.

Contents

1. Introduction

Current progress of the Human Genome Project (Hoffman, 1994; Jordan, 1992; Watson, 1990) has resulted in the construction of highly useful genetic and physical maps (Gyapay *et al.*, 1994; Matise *et al.*, 1994; NIH/CEPH Collaborative Mapping Group, 1992) which, for the first time, enabled the routine isolation of the human genes. Using the genetic maps, geneticists can systematically isolate the causative genes for genetic diseases (Davies *et al.*, 1994; Lander and Schork, 1994) and (eventually) develop effective strategies for diagnosis, treatment, and prevention of the diseases by understanding the disease biochemistry from the causative genes.

Key to the construction and use of the genetic maps is the ability to rapidly and accurately sample points of interest on a chromosome, i.e., "genotype". The geneticist uses these genomic sample points, or "genetic markers", to assess the inheritance patterns of chromosomal segments between related individuals to discover possible disease gene locations on the genome. For complex genetic diseases (e.g. diabetes), the associated complicated inheritance patterns must be disambiguated by sampling the genome at very high resolution. Currently, to map a complex disease, 300 to 500 genetic markers such as microsatellite markers (Weber and May, 1989) are sampled along the chromosomes for every individual in the study (Hyder *et al.*, 1991; Todd, 1994). This number of markers is expected to increase ten-fold to 3,000 markers or more as the resolution of genetic maps improves. The number of individuals in a disease study, currently ranging between 500 and 5,000, will also increase as more complex (but common) genetic diseases are being studied. Thus, to study just one disease, millions of genotyping experiments must be performed and assayed rapidly. It is therefore critical to fully automate the genotyping process, as there are thousands of common complex genetic diseases in humans that must be studied.

1.1. Problem

There are two major steps in the genotyping process:

1. <u>Generation</u>: The first step is data generation, which involves setting up and running the genotyping experiments in the laboratories. For each experiment, DNA samples from

individuals in the study are isolated, purified, and then PCR-amplified (Mullis *et al.*, 1986)with dye-labeled primers. By choosing the appropriate primers for PCR, a specific region of the DNA, which usually corresponds to a genetic marker, can be selectively amplified to create millions of copies of the specified (marker) DNA region for experimental analysis. The resulting PCR mixture is then loaded onto polyacrylamide gels, on which a process called *electrophoresis* separates out the DNA fragments by size. For signal detection, the primers were tagged with radioactive elements or fluorescent dye molecules so that data can be captured by exposure to a film or by laser sensors. To generate huge amounts of data rapidly, hundreds to thousands of genotyping experiments are typically multiplexed onto a single gel for maximal throughput.

2. <u>Analysis</u>: The second step is data analysis. This involves (i) searching the scanned gel images for regions containing data for each genotyping experiments multiplexed on the gel, (ii) calibrating the genotyping data using known size standards to label the data in molecular units, and (iii) parsing the complex data patterns for each experiment to extract the underlying genotypes. Because of the high degree of data multiplexing and the inherent data artifacts, analysis of genotyping data is a labor-intensive process that has precluded automation. This data analysis step, which we will refer to as the "genotyping problem", is the problem that we will solve in this dissertation. We will use the phrase "genotyping <u>process</u>" to refer to both running the PCR and gel electrophoresis experiments ("generation") and analyzing the data generated ("analysis").

To find disease genes, a third step follows the genotyping process:

3. <u>Discovery</u>: Once thousands to millions of genotypes of individuals in a population become available, we can trace the *chromosomal* inheritance patterns among individuals. Together with the inheritance patterns of a *physical trait* (e.g. the affectation of a genetic disease), we can statistically localize regions on the chromosomes that contain the candidate genes for the trait. The more genotyping data used in tracing inheritance, the higher the resolution of the candidate chromosomal segments that narrow down the disease gene search.

Tremendous efforts have already been expended in automating the data generation step (step 1):

- *machinery:* robot arms are used to handle DNA samples with high precision; PCR machines are employed to purify and amplify the DNA samples with great efficiency; sophisticated DNA sequencing machines are used to run the genotyping gels to directly generate digitized data for computer analysis; and
- *molecular biology:* experimental methods and reagents specially enhanced for high throughput automated genotyping are used. For example, AmpliTaq Gold, an enzyme for PCR amplification, is inactive at room temperature so that amplification reaction mixes may be assembled and pipetted in advance without fear of contamination for high throughput set-up. Experimental methods such as the reverse primer modification method (Magnuson *et al.*, 1996) that uses tailed reverse primers to eliminate the "plus-A" artifacts from the data, greatly improves automatic allele calling by generating allele patterns that can be easily called.

The gene discovery problem (step 3) has also been well-studied: there are many well-established computer programs such as LINKAGE (Ott, 1991; Terwilliger, 1994), MENDEL (Lange *et al.*, 1988), SIBPAL (S.A.G.E., 1997) , and GENEHUNTER (Kruglyak *et al.*, 1996) that are already widely used for automating the gene discovery (at least for simple or Mendelian diseases). The key bottleneck that remains is the step between data generation and gene discovery: the analysis of the genotyping data (step 2), which has yet to be fully automated. Nearly all genetic laboratories require at least an experienced human technician to visually inspect and analyze the gel data. This requisite manual labor attributes to roughly half of the cost (about $2-3 total per genotype), while increasing the error, time, and tedium.

Attempts to solve the genotyping problem have been unsuccessful because of the high degree of data multiplexing and the various confounding data artifacts. To automate the analysis of genotyping data, we have to overcome the following key difficulties:

- <u>Data tracking</u>. Because of the large quantities of genotypes necessary for genetic studies, high throughput data generation has been the main focus in genotyping. Thermocyclers (e.g. PE 9600 and MJ tetrad) which perform high throughput PCR and integrated catalyst machines (ABI/877) which handle both the liquid handling and thermal cycling process have greatly increased the number of genotyping experiments that can be prepared in a laboratory daily. Advances in genotyping technology enable hundreds to thousands of these genotyping experiments to be multiplexed onto a single gel. The PCR products are loaded onto different lanes on a single gel, conducted in multiple non-overlapping windows on a single lane, and even run in parallel in the same lane and window by tagging the DNA for different experiments with distinguishing fluorescent labels. It is a nontrivial task to automatically localize and extract the data for each of the highly multiplexed experiments from a gel for analysis. Even state-of-the-art genotyping systems currently rely on human technicians to manually track the lanes on the gels and identify the molecular weight standards used for calibrating the lanes into molecular size units ("base pairs", or bps). This is only a temporary measure, as the total number of experiments that can be packed onto a single gel will continue to increase (e.g. from 48 to 96) such that even manual tracking may soon become infeasible.

- Sizing precision. Microsatellites, the commonly used genetic markers for genotyping, have alleles (i.e. molecular values) that vary in sizes of 2-4 bp (i.e. approximately 1% difference in size). Molecular size standards technologies (e.g. Genescan-500, Bioventures) for sizing the DNA can currently only provide up to 20-50 bp resolution. The exact allele sizes of DNA must be interpolated from molecular weight calibration curves of inadequate resolution, incurring interpolation errors.

- Binning consistency. In microsatellite genotyping, the allele sizes are discrete values (i.e. each allele must be a whole number in bp), but the measured allele sizes of the DNAs are real numbers estimated from low-resolution size standards calibration curve. These real-numbered values must be mapped to unique integral labels or "allele bins". The conventional approach of rounding the estimated allele sizes to the nearest integers can be ambiguous for allele sizes with large rounding errors, and inconsistent allele binning is costly as it can reduce the power to detect linkage or give rise to inflated map lengths (Buetow, 1991).

- PCR stuttering. A major limitation of using microsatellite markers, especially the dinucleotide repeat markers, is the inherent stutter (or shadow) bands associated with the PCR amplification of the DNA samples. With a genotype containing two closely-spaced alleles, the stutter bands at one allele overlap with those at the other allele, resulting in a convoluted stutter pattern that is difficult to decipher even to the human eye. As such, many geneticists have resorted to using the less informative trinucleotide and tetranucleotide repeat markers instead of the dinucleotide repeat markers, primarily because of the reduced PCR stuttering typically observed with the former.

- Pattern specificity. Pattern characteristics such as the shape of the stutter trails, and the relative amplification of two alleles in a genotype, may vary from one allele class to another. In huge studies where there are hundreds of markers (thousands of allele classes) to be genotyped, allele calling can be a highly cognitive and visual task, as there are many specific patterns that have to be recognized and distinguished.

- Machine portability. Each commercial DNA sequencing machine typically comes with its own highly specific and labor intensive genotyping software (e.g. the GeneScan Analysis/GENOTYPER software for the ABI machines, the Fragment Manager software for the Pharmacia ALF machines) that takes time and effort for an incoming laboratory technician to learn. A generalized genotyping system that is portable between different DNA sequencers and experimental setups (and preferably, also runs on different computer platforms) would greatly lower the learning cost when different sequencers or experimental setups are used in a laboratory.

1.2. Solution

Most of the difficulties mentioned above are limitations that cannot be eliminated experimentally by adjusting experiment conditions or trying out different chemical reagents in the laboratories, which is a reason why the microsatellite genotyping problem remains unsolved in molecular genetics. We hope, with the work in this thesis, to bring to the genetic laboratories an additional nontraditional problem solving tool: *computational methods*. We will show how computational techniques from computer science and artificial intelligence can be employed to overcome key molecular biology bottlenecks. To demonstrate this, we have built a *fully automated* genotyping system for microsatellite markers that is accurate, robust, and efficient for real world applications. The system has been tested and used by real molecular geneticists in real laboratories to process real data from large scale, high throughput genotyping experiments.

The central claims of this dissertation are:

- By systematically applying computational techniques from computer science and artificial intelligence, we can overcome key molecular biology bottlenecks, such as the genotyping problem.

- By understanding the science and technology of genotyping, working closely with molecular biologists, and testing on large amount of real (not simulated) data, we can build a computer system to fully automate microsatellite genotyping.

- By modeling the genotyping data with mathematical models based on their biological elements, and constantly refining the models by testing with real data, we can formulate exquisite computational solutions for the genotyping problem, and even enable new functionality (e.g. pooled genotyping).

- By using domain knowledge *and* data to construct refined expectations to intelligently guide the computations, we can reduce computation-intensive tasks (e.g. 2-D lane tracking) into tractable problems that are solvable by simple algorithms.

- By exploiting data redundancy and abundancy, which is typical in experimental genetic data, we can attain extra consistency and robustness to overcome spurious noise and errors.

- By employing the computer's organizational and computational capability to simultaneously apply multiple sources of data and knowledge, we can improve the quality of the results and provide invaluable organizational utilities to the human user, especially when data sets become massive.

Finally, we also hope that the work in this dissertation will bring us closer to the following grand objectives:

- <u>Computer Science</u>. Although computers are currently present in modern genetic laboratories, they often play a passive role as they are typically used as mere data storage or presentation devices. Although computers are widely used in the genetic laboratories, very little of *computer science* is employed. As the Human Genome Project rapidly reaches its completion, there will be an impending explosion of genetic information. Computer science has much to offer in handling the information explosion from this coming era of genetic revolution. As one of the first steps, in this dissertation we use the computer to solve a major molecular genetics bottleneck. By doing so, we hope to demonstrate that the great arsenal of computational techniques in computer science for problem solving and information processing can be invaluable in solving the many data-intensive problems in molecular genetics.

- <u>Genetics</u>. With the work in this dissertation, full automation of genotyping becomes possible. A tremendous amount of useful genetic data that has been too expensive to acquire will become available to the geneticist, leading to new and challenging avenues for molecular biology research. In addition, the associated reduction in cost may eventually make genome scans a routine test, accessible one day to the general pubic. Our ultimate goal is to increase health and prolong life of the public by genotyping families, determining their risk profiles, and then moderating the environmental (or even genetic) component of genetic disease to reduce their greatest risks. By enabling the full automation of microsatellite genotyping, it is our hope that the work in this thesis will bring us one step closer to such a future.

2. Domain

A common feature of computer science, especially artificial intelligence, is that it is multi-disciplinary. Typically, half the battle is waged by learning about the other field. For example, speech recognition involves the application of advanced signal processing techniques, natural language understanding requires the awareness of linguistic theories, and expert system construction depends on the gathering of expertise from the problem domains. To successfully apply computer science to solve a problem from a different domain, the versatile computer scientist must first overcome the formidable task of learning an entirely different field quickly, sufficiently, and selectively.

The genotyping problem is particularly challenging in this aspect as its domain actually comprises three highly technical and inter-related fields:

- *Biology:* particularly, the study of the structure and organization of cells and DNAs;
- *Genetics:* the study of the mechanics of heredity, and
- *Biotechnology:* the state-of-the-art engineering tools for investigating problems in biology and genetics.

In view of the large number of technical terms involved in the three fields, a glossary of common technical terminology is provided at the end of this dissertation.

2.1. Biology

Life begins with a single cell (the *zygote*) which must contain sufficient programmatic instructions for it to develop into a complex multi-cellular organism in due course. In this section, we study how such genetic information is organized and structured in a compact form, and how this complex information is permuted for diversity and passed on from one generation to another.

2.1.1. Cells, DNAs, and chromosomes

All living organisms are strikingly similar at the cellular and molecular levels. We are all built from basic units called *cells*. Each cell is a complex automaton capable of generating new cellular molecules which are self-sustaining and self-replicating. The "brain" in each cell (with the exception of bacteria) is a nucleus which contain DNA (*deoxyribobucleic acid*) that encodes the requisite genetic information for life.

The cell's nuclear DNAs are called the *chromosomes*. There are typically not one, but several chromosomes in each cell, forming, in effect, a distributed DNA database of genetic information. In a given species, the number of chromosomes is the same for all members, for any aberration (e.g. missing one or having extra chromosomes) can be lethal. In humans, there are twenty-three pairs of homologous (matching) chromosomes. Each chromosome pair contains one chromosome from each parent. Collectively, the chromosomes are known as the *genome*. They contain all the genetic instructions necessary for building a complex living organism from a single cell.

The biochemistry of the DNA has revealed an amazing fact: the complex instructions of life are actually encoded in a deceptively simple language from an alphabet containing only four letters: {A, C, G, T}. Each chromosome (DNA) is a macromolecule consisting two intertwining polynucleotide chains commonly known as the "double helix" (Watson and Crick, 1953). The nucleotides are distinguished by their bases, which can be from one of two classes: purine (adenine, guanine) or pyrimidine (cytosine, thymine). On the double helix, the adenine (A) on one strand is always paired to a thymine (T) on the other strand, and the guanine (G) paired to a cytosine (C). Knowing the nucleotide sequence of one DNA strand implies the sequence of the other. As such, we often refer to DNA information as a *single* string containing letters from a four-letter alphabet: A, C, G, and T.

2.1.2. Genes, markers, and microsatellites

The genetic information in the human genome is organized hierarchically, as shown in Figure 2.1. The massive 3 billion DNA letters of genetic information are partitioned into 23 chromosomes. Each chromosome is a linear sequence of thousands of DNA sentences called *genes,* with large amounts of non-gene DNA sequences interspersed between the genes. These non-gene sequences do not code for any known biological function. They may be considered as "nonsense" biologically, but can be invaluable to scientists as genetic markers localizing regions on the chromosomes for tracking genetic inheritance.

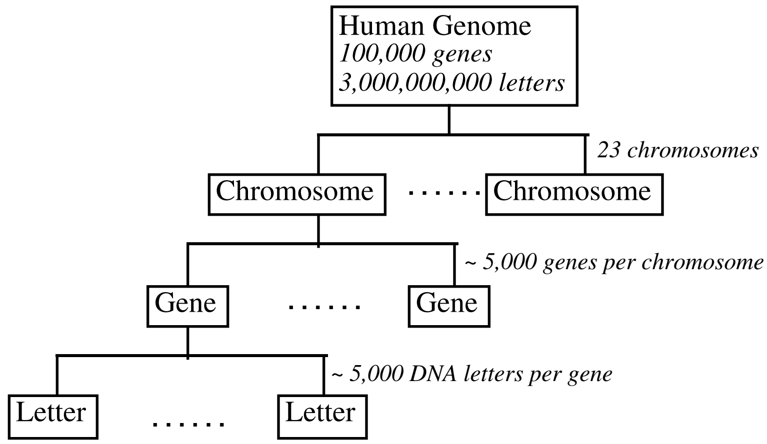


Figure 2.1. The genetic information hierarchy in the human genome. The entire genomic information is partitioned into 23 chromosomes. Each chromosome is a linear sequence of DNA letters, some of which are DNA "sentences" (genes) that encode certain biological functions, while others are merely "nonsense" or "whitespace" letters which have no known biological function.

Genes

Genes are DNA sentences along the chromosomes that encode specific functions. The "classical" genes are those that encode the recipes for constructing the proteins that the body needs. Other genes encode functions that determine physical traits such as hair color or increased susceptibility to heart diseases.

Genes are identically ordered along each linear chromosome for every normal human individual. The "versions" of the genes, however, may differ from one person to another, leading to the variations in traits (e.g. different hair color) that are observed in the population. These different versions of a gene are called its alleles. Every person has two alleles at every gene locus (except for some of the genes on the sex chromosomes) in the chromosomal pairs inherited from the parents. The genetic makeup of a person with respect to a particular gene is called the genotype, and it comprises two alleles for every gene (with the exclusion of some of the genes on the sex chromosomes).

In this dissertation, we will use the term *gene* to refer to functional DNA. This includes the protein-encoding and trait-determining DNA, as well as regulatory sequences such as promoters and enhancers. Therefore, a gene is defined in this dissertation as any inherited DNA sequence that encodes some biological function.

Markers

Markers are special classes of DNA sentences that are used by geneticists as distinctive landmarks along the chromosomes. Some of the markers are unique DNA sequences which occur only at a specific chromosomal location in everyone's genome (in the same species). These unique markers (e.g. sequence tagged sites or STSs) are useful in locating positions on the genome, and they are used extensively in physical mapping. Another type of landmark consists of genetic loci that are highly variable (i.e., polymorphic). This means that not everyone has the same version (allele) at that landmark location. In the (extreme) case of a completely polymorphic marker, chromosomes originated from different sources will inherit unique alleles. These alleles can serve as unique tags for chromosomes sharing a common origin.

In this thesis, we are concerned with this second type of marker for tracking inheritance. Such a marker must express a measurable form of *polymorphism* so that its inheritance can be traced in a pedigree. It should also be *stable,* with each allele inherited intact (that is, no mutation occurs) from parents to offspring. An ideal marker is one that is both stable and highly polymorphic, so that its alleles reliably indicate the origin of its chromosomal segment as it replicates in a family or population through multiple generations.

A marker can be any polymorphic segment of DNA, functional or nonfunctional. However, genes (functional DNA) are typically not very polymorphic, since any variation can be a lethal aberration eliminating the survival of the carrier. Therefore, genes are typically not very useful as markers for tracking complex inheritance patterns.

Much of the DNA text, however, consists of "nonsense" sequences which do not code for any known biological function. Long stretches of non-coding "fillers" called introns are found interspersed within the DNA sentences of the genes. Repetitive DNA, or *repeats*, also occurs abundantly and randomly throughout the genome. The Alu sequences (Schmid and Jelinek, 1982), VNTRs (Nakamura *et al.*, 1987), minisatellite arrays (Jeffreys *et al.*, 1985), and microsatellites (Gyapay *et al.*, 1994; Litt and Luty, 1989; Weber and May, 1989), are some examples of repetitive DNA. The non-encoding nature of these repetitive DNA elements leads to their relatively high degree of polymorphism; together with their abundant occurrence in the genome, these repetitive elements are extremely useful as genetic markers. In particular, the *microsatellites*, a class of repeats that are highly polymorphic, abundant, and easily assayed (Weber and May, 1989), have been used routinely by the geneticists to track inheritance of genetic traits in humans (Hearne *et al.*, 1992).

Microsatellites

The microsatellite family includes di-, tri-, and tetra-nucleotide repeats that are DNA words of the form "PR^*S", where P is a fixed prefix string, S is a fixed suffix string, R is the nucleotide unit of the repetitive sequence R^* with the length of R small (e.g., 2, 3, or 4), and "*" denotes Kleene star (Hopcroft and Ullman, 1979). Microsatellite markers have been found to occur abundantly in the human genome. For example, there are an estimated 100,000 CA-repeats (i.e., a microsatellite with R = "CA"), which are sufficient to cover the entire genome at high resolution.

Within the last decade, microsatellite markers such as the CA repeats (Weber and May, 1989) have become the polymorphic markers of choice for constructing high-resolution genetic maps (Gyapay *et al.*, 1994; Matise *et al.*, 1994) . These repeats are abundant and easy to find, occurring, on average, every 30,000 bp throughout the genome. They can be amplified *in vitro* using polymerase chain reaction, or PCR (Mullis *et al.*, 1986), thereby consuming little genomic DNA and requiring less time and effort than Southern blotting[1]. They also show a very high PIC (polymorphic information content) and can be informative for genetic linkage. Most importantly, because of their regular structures, the microsatellite markers are *length polymorphisms*, with each allele corresponding directly to the number of the repeated unit (n, in PR^nS). This means that geneticists do not need to sequence each and every DNA letter of the marker to determine the alleles. Instead, the alleles can be determined (i.e., genotyped) by simply sizing the amplified PCR products on a standard electrophoretic sequencing gel. These characteristics make microsatellites ideal for large scale, high throughput genetic studies, making most older markers obsolete.

With the discovery of the microsatellite markers, the older genetic markers (e.g., restriction fragment length polymorphisms, or RFLPs) have all but disappeared as markers for tracking inheritance. The PCR-based microsatellites are more informative and relatively easier and less expensive to genotype than the less informative bi-allelic markers or the RFLPs which require Southern blotting. The minisatellite arrays, their larger counterparts in the VNTR (variable number of tandem repeat) family, have larger sequence motifs (i.e. the repetitive unit R in "PR*S" ranges from 10 to 60 bp long), making minisatellites less

---

[1]Southern blotting involves size-separating DNA on a gel, then transferring the gel DNA to a filter ("blotting"), and then probing the blot with a second DNA by hybridization.

amenable to PCR amplification than the microsatellites. The minisatellites are also less common and not as evenly distributed along the human genome as are the microsatellites.

Among the microsatellites, the dinucleotide repeats such as the CA-repeats are the most widely used. The dinucleotides are also the best candidates for fine-mapping, because they are generally more polymorphic and densely spaced than the larger repeats, and they are also genetically stable, with an average mutation rate of about 10^{-4} per generation (Weissenbach *et al.*, 1992). The dinucleotides are also found in abundance in other mammalian species (Moore *et al.*, 1991), and are widely used in agricultural applications.

2.1.3. Meiosis, recombinations, and genetic variations

In sexually reproducing organisms, a special type of cell division called *meiosis* produces the sexual cells (eggs and sperms). During meiosis, the chromosomes are duplicated, followed by two consecutive divisions. In the male, this results in four haploid (cells with half the number of chromosomes) sperms, and in the female, one ovum and three polar bodies. With the number of chromosomes halved in meiosis, the sperm and the ovum can then merge during fertilization to form a zygote which contains the correct number of chromosomes.

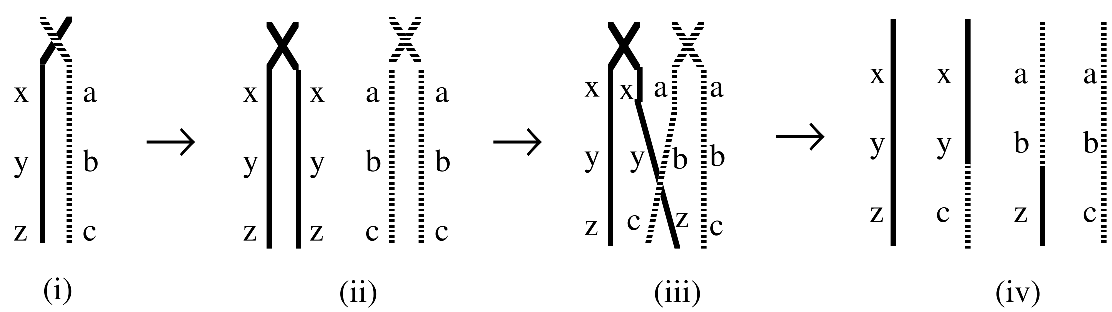


Figure 2.2. Crossing-over. (i) Each parent's chromosomal pairs may contain different alleles for the various genes on the chromosome. The example shown here has alleles x, y, and z on one chromosome, and alleles a, b, and c on the other for the three genes on this chromosome. (ii) During meiosis, the chromosome pairs double, resulting in two pairs of each chromosome in each parent. The exact copies are paired together, and the two pairs of homologous chromosomes line up side by side. (iii) Here, crossing-over takes place between two of the four homologous chromosomes. (iv) A total of four homologous chromosomes with different allele compositions are produced in each parent. The offspring may inherit any one of these four chromosomes from this parent.

If each parent simply passes on a half of the chromosome pairs to the offspring *intact*, there would be very limited genetic variation in the progeny. To further ensure genetic variation (a crucial factor for the species' survival), the parental chromosome pairs "cross over" during meiosis and exchange homologous segments before they split into halves (Figure 2.2). Thus, each of the chromosomes in the resulting haploid cells is actually a mixture of the two halves of the original parental pairs.

Meiotic recombinations can make it difficult for the geneticist to determine the origins of inherited chromosomal segments. When tracing complex inheritance patterns, the chromosomal segments must be sampled (genotyped) at a high enough resolution to resolve any ambiguities due to meiotic recombination. The discovery of microsatellite markers has made it possible to sample the genome at a high resolution of 10-20 cM. However, the current cost associated with microsatellite genotyping has precluded such dense genotyping, except at well-funded genotyping centers. By fully automating microsatellite genotyping, the associated cost can be lowered tremendously, making high throughput microsatellite genotyping more accessible.

2.2. Genetics

Thousands of common diseases in humans are known to have genetic causes. Discovering the causative genes for these diseases is essential for early detection and prevention, and for finding eventual cures. However, many common diseases (e.g. cancer, diabetes, heart diseases) are *complex* genetic diseases. To discover the causative genes for these diseases, we need to genotype large populations at a high genomic resolution in order to trace the associated complex inheritance patterns of the diseases. The number of genotypes needed can number near the millions, requiring full automation of genotyping.

2.2.1. The genetics of common diseases

Human genetic disorders can be grouped into two major classes: simple Mendelian diseases, and complex non-Mendelian diseases. A Mendelian disease is caused by a defective genotype at a *single* gene locus. Usually, only one disease mechanism is operating in a given family, and possession of the high-risk genotype is necessary for disease expression. Examples of simple Mendelian disease include cystic fibrosis, Duchenne's muscular dystrophy and Huntington's disease.

A non-Mendelian (complex) disease, on the other hand, is multifactorial and results from a complex interaction of multiple genetic and non-genetic (environmental) factors. Most common genetic disorders in humans (e.g. diabetes, colon or breast cancer, coronary heart disease, obesity, alcoholism and schizophrenia) are complex non-Mendelian disease. We can model a complex disease as one that is not entirely genetic; its genetic component merely imparts a certain predisposition toward the disease. This predisposition is triggered when an individual inherits defective alleles of some small set of the controlling genes. There may be several sets of controlling genes, such that individuals who have the same genetic predisposition may carry different sets of defective alleles. Whether each of these individuals will eventually develop the disease depends on the combined effects from the genetic and non-genetic environmental factors (e.g. diet, exposure).

There are many other factors that complicate the genetics of common diseases further. For example, false-negative cases can be caused by reduced penetrance or late age-of-onset of the disease, while false-positive cases can be caused by the presence of phenocopies or non-genetic cases. Etiologic heterogeneity and genetic interaction are also not uncommon in complex diseases. To begin to unravel all the confounding factors associated with complex diseases, it is necessary to have a large amount of genetic data. Geneticists have had great success with simple Mendelian diseases (Ott, 1991); they are now beginning to study and dissect complex genetic traits (Lander and Schork, 1994; Risch, 1990; Risch, 1991).

2.2.2. Finding disease genes: positional cloning

Our ability to understand, diagnose, and (eventually) find a treatment for human genetic diseases depends largely on our ability to locate and clone genes. By cloning the genes, the disease-related biochemical malfunctions can be studied extensively. Cloning is a technique in molecular biology to obtain an interesting piece of DNA (e.g. a gene) in a large quantity that is convenient for detailed analysis and further experiments. The DNA segment to be studied is isolated and then inserted into a cloning vector (e.g. yeast artificial chromosomes, plasmids, or cosmids) that is capable of integrating the foreign DNA into itself without losing its capacity for self-replication. Through these cloning vectors, foreign DNA can be introduced into host cells where it can be reproduced and studied at length.

To clone a gene, we must first know where it is on the chromosomes. However, locating disease genes is a Herculean task. Consider the following scenario: a disease is known to run in families, following, say, a simple Mendelian pattern of inheritance. The disease gene is a small DNA word or sentence that lies somewhere among the one hundred thousand genes on the twenty-three pairs of human chromosomes containing a text of three billion DNA letters. Because the biochemical pathways of the disease are not yet known, there is no biological clue to help us locate the defective gene. In fact, we do not even know where most of the genes are on the chromosomes. How do we proceed to isolate and clone the defective gene without even knowing what it is?

This is a classic search problem — a problem that is also paradigmatic in computer science, in particular, artificial intelligence (Rich and Knight, 1991). The search space is vast: a prohibitive three billion characters of text, the content of which is largely incomprehensible to the humans (for now). To read this text character by character (bottom-up) would take an astronomical amount of time, effort, and money. It is more feasible to adopt a top-down search strategy in which we incrementally prune the global search space (~ 3 gigabase, or Gb) to the resolution level of a single chromosome (~ 100 to 300 megabase, or Mb), and then to chromosomal regions that are small enough (e.g. ~ 100 kilobase, or kb, to 1 Mb) labor-intensive local searches. The geneticists can then use physical maps of cloned DNA fragments to identify candidate genes within the chromosomal regions. The candidate genes can then be meticulously sequenced and tested until the true disease gene is found.

This top-down disease gene discovery approach is known as <u>positional cloning</u> (Collins, 1992; Collins, 1995), a process that is at the heart of the current genetic revolution[2]. What we need for this top-down approach is a good evaluation function to prune the search space from the gigabase genome down to kilobase chromosomal regions. One well-utilized evaluation function for pruning the genomic search space is the matching between the inheritance patterns of the genetic trait (in this case, a disease) and the shared chromosomal inheritance patterns of affected individuals in a population. For example, consider two

---

[2]Positional cloning is not the only method for discovering disease genes. One increasingly popular approach is the EST (Adams *et al.*, 1991) or expressed gene method. It involves sequencing short regions of cDNA clones isolated from a library of a particular tissue such as brain, and then looking for those (expressed) genes that have interesting amino acid motifs, and comparing the results of normal and affected tissues.

related individuals (e.g., siblings) who are both affected by a genetic disease. The chromosomal regions that they inherit in common from their parents (about 50% of the genome) are likely places to look for a causative gene. These shared genome regions are *identical-by-descent* (IBD) for the pair. When the IBD regions of many other paired individuals are determined, the "intersection" of these regions can point to a small region (e.g., ~1 Mb) that contains the causative gene. All that is needed is an efficient way to determine the shared IBD regions of related individuals.

The determination of this sharing is done by *genotyping*. A genotype can be viewed as a small point-like sampling of an individual's DNA at a known chromosomal location (e.g. a genetic marker). A non-polymorphic DNA point (e.g., a highly conserved gene) would have only one allele for everyone, and it would be useless for differentiating two individuals. The best genetic markers are highly polymorphic with different alleles in a population so that when two relatives share the same alleles (*identical-by-state*, or IBS), it tends to imply that they both inherit the same chromosomal region (IBD). By sampling an individual's genome with a large number of highly polymorphic, closely spaced markers, the geneticist can obtain a high resolution "snapshot" of an individual's genome which can then be compared with the genetic snapshots of the individual's relatives. With the advent of very high resolution genetic maps, such genotypic snapshots have become the mainstay of both regional and genome-wide searches for genes (Hearne *et al.*, 1992).

2.2.3. Tracking genetic inheritance: dense genotyping

To accurately assess the risk of an individual to a disease, it is necessary to determine whether the individual carries the defective genotypes at the respective gene loci. Ideally, the individual's genes should be sequenced so that the exact genetic content can be used to ascertain risk. However, there are thousands of bases in a gene (Figure 2.1). As such, it is highly expensive and time-consuming to sequence just one gene. To begin to assess the risk of a single individual for all the common human genetic diseases, hundreds of millions of DNA bases will have to be sequenced for each individual.

There is a much more feasible alternative. Using a search strategy similar to that in positional cloning for finding disease genes, it is possible to determine if an individual carries the defective alleles without actually sequencing the genes. As in positional cloning, we can tag the genome using polymorphic markers that occur along the chromosomes at high resolution. The chromosomal sharing information between two related individuals can be reliably inferred from their genotypes. If an individual had inherited a chromosomal segment containing the disease gene from an affected individual in the family, then we can conclude that this individual must also have inherited the defective allele (without having to know the actual allele of that gene). Furthermore, if the genotyping were done using a standard set of microsatellite markers that covers the entire genome, the genotyping data could be *re-used* to compute the risk for many different genetic diseases, without having to meticulously sequence any new genes.

Of course, key to using chromosomal inheritance patterns to reliably assess risks is the ability to determine the origins of each inherited chromosomal segments (particularly, the segment that may contain the disease allele from the affected parent). If the parents had passed each half of their chromosome pairs on to the offspring *intact*, then, as long as we have a single informative marker on that chromosome, we can tell if an offspring has inherited the defective allele because the allele must necessarily be inherited together with the chromosome from the parent. However, because of meiotic recombinations, parents do not pass on entire chromosome strands to their offspring intact. Instead, the offspring receives a chromosomal strand that is a combination of the original parental chromosome pair. When the markers flanking a chromosomal segment are not informative (i.e. IBS does not imply IBD), the parental origin of that chromosomal segment can be ambiguous. Figure 2.3 shows an example in which it is impossible to determine whether the alleles in a marker (the second marker) in the siblings actually originated from the father or from the mother.

The problem of disambiguating the parental origins of alleles or chromosomal segments is known as the *haplotyping problem*. When the genetic markers (sample points) are far apart or when they are not sufficiently polymorphic, recombination events may remain undetected between uninformative markers, and inheritance of an affected allele cannot be inferred with certainty from the genotyping information of the flanking markers. To solve this problem, various analytic solutions have been proposed (Weeks *et al.*, 1995; Wijsman, 1987). However, the haplotyping problem is actually an inherent limitation as in the example depicted in Figure 2.3. If the parental origins of the chromosomal segments containing disease alleles cannot be determined unequivocally, it is not possible to assess the risk of the children to the disease accurately based merely on the genotypes of flanking markers.

It turns out that the problem can become nonexistent if the chromosomes were *densely genotyped.* By sampling the genome at a high resolution with closely-spaced polymorphic

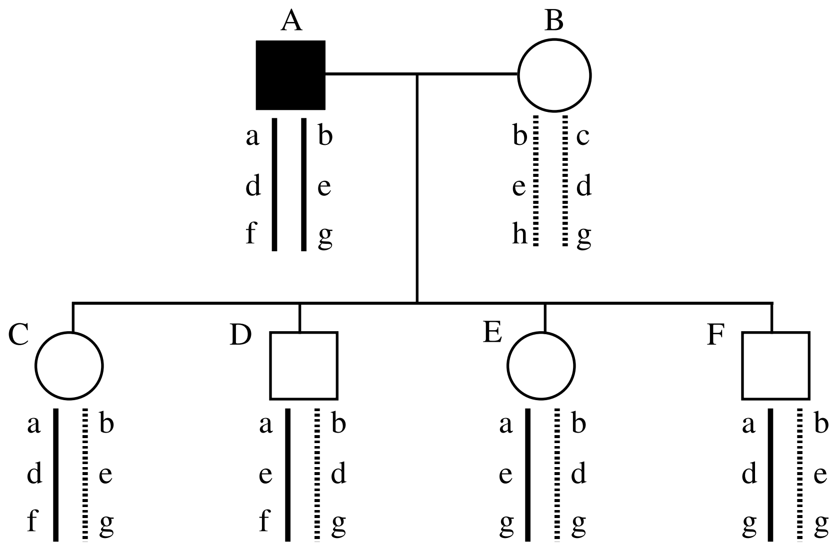


Figure 2.3. The haplotyping problem. Two individuals may have the same genotypes but their haplotypes can be different. Both C and D have the same set of alleles for all three markers. However, for the second marker, C receives allele "d" from her father (who is affected, as indicated with a darkened node) whereas D receives allele "d" from his mother (who is healthy). In C, there is only one recombination event from the maternal side. For D, his haplotype is a result of three recombination events: a double recombination at the paternal side and one at the maternal side. These recombination events are not unlikely when the markers are far apart (low-resolution genotyping). The other pair of siblings also share the same set of alleles, except that E and F's haplotypes both resulted from the same number of recombination events (two, one from each parent).

markers, recombination events are unlikely to remain undetected since (1) the likelihood of having all flanking markers uninformative is negligible as the number of flanking markers increases, and (2) the likelihood of having more than one recombination event occurring between two closely-spaced flanking markers[3] is also negligible. With the full automation of microsatellite genotyping, it might be feasible to sample genomic points at high resolution efficiently and inexpensively so that the haplotyping problem is solved.

[3]If an even number of recombination events had occured between a pair of flanking markers, the markers' genotypes would be indistinguishable from the genotypes in the case of *no* recombination. If an odd number of recombination events had occured, the genotypes would be the same as in the case of only *one* recombination event. Thus, if there were more than one recombination event between the markers, the genotypes alone would not be informative enough to distinguish between multiple and single (or no) recombination events.

2.3. Biotechnology

The invention of the polymerase chain reaction (PCR) technique by Kary Mullis (Mullis *et al.*, 1986) in the mid 1980s was a major breakthrough in modern molecular genetics. With PCR, it became possible to rapidly produce millions of copies of a *specified* DNA sequence in vitro. The next related breakthrough for genotyping was the application of PCR to microsatellite markers such as the CA-repeats by Weber and May in 1989 (Weber and May, 1989). The microsatellites provide an abundant class of closely spaced highly informative markers that can be amplified by PCR. In addition, they can be genotyped simply by their size differences (using gel electrophoresis), instead of having to meticulously sequence the DNA segment for each allele. Using sophisticated high throughput sequencer machines (for example, the ABI/377 and the Pharmacia/ALF machines), it is now possible for a genetic laboratory to generate vast amounts of data daily. The main bottleneck that remains is the tedious task of genotyping all that efficiently generated data.

2.3.1. PCR and PCR stutter

The enzymes that make DNA in the cells are called <u>DNA polymerases</u>. These DNA polymerases will catalyze the duplication of DNA only in the presence of pre-existing DNA templates. The PCR process exploits the biology of the DNA polymerases to replicate DNA *in vitro*. Using the selectivity of short DNA oligonucleotide primers towards specific DNA templates, the PCR process can be directed to synthesize a *specific* region of DNA. By iterating the replication cycle, millions of copies of the specified DNA regions can be rapidly generated at an exponential rate. The PCR process is a relatively straightforward laboratory technique, requiring only a very small amount of the source DNA.

PCR cycle

The PCR cycle is a 3-step process (Figure 2.4):

(1) *Denaturation*. The first step of the PCR process is to create single-stranded DNA templates for replication. The double-stranded DNA molecules are separated ("denatured") to form single DNA strands by an increase in temperature (to around 94°C);

(2) *Primer annealing*. The second step is to specify a region on the DNA to be replicated. To do so, we label the starting point for DNA synthesis with a oligonucleotide (synthetic) primer that anneals (at a lower temperature, say, 30-65°C) to the template at that point. By supplying a pair of flanking primers, only the DNA region between the flanking primers will be amplified .

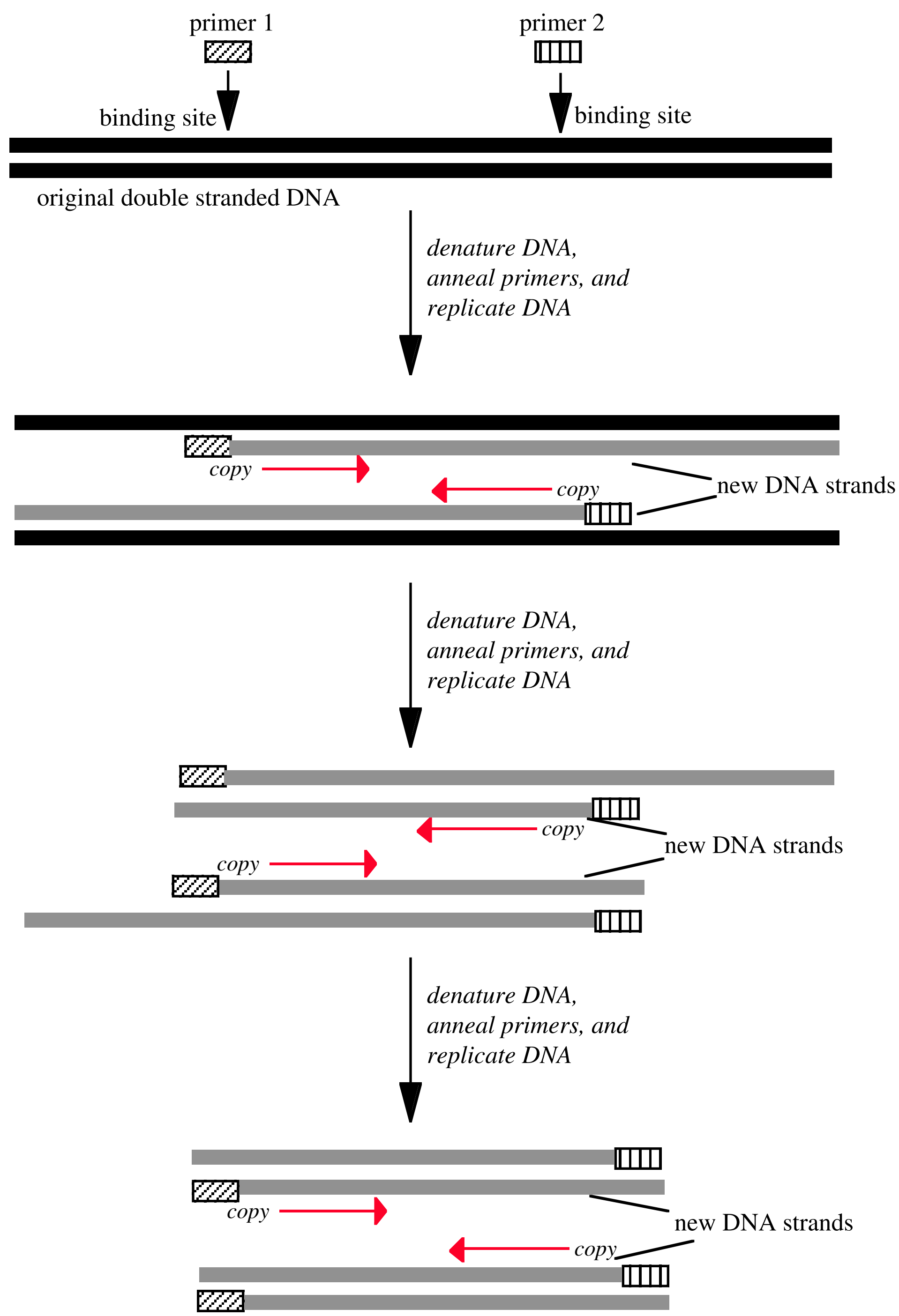


Figure 2.4. The polymerase chain reaction (PCR). By annealing two flanking primers (primer 1 and 2), the desired short DNA segment is synthesized by a DNA polymerase. The process is generally repeated 30-60 times to generate millions of copies of the DNA segment flanked by the two primers.

(3) *DNA synthesis.* The third step is to replicate the specified DNA region. With the primers annealed to the binding sites to direct replication, the DNA polymerase (e.g. *Taq* polymerase) synthesizes new complementary strands, which can then be denatured, annealed with primers, and replicated again.

The concentration of the target sequence is doubled with each PCR cycle. After n PCR cycles, and there are 2^n replicated DNA molecules in the resulting PCR mixture. Typically, the PCR cycle is repeated for as many as 30-60 cycles, taking several hours to complete.

PCR stutter

Ideally, the PCR product should contain only DNA fragments that are exact copies of the target DNA segment being amplified. When the target segment contains repetitive DNA regions, a series of secondary fragments is often generated in addition to the target segment. These extraneous fragments are typically shorter than the target segment, creating the characteristic "stutter" or "shadow" bands that are observed when the DNA in the PCR product is size-separated by gel electrophoresis (see Figure 2.5).

Figure 2.5. Autoradiogram of denaturing polyacrylamide gel showing the typing of 5 individuals at a microsatellite marker. Each of the five columns (lanes) contains the result of size-separating the PCR product of the DNA sample from a different individual. Because of the pronounced stutter bands, it is difficult to tell if the resulting pattern in each lane was caused by a single allele (homozygous) or a pair of alleles (heterozygous) in the DNA sample, or what these alleles are. *(Provided by Dr. Mark Shriver, University of Pittsburgh.)*

PCR stuttering is generally attributed to the slippage (misreading) of the polymerase during the amplification of the template DNA strands (Hauge and Litt, 1993). It is particularly pronounced with markers that are tandem repeats of short nucleotide sequences (e.g. dinucleotides). The slippage of polymerase creates new DNA template strands that are shortened by a length equal to an integral multiple of the length of the repeating units, which are then amplified in subsequent PCR cycles. Shorter error templates may also be created in the subsequent cycles, causing the trail of shadow bands when the PCR product is size-separated on an electrophoretic gel. The stutter bands from one allele may overlap with those from the other, creating complex band patterns that can be confounding even to the trained eye of a human technician.

Molecular biologists have made numerous attempts to reduce or eliminate the PCR stutters:

1. Modifying the PCR conditions. This approach works to a point (Odelberg and White, 1993) but generally does not remove the artifact completely.
2. Using microsatellite markers with smaller alleles (e.g. using $(CA)_n$ markers with n small). Smaller alleles are generally associated with fewer stutter bands as there are fewer repeats for the polymerase to "slip" over. However, this approach drastically limits the choice of markers, and the smaller n also implies lower polymorphism (fewer possible alleles) for a marker.
3. Using tri- and tetranucleotide repeats. Markers with larger repeats tend to display little or no stutter artifact. However, the larger repeats are also more complex, less informative, and consume more "real estate" on the gel. They are less densely distributed over the genome than the dinucleotides, and are thus less suitable for fine analysis.

These conventional experimental efforts have failed to remove the stutter artifact completely. As a result, current genotyping systems either rely on the human to call the alleles, or attempt naive allele calling by focusing on the highest peaks (Mansfield *et al.*, 1994; Ziegle *et al.*, 1992). The latter approach is problematic with single (homozygotic) or closely-spaced alleles, and widely-separated alleles with pronounced relative amplification. It will not work when more than two alleles are present, say, when DNA are pooled from multiple individuals in a population.

It fact, it may even be unwise to try to remove the PCR stutter from the data totally. Without the signature stutter pattern, it may be impossible to distinguish a spurious noise band from an actual data band (Schwengel *et al.*, 1994). With markers that exhibit stuttering, the *expected* presence of the PCR stutter signatures can be used to disambiguate true data from spurious noise and contamination bands. If we can handle the shadow bands, data analysis for the stuttering markers (e.g. the dinucleotides) can actually be more robust against noise and contamination than the stutterless markers (e.g. most tri- and tetranucleotides).

2.3.2. Gel electrophoresis

Gel electrophoresis is a standard laboratory technique to separate DNA fragments by size. When exposed to an electric field, a mixture of DNA molecules travels through a medium (e.g. an agarose or acrylamide gel) at different rates depending on their molecular sizes. By labeling the DNA molecules with a radioactive or fluorescent molecule, we can detect the relative gel positions of the DNA molecules after they have been size-separated and deduce their relative size differences. If it is necessary to determine the *actual* sizes of these DNA molecules, we can run molecular weight (MW) standards or DNAs with known sizes on the gel together with the unknown DNA molecules (Figure 2.6). We can then use the calibration curve generated from the gel positions of the size standards to interpolate the actual sizes of the DNA molecules.

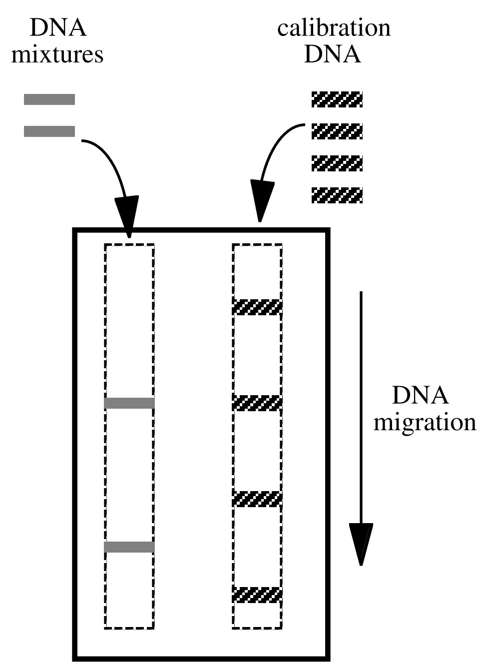


Figure 2.6. Gel electrophoresis. DNA of different sizes migrate at different speeds in a gel subjected to an electric field. Typically, calibration DNAs with known sizes (molecular weight standards) are run together with the unknown DNAs so that the unknown DNAs can be typed by size interpolation.

Gel electrophoresis is essential for genotyping microsatellite markers. As microsatellites exhibit length polymorphism, the different alleles directly correspond to different DNA sizes. Therefore, instead of sequencing the DNA fragments (which can contain 100–500 DNA letters) for the exact DNA content to call the alleles, we only need to size-separate the PCR product on an electrophoretic gel, and use the calibrated DNA sizes as the alleles for the markers' genotypes.

2.3.3. High throughput genotyping

Because large amounts of genetic data are necessary for statistical genetic linkage analysis, the main emphasis of genotyping technology is to generate as much data as rapidly as possible. Figure 2.7 shows the various ways to multiplex gel readout for attaining high throughput in genotyping:

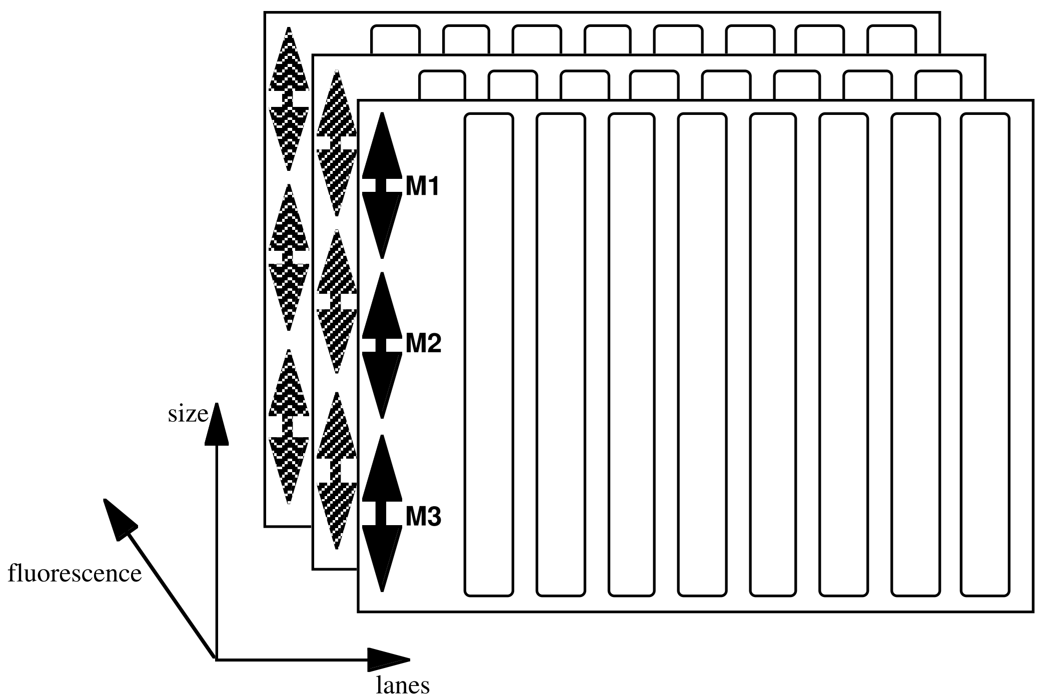


Figure 2.7. High throughput genotyping. Multiplexing can occur in three dimensions: *lanes, size,* and *fluorescence.* First, the gel is divided into many lanes (vertical columns) to allow multiple genotyping on the gel. Second, markers (e.g., M1, M2, and M3 as shown) with non-overlapping allele window can be genotyped in the same lane. Third, by labeling markers with different fluorescent dyes, markers with overlapping allele windows can be multiplexed in each lane (but in a different "plane" or dye). Here, three fluorescent dyes are used to multiplex a total of nine markers.

- *Multiple lanes.* The gel is divided into as many lanes as possible so that different genotyping experiments can be run in different lanes on a single gel. Currently, each gel contains 30-96 lanes, and this number will continue to increase because of the constant demand in data throughput.

- *Multiple size windows.* Markers that have non-overlapping allele windows can be analyzed together in the same lane because their data will occupy different regions along the lane. PCR products from three to six different marker assays are routinely pooled in a single gel lane for multiplexed readout.

- *Multiple dye labeling.* In multi-channel fluorescence DNA sequencers, additional multiplexing can be achieved by using different fluorescent dyes to label the markers. This allows markers that have overlapping allele windows to be run in the same lane, but emit data from relatively disjoint imaging "planes" (fluorescent dye). For example, the ABI (Applied Biosystems) machines are four-color systems, in which three of the dyes are typically allocated for genetic markers and one for running the MW sizing markers. This setup increases data throughput by three-fold, while providing each lane with its own size standards calibration curve.

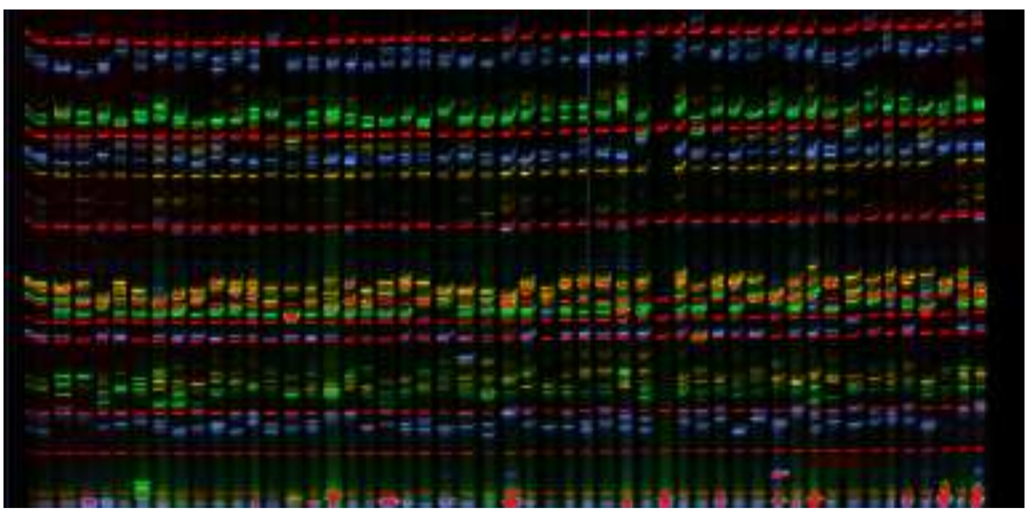


Figure 2.8. An example gel with 50 lanes and dye-multiplexed marker data. PCR products of DNA samples were labeled with one of four fluorescent dyes so that the genotyping experiments could share the same gel lanes and size windows. Here, one of the dyes was used solely for running internal MW size standards. On this gel, the bands of the MW standards formed horizontal rows amidst the marker bands since the same size standards were run in every lane. *(Provided by Dr. Vicki Magnuson, National Institutes of Health.)*

With an average of 12 markers per lane (4 size windows per lane and 3 fluorescent labels) and 32 lanes (as shown in Figure 2.8), about 400 different microsatellite experiments can be read out on a single multiplexed gel. Current efforts to increase data throughput will continue to increase the number of lanes and fluorescent dyes per gel so that thousands or more genotypes can be produced on a single gel. This super-efficiency in data generation must be coupled by the full automation of data analysis, for even a highly staffed laboratory will quickly be inundated with a flood of data if each of the genotypes has to be analyzed by hand, creating a genotyping bottleneck.

2.4. The Bottleneck: Genotyping

Almost all the major steps in the genotyping pipeline have been successfully automated: robotic sample preparation, PCR amplification, gel electrophoresis, and gel data digitization. The *genotyping* (gel data analysis) step has remained the critical bottleneck precluding full automation. Nearly all laboratories require an experienced human technician to visually inspect the microsatellite data, a task that is tedious, error-prone, time-consuming and expensive. In fact, roughly half of the error and cost (about $2-3 total per genotype) in current high-throughput microsatellite-based genotyping is attributable to the need for human operators to semi-automatically score the data. A system that can directly analyze the gel image files from automated DNA sequencers, and automatically detect, calibrate, quantitate, and genotype the marker data on the gel will remove this costly bottleneck and achieve the requisite throughput in genotyping studies.

3. Problem Overview

The data flow of the genotyping problem is straightforward: given a digitized gel image as input, output the genotypes for all the genotyping experiments multiplexed on the gel (Box 3.1). The underlying tasks, however, are nontrivial:

1. Data retrieval: How do we parse a highly-multiplexed gel image to retrieve segments of signal intensity profiles (electropherograms) that contain the data for each multiplexed genotyping experiment?
2. Data quantitation: How do we extract from the continuous intensity profile of an electropherogram discrete data bands binned with molecular units and quantified with DNA concentration measures?
3. Data deconvolution: How do we reduce a convoluted series of quantitated data band patterns into two consistently labeled alleles?

Input:	a digitized multi-dimensional gel image on which hundreds to thousands of genotyping experiments are multiplexed.
	↓
	retrieval
	↓
	quantitation
	↓
	deconvolution
	↓
Output:	a pair of consistently labeled alleles for each of genotyping experiments.

Box 3.1. The overall data flow in the genotyping problem. The input data is a highly multiplexed gel image, from which we retrieve the data window for each multiplexed experiment for analysis, quantitate the extracted signal data into molecular units, and deconvolve the complex data patterns to accurately call the two alleles in the genotype.

3.1. Retrieving data from gel

As we have discussed previously, current biotechnology allows hundreds to thousands of genotyping experiments to be multiplexed on a single gel in three dimensions:

- *width*: by partitioning the gels into vertical tracks or lanes, genotyping experiments can be run in parallel on a single gel in different lanes;
- *length*: by pooling markers with non-overlapping size windows together, they can be run out on the same gel lane; and
- *plane*: by labeling markers with different fluorescent dyes, genotyping experiments can be run in the same size window and gel lane, emitting data signals in different "dye planes".

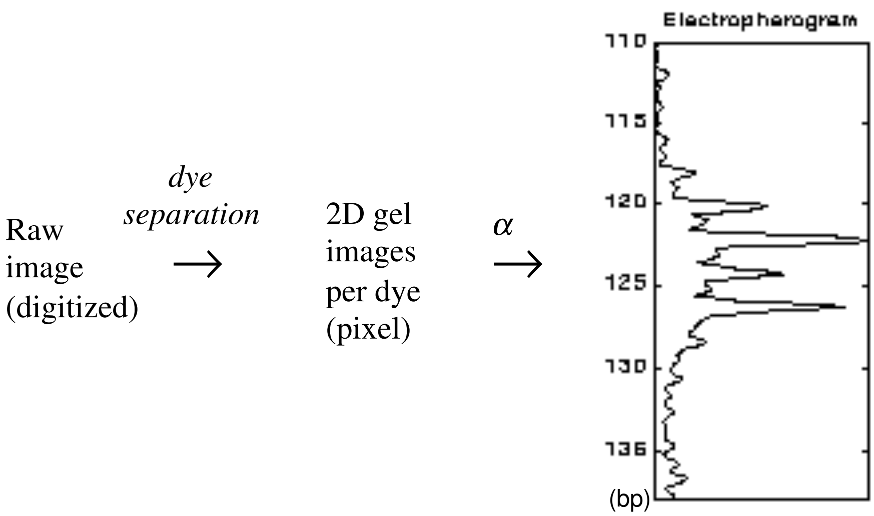


Figure 3.1. Retrieving data from gel. First, the raw digitized image from the DNA sequencer is separated into two-dimensional gel images for each fluorescent dye used in the experiment. Then, the coordinate transformation function α converts the gel image into linear electropherogram traces of microsatellite data in each lane, calibrated by DNA size instead of pixel.

Figure 3.1 shows how we can systematically undo the multidimensional multiplexing systematically to retrieve data[4] from a gel for analysis. First, we undo the "plane" multiplexing by a process called *dye separation.* After we have separated the data into different dye planes, we undo the "width" multiplexing by a process called *lane tracking* which converts the two-dimensional gel images into one-dimensional intensity profiles (electropherograms). With the one-dimensional electropherograms, we can then undo the

[4]Most DNA sequencer systems provide on-line digitized scanning during gel runs, so the raw data is often generated in a computer-readable format. If not, the gel data may be digitized using high resolution scanners. We will assume in this dissertation that the input gel images are digitized.

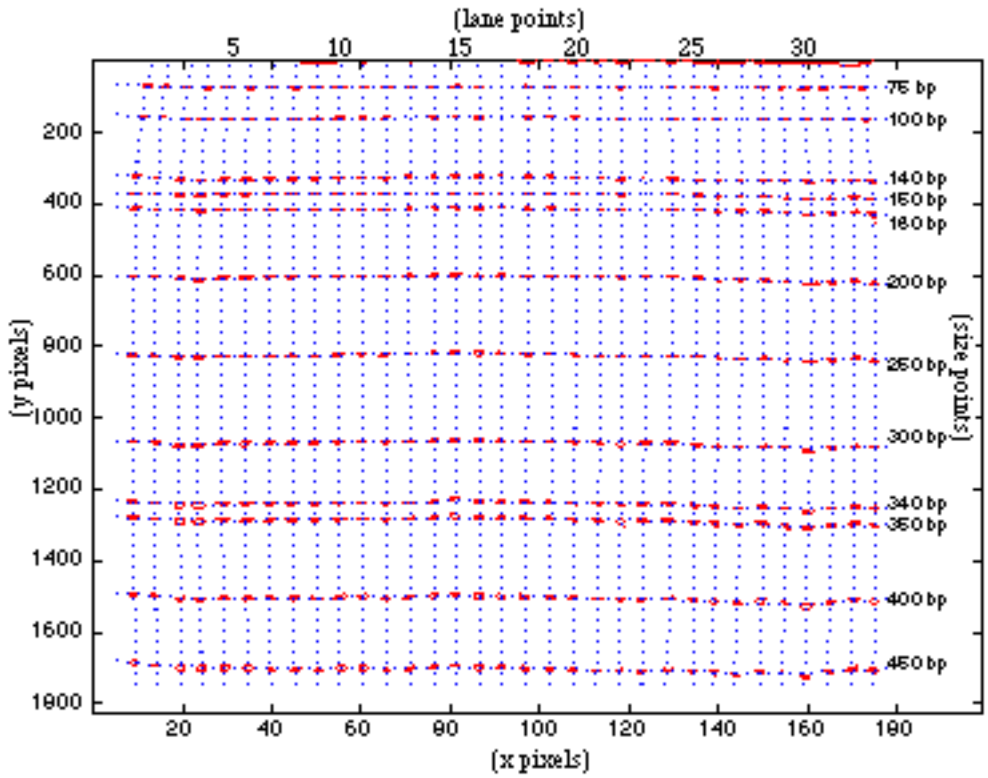


Figure 3.2. A sizing grid constructed from the MW size standards data. The vertical grid lines are the lane traces; the horizontal grid lines are contour lines of identical MW mobility and size. The sizing grid is useful for localizing expected marker allele events on the gel image.

"length" multiplexing by identifying regions on the gel lanes that contain marker data for analysis. As the data regions are defined by the size ranges of the possible marker alleles, it is necessary to translate the image pixel coordinates into molecular weight units first. We call this process *molecular weight (MW) calibration.* Once a lane has been calibrated in MW sizes (bps), we can then zoom in to regions on the lanes based on the possible allele sizes of the size-multiplexed markers. Excluding the dye separation step (since this is typically done by the DNA sequencers as part of the data generation step), the overall operation can be described with a *coordinate transformation* function:

$$\boxed{\alpha: \langle x, y\rangle \text{ pixels} \leftrightarrow \langle \text{lane, size}\rangle \text{ points}}$$

This coordinate transformation function is by a *sizing grid* that can be constructed based on the MW size standards data, as shown in Figure 3.2.

3.2. Quantitating data bands

From our understanding about PCR and gel electrophoresis, we know that the fluorescent signals detected by the laser sensors were emitted by *discrete* classes of DNA fragments that have *integral* lengths in bps. Therefore, after we have localized the marker data for a genotyping experiment on a gel using the sizing grid, we must *discretize* the continuous signal intensity profile in the electropherogram extracted from the image. To convert the image data into the corresponding "molecular space", we must (a) reduce the continuous signals into a discrete series of marker bands binned with integer sizes, and (b) quantitate each detected discrete data band into a DNA concentration measure (Figure 3.3). The overall operation can be described with the following transformation function:

$$\boxed{\beta: \text{ image data + sizing grid} \rightarrow \{\text{<bp, concentration>}\}}$$

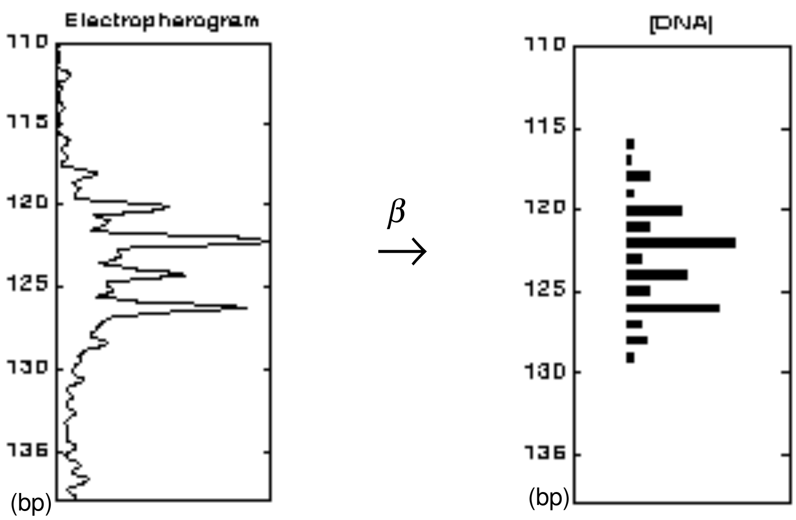


Figure 3.3. Quantitating the marker data bands. The transformation function β discretizes the continuous signal profile in an electropherogram. This involves detecting the marker bands from the background noise, and assigning to each band an integral size (in bp) and a value that corresponds to the concentration of the DNA fragments of that size in the PCR product.

3.3. Calling the alleles

PCR amplification of a short tandem repeat allele typically generates additional DNA fragments, creating more marker bands than the actual alleles present in gel electrophoresis. These extraneous stutter bands must be eliminated mathematically to recover the underlying alleles in the genotype, as shown in Figure 3.4. The requisite transformation, then, is a deconvolution function γ:

$$\boxed{\gamma: \text{ quantitated data with stutter} \rightarrow \text{genotypes}}$$

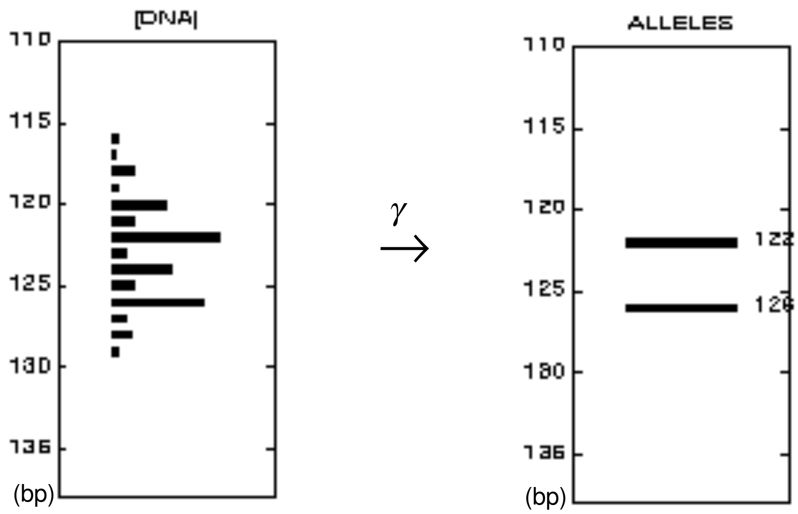


Figure 3.4. Calling the alleles. As a result of PCR stuttering, complicated data band patterns are observed. We must decide which of the bands (at most two) amidst the numerous convoluting stutter bands correspond to the true alleles in the genotype.

It is nontrivial to remove PCR stuttering from the quantitated data, as (a) PCR stutters from two alleles may superimpose onto one another, (b) stuttering patterns may differ from one allele to another, with the smaller alleles generally displaying steeper stutters than the larger alleles, and (c) there may be differential amplification between the two alleles in the genotype as the alleles compete for amplification during PCR. Without properly accounting for these artifacts, a human genotyper must be relied upon to call the alleles by visual inspection.

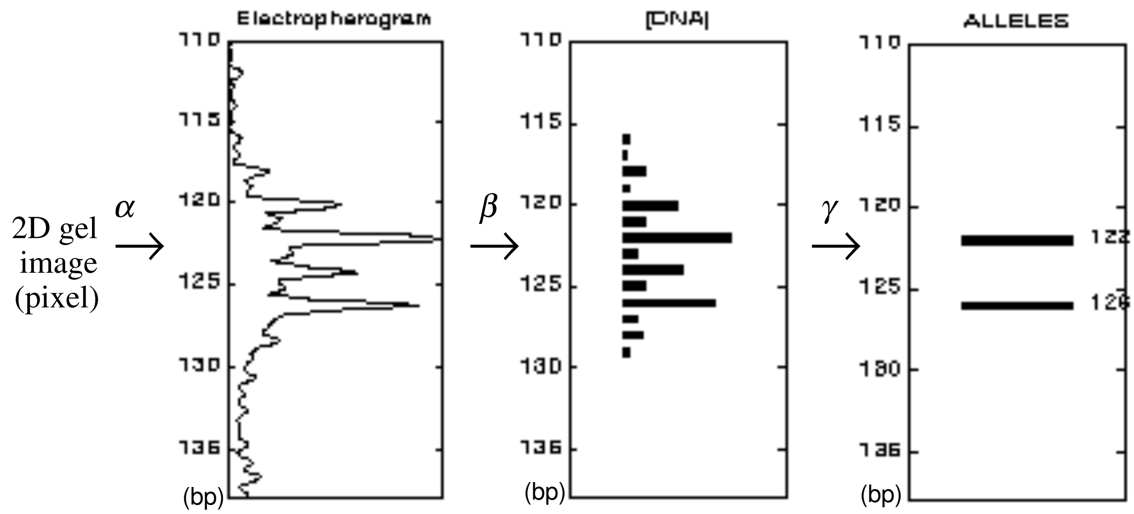


Figure 3.5. An automated genotyping system takes a two-dimensional gel image, executes three transformations α, β, and γ, and outputs the genotypes of the markers on the gel. The first transformation α converts the gel image into linear electropherogram traces of microsatellite data in each lane, calibrated by discrete DNA sizes. Then, β quantitates each detected data band and computes the DNA concentrations. Finally, γ transforms the quantitated stutter data into the two underlying alleles.

3.4. Summary

As depicted in Figure 3.5, a fully automated genotyping system analyzes the gel electrophoresis images from automated DNA sequencers by correctly detecting and determining the lane, size, and relative DNA concentration of every data-related band detected on the gel, and then mathematically removing the PCR stutter artifacts from the marker data to correctly call the genotypes. Box 3.2 summarizes the three requisite transformations that we have identified in this chapter:

1. Coordinate transformation

   $\alpha:$ <x, y> pixels → <lane, size> points
2. Data band quantitation

   $\beta:$ image data + sizing grid → {<bp, concentration>}
3. Stutter data genotyping

   $\gamma:$ quantitated data with stutter → genotypes

Box 3.2. Data transformations in solving the genotyping problem: (1) image pixel coordinates are transformed to facilitate data extraction, (2) data bands are quantitated into molecular units, and (3) stutter data tare deconvolved into recover the true alleles in the genotypes.

In the next three chapters, we will describe, with examples from *real data*, how we *computationally* handle each of the three data transformations. To further illustrate the power of the computational approaches that we advocate in this dissertation, we will describe, in a following chapter, how we can enable a new functionality called *pooled genotyping* with a simple generalization of our computational solutions.

4. Grid Construction

In Chapter 1, we outlined the various problems that make full automation of microsatellite genotyping difficult. The first problem was:

• <u>Data tracking</u>: how to automatically track the relevant data regions on a highly multiplexed gel with hundreds to thousands of genotyping experiments?

In high throughput microsatellite genotyping, data multiplexing can occur in up to three dimensions: dyes, lanes, and marker size windows (as shown in Figure 4.1). To unravel this intricate multiplexing, we perform the following transformations:

<u>Stage 1</u>. *Dye separation.* Separating the multi-dimensional raw data into two-dimensional image planes for each of the dyes in the system;

<u>Stage 2</u>. *Lane tracking.* Reducing the two-dimensional image planes into one-dimensional electropherograms for each of the loaded gel lanes; and

<u>Stage 3</u>. *MW calibration.* Calibrating the electropherograms from the image units (pixels) into in molecular units (bps) along each of the gel lanes, so that marker

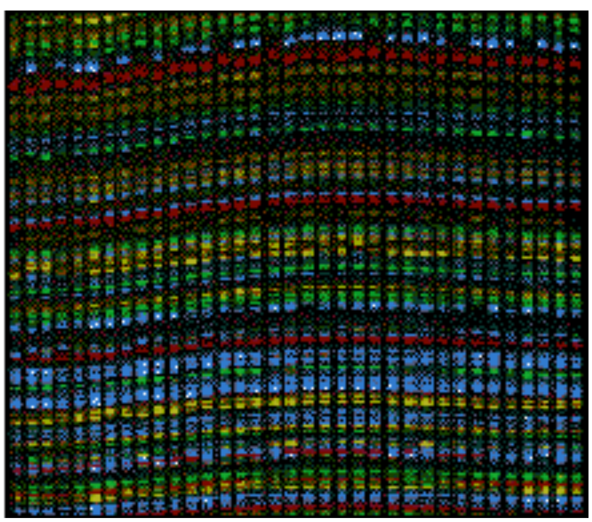


Figure 4.1. A highly multiplexed gel on which hundreds of genotyping experiments were ran in parallel. The experiments were highly multiplexed in terms of lanes, size windows, and fluorescent dyes. One this gel, a total of three fluorescent dyes were used in labeling the DNA samples to allow multiple genotyping experiments to share the same gel lanes and size windows. A fourth dye was dedicated to labeling the internal MW size standards for accurate sizing.

knowledge (e.g. the expected allele size window) can be used to localize data regions on the electropherograms for analysis.

4.1. Problem

Commercial DNA sequencers (such as the ABI machines) exploit fluorescent primer-labeling technology to generate maximally multiplexed gel data labeled in different dyes. These machines are usually equipped with internal dye-separation mechanisms, so that dye-separated data are automatically generated. Therefore, in this dissertation, we will assume that all gel images are already dye-separated by the sequencers, and that the dye-separations are generally adequate. In the event that the internal dye separation is imperfect, some post-processing of the data will be necessary. For this, we will describe in Section 4.2 how to filter out artifacts (e.g. dye bleedthroughs) from gel images that are imperfectly separated.

The main data tracking problem is in tracking the gel lanes and detecting the MW bands. This tracking involves the construction of a sizing grid α that transforms pixel coordinates into lanes and molecular sizes:

$$\alpha: \langle x, y\rangle \text{ pixels} \rightarrow \langle \text{lane, size}\rangle \text{ points}$$

Sizing grid construction has not previously been successfully automated. Even in the most advanced genotyping systems, the human eye is required to resolve various confounding data artifacts on the gels. Figures 4.2 (a,b), 4.3 (a,b), and 4.4 (a,b,c) show the common artifacts (from actual gels) that make automated sizing grid construction difficult. Each figure shows a gel image for the TAMRA dye, the dye used for labeling the MW size standards. The same set of TAMRA-labeled MW sizes has been loaded in the lanes on each of the gels, forming the rows of bands observed. The common confounding data artifacts are:

- *Dye bleedthroughs.* Imperfect dye separation causes "bleedthrough" bands from a non-resident dye to occur in the data image for another dye, as shown in Figures 4.2a and 4.2b. These non-resident bands can interfere destructively with the actual data bands, or masquerade as data bands of the resident dye;

- *Gel smiles.* Identical DNA fragments may migrate at varying rates along different lanes on the same gel, causing "gel smiling" patterns such as those depicted in Figures 4.3a and 4.3b. The presence of gel smiles impedes size calibration by linear interpolation across lanes;

- *Staggered loading.* It is a common practice to load DNA samples into the gel lanes in a staggered manner[5]. This loading practice creates highly complicated patterns such as those shown in Figures 4.4a, 4.4b and 4.4c. When coupled with the "gel smile" effect, the patterns can be visually indecipherable on a high throughput gel with a large number of gel lanes.

Relying on human technicians to manually track the data is at best a temporary measure. Recent advances in gel separation technology and demands for higher throughput will further increase the number of lanes that can be packed onto a single gel. Moreover, the need for sizing precision will lead to common usage of high resolution MW size standards, increasing the number of MW bands that has to be calibrated per lane (by hand). As manual data tracking becomes less feasible, the fully automated construction of gel sizing grids becomes increasingly essential.

[5]This is a practice carried over by technicians who are used to loading gels for *DNA sequencing* instead of fragment analysis such as microsatellite genotyping. Instead of using a "square-tooth" loading comb, a "shark-tooth" loading comb is used to load the DNA samples onto a gel. With this loading method, there are no pre-defined loading wells for the lanes on the gel. As such, the lanes must be loaded in *batches*. First, a subset of the lanes (e.g. the odd-numbered lanes) is loaded onto the gel with the shark-tooth comb. The gel is then allowed to run-out partially before loading the other set of lanes with the shark-tooth loading comb. This time delay results in a zig-zagging band patterns observed on the gel, which are useful for detecting spillovers from neighboring lanes.

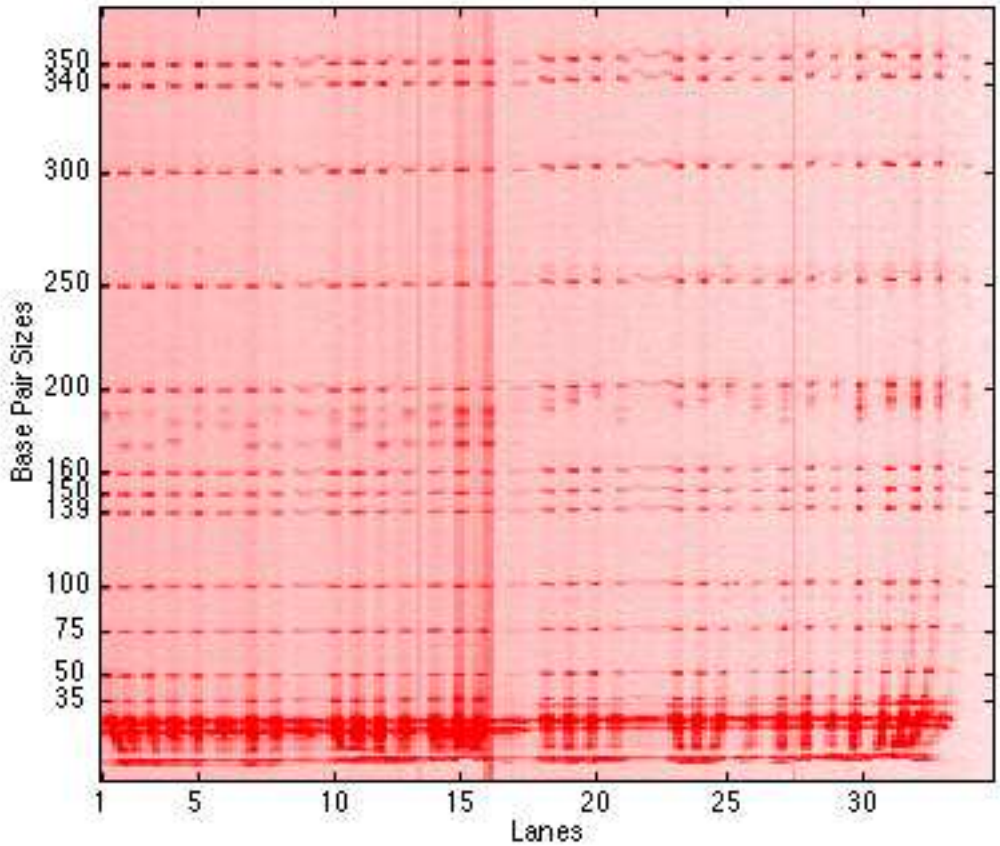


Figure 4.2a. Gel bleedthroughs. Shown here is the image of a 32-lane gel for the TAMRA dye. The rows of data bands are from the identical set of TAMRA-labeled MW sizes loaded in each of the 32 lanes. On closer inspection, other data bands can also be seen in the region between 160-200 bp and lanes 1-15. As magnified in Figure 4.2b, these bands are "bleedthrough bands" from the HEX dye, which labels a genetic marker with allele sizes 160-200 bp on this gel. *(Provided by Lillian M. Bloch, Cybergenetics, Inc.)*

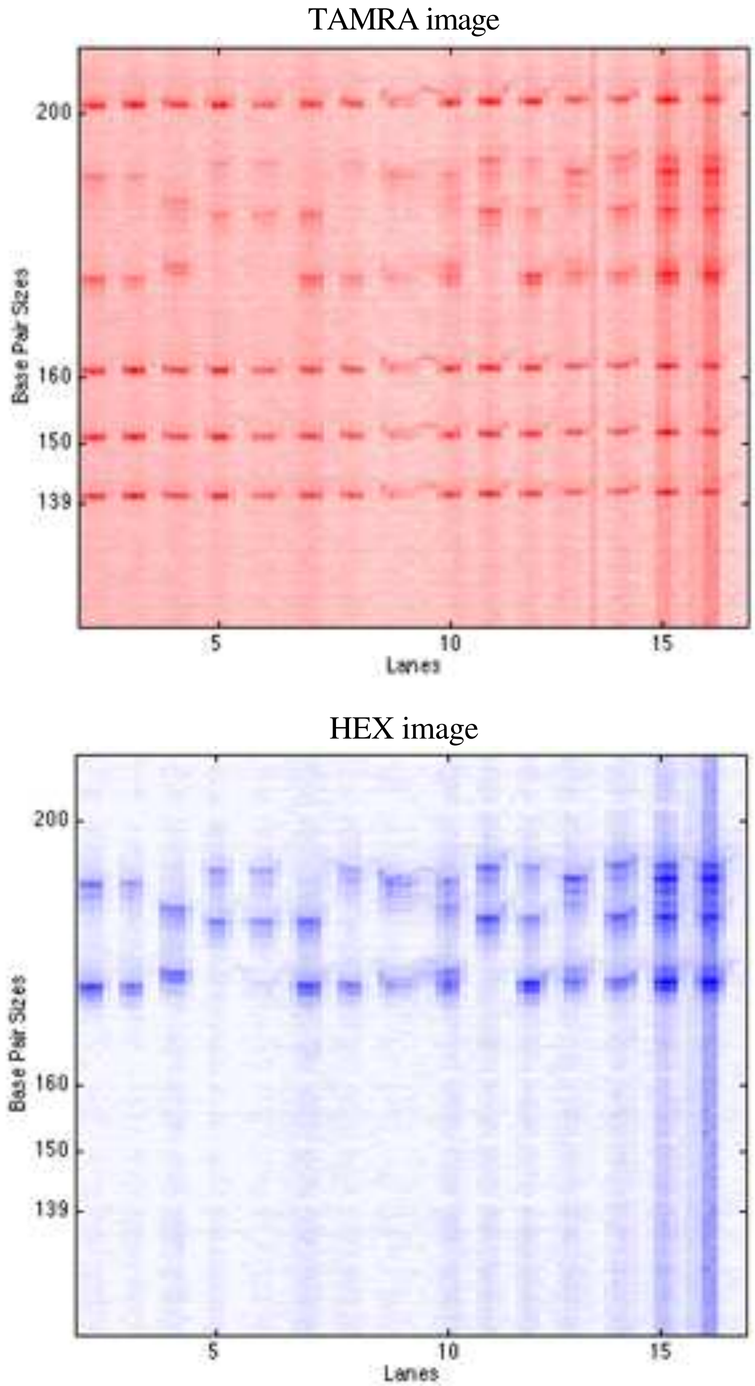


Figure 4.2b. We zoom in to the gel region between 160-200 bp, and lanes 1 through 15 for the gel shown in Figure 4.2a. On the top is the gel image for the TAMRA dye, below it is the image for the HEX dye for the same gel region. The data bands from the marker in the HEX dye have clearly bled into the TAMRA dye image.

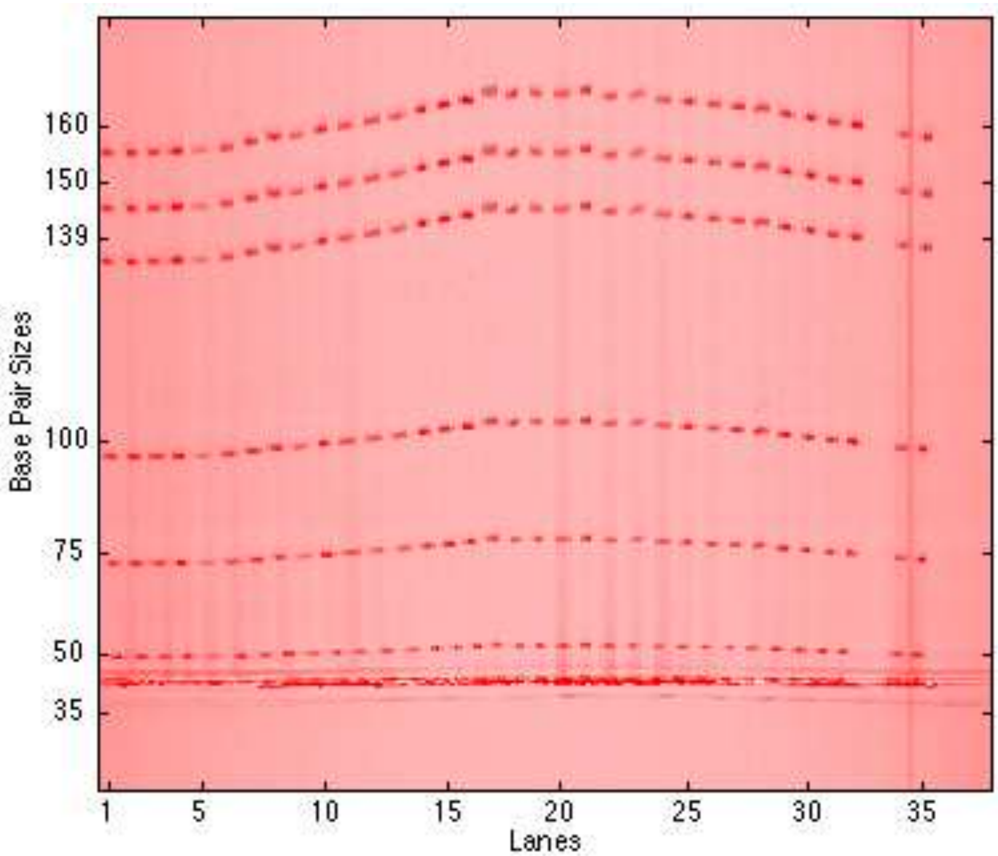


Figure 4.3a. Gel smiles. Instead of forming *straight* rows across the lanes, the identical sizing DNA fragments loaded in each of the 34 lanes on this gel migrated at different rates in different lanes, resulting in curved rows ("smiles") across the lanes. *(Provided by Dr. Charles Mein, University of Oxford.)*

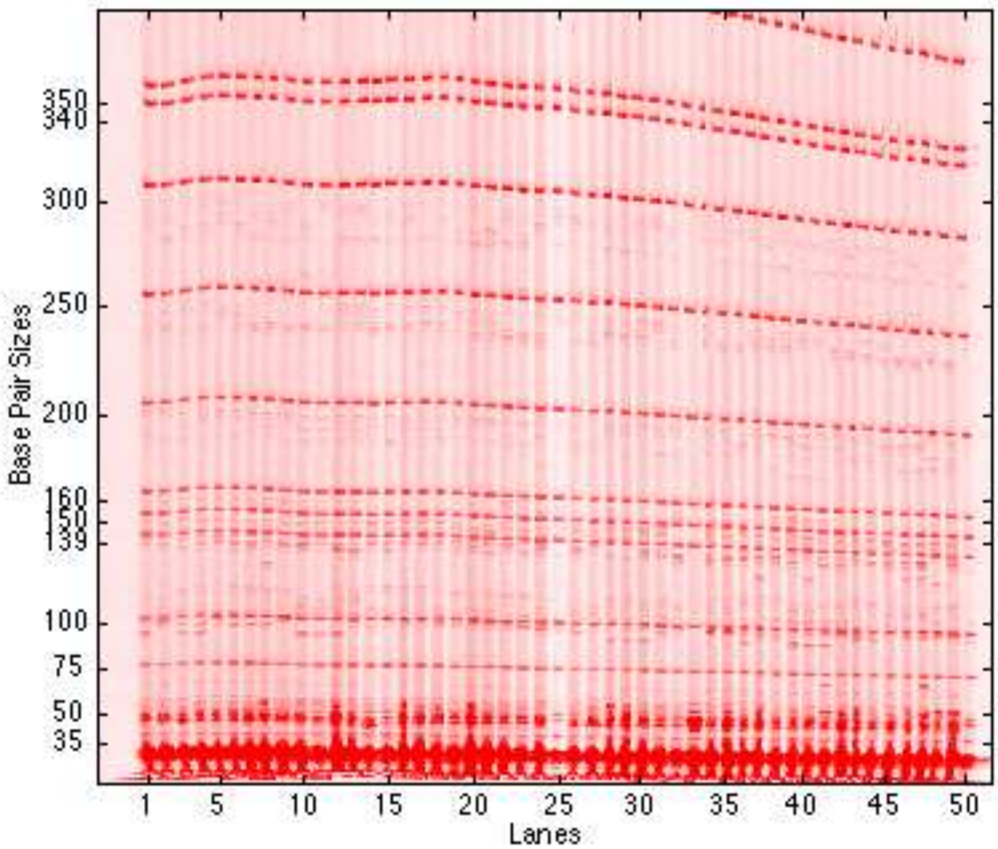


Figure 4.3b. Another example of gel smiles. This 50-lane gel exhibits a different "gel smile" pattern from the one in Figure 4.3a. *(Provided by Dr. Vicki Magnuson, National Institutes of Health.)*

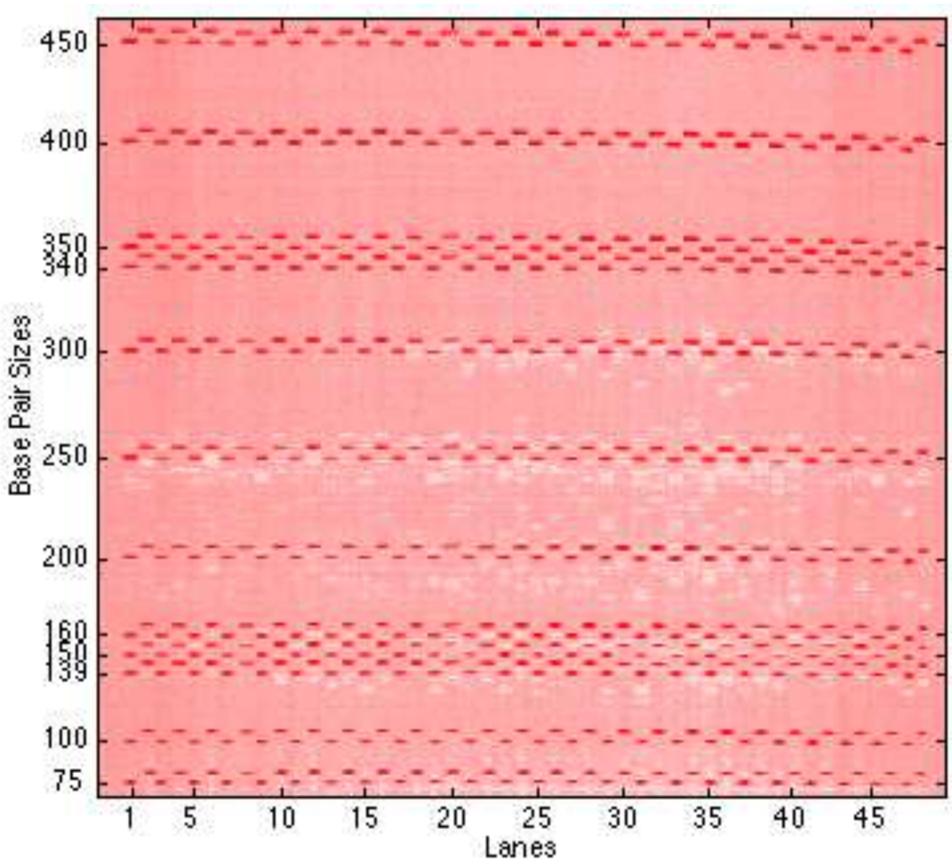


Figure 4.4a. Staggered loading. Instead of loading all the 48 lanes on this gel simultaneously, the odd numbered lanes were loaded first, followed by the even numbered lanes after a slight time delay, resulting in a zig-zagged pattern. *(Provided by Frosti Palsson, deCODE Genetics, Inc., Iceland.)*

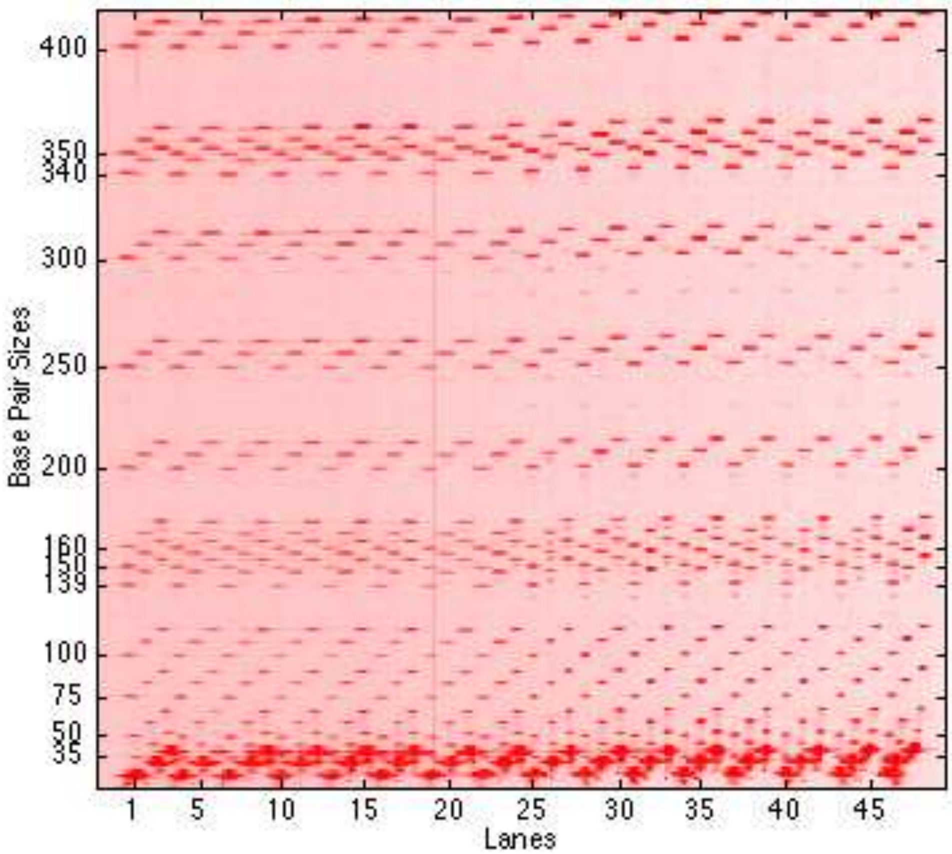


Figure 4.4b. Another example of staggered loading. On this 48-lane gels, sets of lanes were loaded at three different times. First, lanes 1, 4, 7, and so on were loaded. Then, lanes 2, 5, 8, and so on were loaded. Finally, the remaining lanes, lanes 3, 6, 9, and so on, were loaded. This resulted in a complex zig-zagging pattern that is very different from the one shown in Figure 4.3a. *(Provided by Frosti Palsson, deCODE Genetics, Inc., Inceland.)*

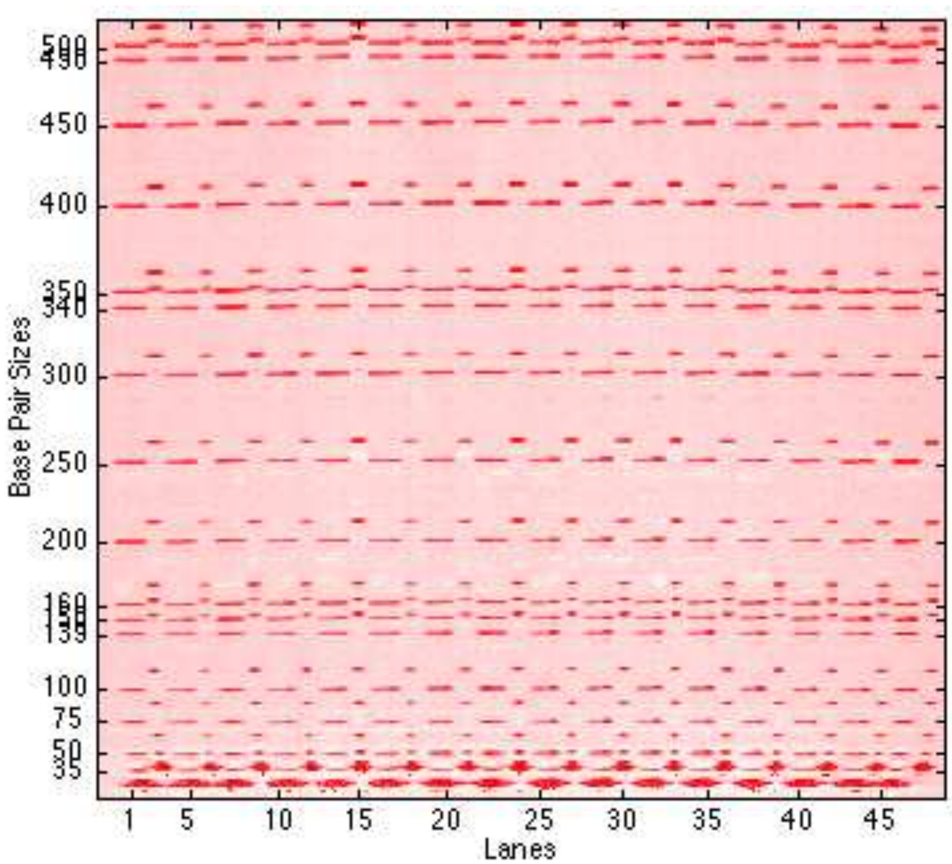


Figure 4.4c. Yet another example of staggered loading. In this 48-lane gel, lane numbers that are multiples of 3 were loaded first, followed by the remaining lanes. This resulted in another pattern that is visually interesting but difficult to track manually. *(Provided by Frosti Palsson, deCODE Genetics, Inc., Iceland.)*

4.1.1. Algorithm

Because of the gel image size[6] and the various confounding factors such as dye bleedthroughs, gel smiles, and staggered loading, direct *two-dimensional* pattern matching is too computationally intensive for a practical genotyping system which the complex sizing grids must be constructed in minutes. To achieve the requisite speedup of our expectation-based methods, we simplify the problem using a *divide-and-conquer* strategy (Figure 4.5): first, we reduce the two-dimensional problem into multiple one-dimensional spaces to quickly generate a grid template; then, we locally refine the grid template in two-dimensional space. Assume that there are n gel lanes each loaded with a MW standard of m DNA sizing fragments.

<u>Step 1</u>: Lane tracking.

Instead of initially searching the entire gel image two-dimensionally for all the $m \times n$ MW bands, we first track the n vertical lanes. Note that gel lanes are reasonably straight , at least within a small gel section. We therefore divide the gel image into several horizontal subsets, as shown in the first pane of Figure 4.5. For each horizontal subset, we project the two dimensional subset region onto the x-axis to form a local "lane template". We then combine these partial lane templates to form a complete lane template for the entire gel.

<u>Step 2</u>: MW calibration.

With the n lanes tracked, we reduce the image into n one-dimensional intensity profiles or *electropherograms*, as shown in the second pane of Figure 4.5. We then search one-dimensionally along each of the electropherograms for the m MW bands. Once every lane has been processed, we have found the $m \times n$ MW bands that form the sizing grid α for the gel.

<u>Step 3</u>: 2D grid refinement.

Finally, as shown in the third pane of Figure 4.5, we perform *local* two-dimensional refinements on α to account for the two-dimensional dependencies lost during the divide-and-conquer problem reduction.

---

[6]Currently, a gel image is about 15Mb. The need for high precision and high throughput will only continue to increase the gel size further.

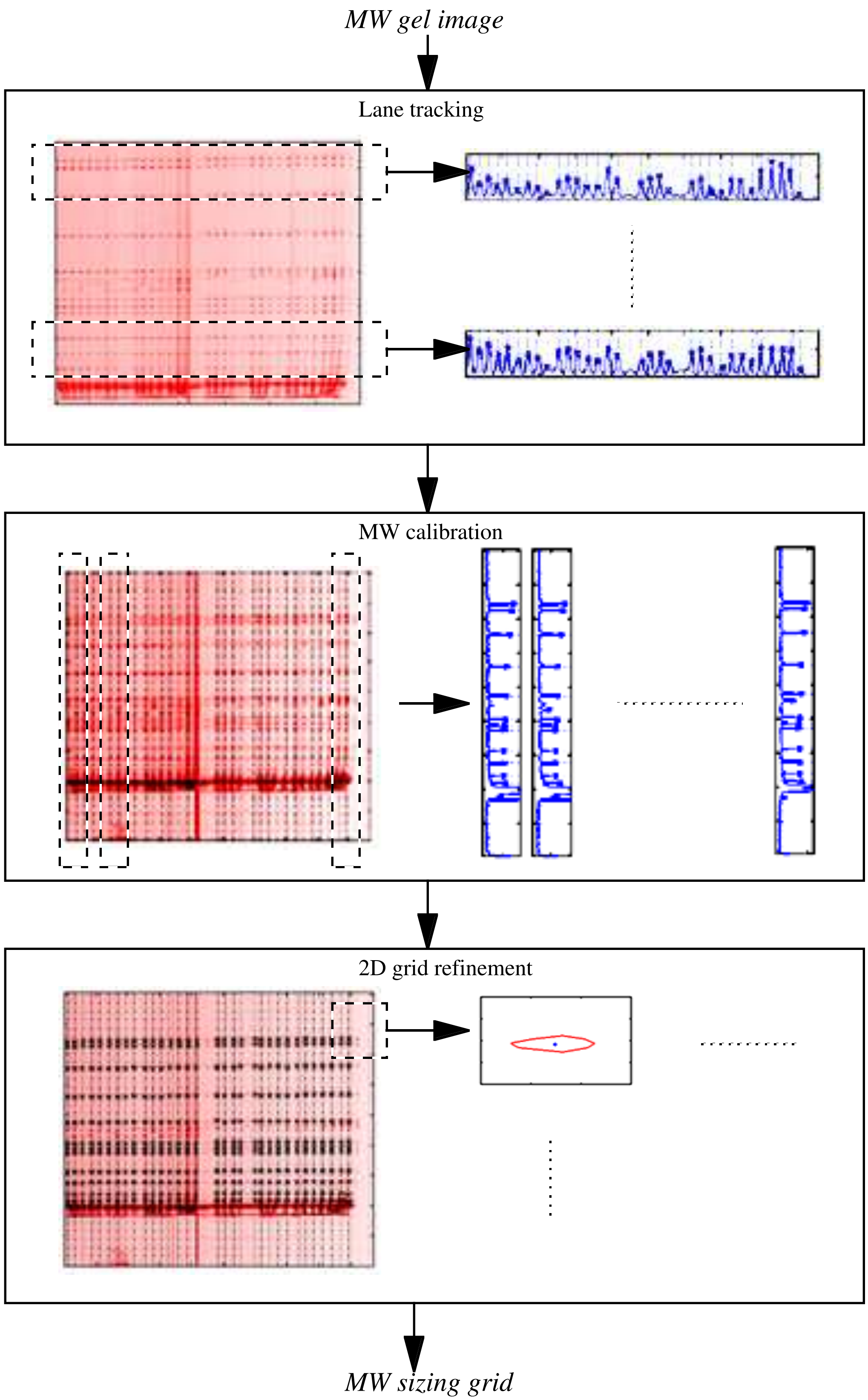


Figure 4.5. Sizing grid construction. A three-step divide-and-conquer approach for constructing the sizing grid for a two-dimensional gel image.

Algorithm CONSTRUCT_GRID

In this algorithm, we assume that DNA migrates along the direction in the y-axis of the gel image.

Step 1: Lane tracking.

Construct, using algorithm TRACK_LANE, the mapping function

$$\alpha_{lane}: \langle x, y \rangle \text{ pixels} \rightarrow \text{lane}$$

This transformation reduces the vast 2D gel image into 1D lane electropherograms to facilitate separate MW calibration on each lane.

Step 2: MW calibration.

Construct, using algorithm CALIBRATE_MW, the mapping function

$$\alpha_{size}: \langle \text{lane}, y \rangle \rightarrow \text{size}$$

This function maps the y-coordinates of each gel lane from the image pixel units to the molecular base pair (bp) units.

Step 3: Grid refinement.

Construct, from the composition of the two mapping functions, an initial band localization grid:

$$\alpha = \alpha_{size} \,.\, \alpha_{lane} : \langle x, y \rangle \text{ pixels} \rightarrow \langle \text{lane, size} \rangle$$

Refine, using algorithm 2D_REFINE_GRID, each expected MW grid point locally to adjust for any unaccounted two-dimensional shifts.

Box 4.1. CONSTRUCT_GRID: a divide-and-conquer algorithm for constructing a two-dimensional band-localization grid. We track the lanes first so that we can reduce the vast gel image into one-dimensional lane intensity profiles (electropherograms), on which we search for the MW bands one-dimensionally. To account for the two-dimensional irregularities resulting from the problem space reduction, we post-process the sizing grid by refining each grid point locally in two-dimensions.

Box 4.1 shows the overall algorithm for automatic construction of the sizing grid α on a dye-separated gel image containing MW data. The component algorithms for dye separation, lane tracking, MW calibration, and grid refinement will be described separately in the ensuing sections.

4.2. Dye separation

By tagging DNA samples with primers labeled with different fluorescent dyes for separate signal detection, multiple DNA samples can share *both* the same lane and the same size window on the same gel. Using dyes that have non-overlapping fluorescent spectrums, we can detect the signals from the DNA samples labeled with different dyes separately using multiple laser-induced fluorescence sensors.

The fluorescent spectral ranges of the dye labels may not be perfectly non-overlapping in reality. As a result, dye signals occurring in the overlapping range may also be picked up by a sensor for detecting data signals from another dye. As a result, the signals from one dye can show up in the image for a different dye (as shown in Figure 4.2a and 4.2b). Such signal cross-talk is known as *dye bleedthrough.* A bleedthrough band may be indistinguishable from a data band of the resident dye, especially when there is a high concentration of data in the gel.

Most DNA sequencers provide a multi-component correction filter, usually in the form of a "dye separation matrix", that can be applied to the raw data to minimize the bleedthrough effects. For best results, the dye separation matrix must be frequently calibrated on a DNA sequencer to *experimentally* minimize the bleedthrough effect observed on gels generated from that machine. This is the recommended approach for minimizing dye bleedthrough. However, as a fallback, we will also describe an algorithm to *computationally* refine a dye matrix that has not been perfected experimentally.

4.2.1. Algorithm

Algorithm MINIMIZE_BLEEDTHROUGH shown in Box 4.2 describes one approach for computationally refining a dye correction filter. The algorithm exploits the expected location of disjoint size data for overlapping spectral dyes. In essence, the algorithm maximizes, at every image pixel, the signal from one dye while minimizing the signals from the other dyes. As such, the algorithm works best with data that contain mutually exclusive regions of signals from different dyes. On a high throughput gel where there is a high concentration of data from all the dyes, regions with mutual dye exclusion may be hard to find. In this case, we refine the dye separation matrix using separate calibration lanes, such as a gel with lanes loaded with only samples from a single dye to ensure maximal mutual dye exclusion, or a gel with one or more lanes loaded with DNA of known genotypes for all the dyes, and then using the algorithm at selected pixels of known dye exclusions.

Algorithm MINIMIZE_BLEEDTHROUGH

Assume an *n*-dye system. Let Ψ denote a given bleedthrough correction filter (e.g. a dye separation correction matrix) to be applied to the dye planes $P_1, P_2, \ldots, P_n$. The goal is to refine the filter Ψ to maximize, at each image pixel, the signal for one dye while minimizing the interference of the other dyes.

Let $P_{\Psi,d}(i)$ denote the resulting signal intensity at dye *d*'s *i*-th image pixel after applying the filter Ψ. We can solve the bleedthrough minimization problem by iteratively refining Ψ until we find:

$$\max_{\Psi}\left(\sum_i\left(P_{\Psi,D(i)}(i)-\sum_{d\neq D(i)}P_{\Psi,d}(i)\right)\right),$$

where $D(i)$ is the dye plane with the maximum value at pixel *i;* in other words, $P_{D(i)}(i)=\max_d P_d(i)$

Box 4.2. MINIMIZE_BLEEDTHROUGH: an algorithm for refining dye-separation to minimize bleedthroughs.

4.2.2. Example

A typical multi-component dye separation filter for an n-dye system is an $n \times n$ square matrix with each row adjusting for potential dye signal cross-talk for one of the n dye components. That is, each (i,j)-th element of the dye matrix defines the amount of fluorescent signal from the j-th non-resident dye that can be included in (or excluded from) the data for the resident dye i. If the fluorescence range of the dyes were perfectly non-overlapping, then the dye separation matrix is simply the $n \times n$ identity matrix. More typically, a dye matrix would look like D_{given}, an actual dye separation matrix on the ABI 377 sequencer (a 4-dye system with dyes FAM, TET, HEX, and TAMRA) that generated the gel shown in Figure 4.2a:

	FAM	TET	HEX	TAMRA
FAM	2.644	-2.809	1.387	-0.382
TET	-2.203	4.091	-2.369	0.654
HEX	-0.114	-1.084	1.945	-0.664
TAMRA	0.008	-0.055	-0.405	1.159

To filter the multi-channel raw data using this dye matrix, we align the raw data collected by the respective laser sensors into a row-matrix with one row for each of the dyes in the system. Let us call this data matrix *RAW_DATA*. To generate the dye-separated data, we simply multiply *RAW_DATA* with the separation matrix D_{given}:

$$SEPARATED\_DATA = D_{given} \times RAW\_DATA$$

The resulting gel image for the TAMRA dye after the application of D_{given} was shown in Figure 4.2a. For a closer inspection, we show the intensity profiles of the pre-dye separated raw data in Figure 4.6a, and the dye-separated profiles in Figure 4.6b. The high degree of bleedthrough in the raw data was greatly reduced after applying the D_{given} to the data. However, some dye bleedthrough can still be seen to remain in the data, as shown in Figure 4.6b (and Figure 4.2b).

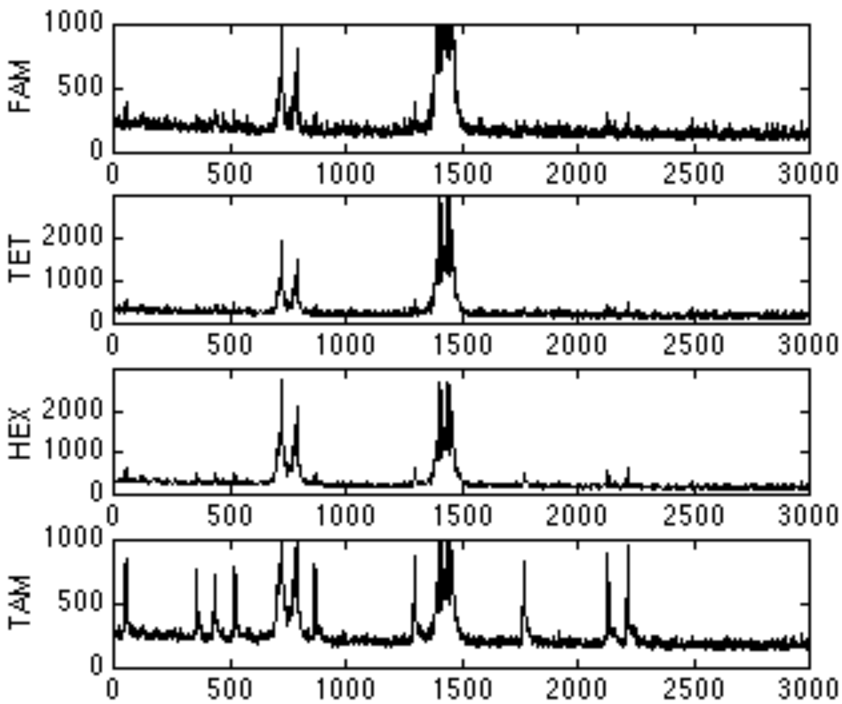


Figure 4.6a. The signal intensity profiles from the raw data from the laser sensors for each of the fluorescent dyes. Without applying the dye separation matrix, there is a high degree of dye bleedthrough. For example, the data signals between scan columns (x-axis) 500 and 1000, as well as those near scan column 1500, showed up in the intensity profiles of all the dyes.

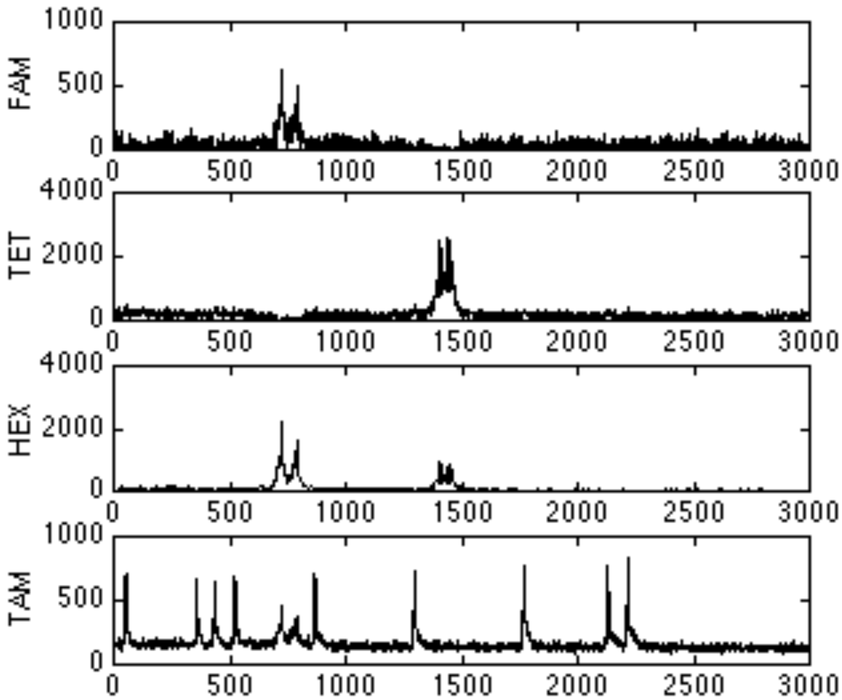


Figure 4.6b. After applying the dye separation matrix D_{given} , it became clear that the data bands between scan columns 500 and 1000 belonged to the HEX dye, whereas the bands around scan column 1500 belonged to the TET dye. However, the bleedthroughs have not been totally eliminated. For example, some of the data bands from the HEX dye can still be found in the data for the TAMRA (labeled as TAM here) and FAM dyes, and some of the bands from the TET dye found in the data for the HEX dye.

By using the strategy outlined in algorithm MINIMIZE_BLEEDTHROUGH, we computationally refine the dye matrix to maximize, at every image pixel, the signal from one dye while minimizing the cross-talk signals from the other dyes. The corrected dye matrix, $D_{refined}$, is shown below with the modified values italicized:

	FAM	TET	HEX	TAM
FAM	2.644	*-2.400*	*0.750*	*0.050*
TET	-2.203	4.091	*-2.300*	0.654
HEX	-0.114	*-1.450*	1.945	*-0.600*
TAM	0.008	*0.150*	*-0.650*	1.159

In Figure 4.6c, we show the signal profiles after applying $D_{refined}$. For comparison with the gel images in Figures 4.2a and 4.2b, we also show the resulting gel image for the TAMRA dye in Figure 4.7a, and magnified gel portions for TAMRA and HEX in Figure 4.7b.

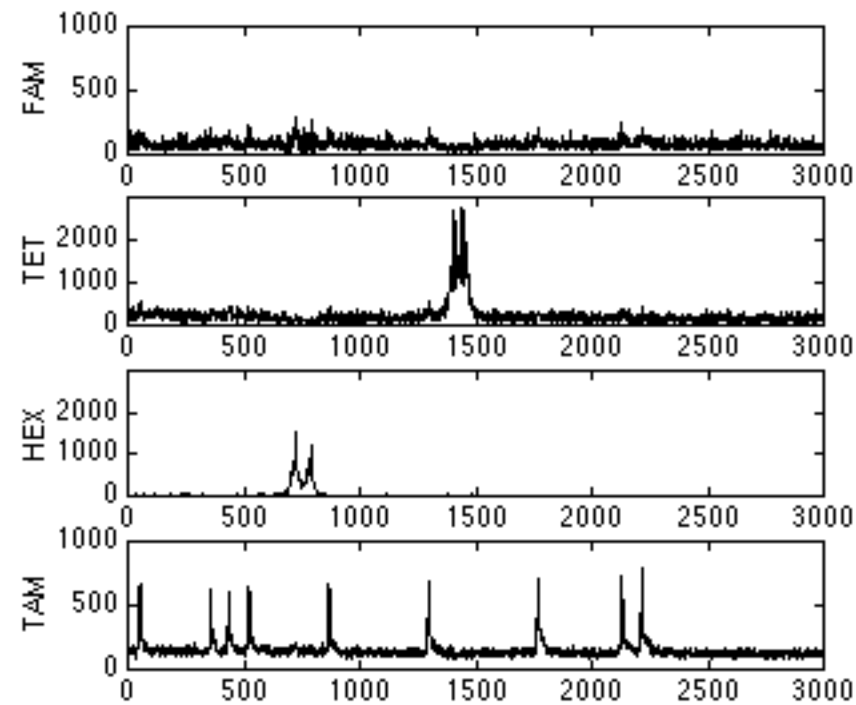


Figure 4.6c. The signal intensity profiles for the different dyes after applying the corrected dye separation matrix $D_{refined}$. The data bands from dye HEX no longer bled into the dye images for TAM and FAM, and the bleedthrough bands from the TET dye have also been eliminated from the data for the HEX dye.

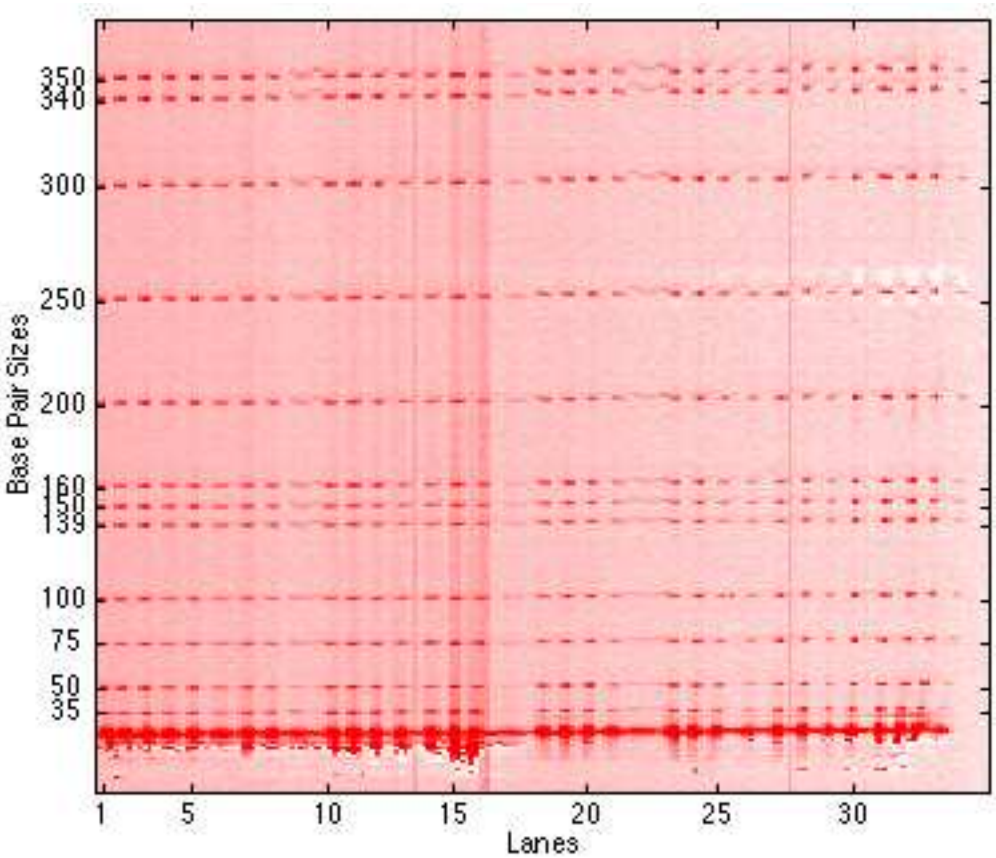
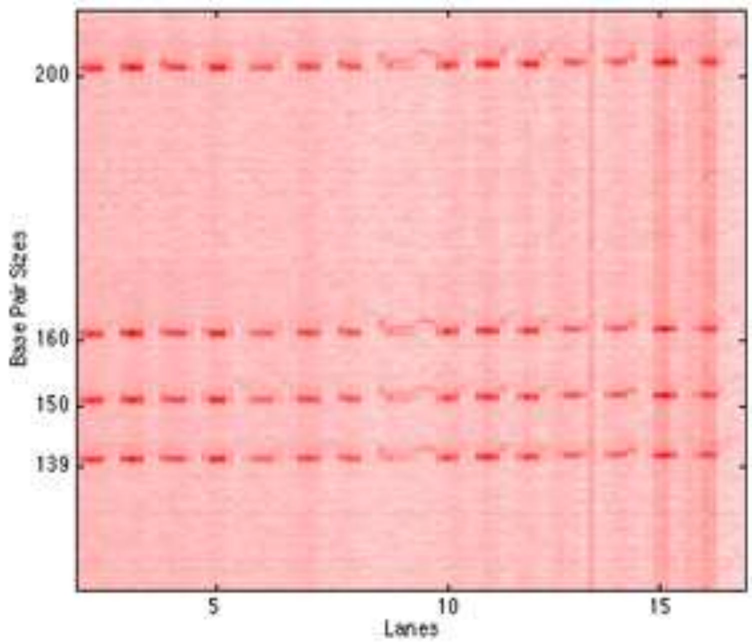


Figure 4.7a. The gel image for the TAMRA dye after the application of the refined dye matrix $D_{refined}$ on the raw data. See also the gel image shown in Figure 4.2a for comparison.

TAMRA image



HEX image

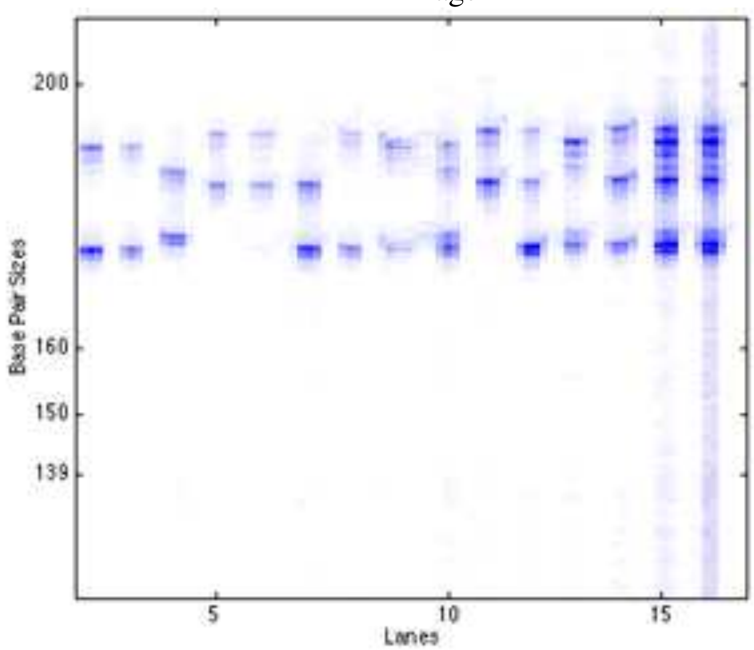


Figure 4.7b. Magnification of the gel region between lanes 1-15, and 139-200 bp. The data for the TAMRA (top) and HEX (bottom) dyes are shown. Unlike the similar images shown previously in Figure 4.2b, data bands from the HEX dye no longer bled into the data image for the TAMRA dye.

4.3. Lane tracking

In multiple-dye systems, data from the different dyes can be treated as if they were run in separate "dye planes" in parallel. In order to calibrate each gel lane internally with molecular weight (MW) size standards, one of the dye planes, the "MW plane", is typically devoted to running the size standards exclusively. In single-dye systems, it is usually not possible to co-electropherosize the size standards with the genetic markers in the same lanes. Instead, some of the gel lanes are devoted to running the MW size standards while other lanes run the genetic markers. For these systems, we can create a pseudo "MW plane" by filtering out the marker data. Thus, data on an electrophoretic gel can generally be classified into two broad categories: data from the genetic markers (the "marker planes"), and data from the MW size standards (the "MW plane").

Because the same size standards are run in every calibration lane on the MW plane, highly regular band patterns are expected to appear on the MW plane. In addition, we can predict the MW band patterns with great certainty since we know the exact molecular sizes of the DNA fragments on the MW plane. As compared with the marker planes where the band patterns are by no means regular or predictable, the MW plane is therefore the best candidate for tracking gel lanes.

4.3.1. Algorithm

Efficiency and robustness are two important criteria for a practical system which handles real data: efficiency can be accomplished by pruning the search space as much as possible, while robustness can be attained by pruning the search space *intelligently*. to avoid the pitfalls created by the various spurious artifacts in real data Computationally, the search space can be pruned by *problem reduction*, and the spurious data artifacts can be evaded using *expectation mapping*. The general framework, which you will see in most of the algorithms described in this dissertation, involves:

- *Problem reduction:* Simplify the problem as much as possible without losing too much information about the original problem;
- *Expectation construction:* Create expectations that are as specific as possible using all the information that we have about the domain, the problem, and the data;
- *Expectation application:* Apply the expectations by mapping them onto the observed patterns of the data; and

- *Iterative refinement:* Refine the solutions constructed from the reduced problem space in the original problem space to adjust for any artifacts that may have been lost in the process of problem reduction.

Let us apply this computational framework to solve the lane tracking problem. In the following discussion, we will use the convention that (a) the gel image (i.e. the MW plane) is a rectangular intensity matrix, (b) the gel lanes (or DNA migration) are vertical, and (c) the same MW standards are loaded in every lane.

<u>Step 1</u>: *Reduce problem.*
Since the lanes on a gel may not be perfectly straight throughout, we divide the gel image into numerous horizontal sections that are small enough so that the segments of lanes can be correctly assumed to be straight within the sections. We can then exploit the lane "straightness" characteristic in each gel section to reduce them into cross-sectional profiles formed by vertically projecting the image pixel with the highest intensity in each column onto the horizontal axis. In this way, we reduce the problem of tracking lanes in a vast two dimensional image into a much simpler problem of identifying peaks along a set of one-dimensional projection profiles.

<u>Step 2</u>: *Build expectation.*
Using our knowledge about the gel's layout and assuming approximately equal lane widths, we build a preliminary expectation of where the lane peaks might lie in each of the projection profiles. We then refine this expectation further by making it conform locally to the actual detected peaks in the best projection profile in the gel.

<u>Step 3</u>: *Apply expectation.*
Using the localized expectation from the best projection profile, we map the detected peaks in the other projection profiles. A good strategy which exploits the gel continuity is to start mapping from the neighboring gel sections from the best projection profile. When all the cross sections have been processed, we join the mapped lane peaks in the each projection profiles to form a piece-wise constant lane template for the entire gel.

<u>Step 4</u>: *Iteratively refine.*
If necessary, we can repeat steps 2 and 3 using different initial projection profiles to see if we can improve on the lane template that we have constructed so far. We will postpone the

adjustment for two dimensional dependencies (using algorithm 2D_REFINE_GRID) until after we have detected the MW bands.

Box 4.3 presents the details of our lane tracking algorithm.

Algorithm: TRACK_LANE

Step 1: Reduce problem.

(a) Divide the gel into several (say, k) horizontal sections: $R_1, R_2, ..., R_k$ such that each R_i is wide enough to contain at least a complete row of MW bands.

(b) Reduce each R_i section into an intensity profile $p_i(x)$ by projecting every (x, y) pixel in the region onto the X-axis using:

$$p_i(x) = max\ R_i(x,y) \text{ for all } y \text{ in } R_i .$$

We use the "*max*" operator instead of the "*sum*" or the "*mean*" operator since it is more robust to slanted MW rows caused by gel smiles[7].

(c) Identify the peaks in each compressed profile $p_i(x)$. Each of these peaks indicates a potential MW-loaded lane. Let us assume that there are n loaded gel lanes.

Step 2: Build expectation.

(a) Identify a best horizontal section by selecting from the R_i 's, an R_{best} that maps with the initial lane expectation (based on gel layout information and assuming equal lane widths) best.

(b) Establish the best mapping between the peaks detected in $p_{best}(x)$ and the n MW-lanes $lane_1, lane_2, ..., lane_n$:

$$\beta_{best}: x \rightarrow lane$$

β_{best} forms the lane expectation that we can apply to the other gel sections.

[7]For example, if the MW rows were slanted from left to right, then a horizontal gel region may contain more MW bands on the left than on the right. By taking the maximum instead of the summation or the mean of each column, only *one* of the MW rows (the one with the maximum intensity) will be projected onto the summarized profile, regardless of how many extra MW rows that column happen to contain.

<u>Step 3</u>: Apply expectation.

Based on β_{best}, construct individual mappings for the flanking regions R_{best-1} and R_{best+1} which maps the peaks in the cross-sectional profiles to $lane_1$, $lane_2, ...,$ $lane_n$:

$\beta_{best-1}: x \rightarrow lane$

$\beta_{best+1}\ x \rightarrow lane$

Locally propagate from R_{best} until all k β_i's are determined. Together, the cross-sectional lane mappings $<\beta_1, \beta_2 ..., \beta_k>$ form an approximate piece-wise constant lane template for the entire gel.

<u>Step 4</u>. Iteratively refine.

Repeat Step 2 and 3 using other R_i as R_{best} to see if $<\beta_1, \beta_2 ..., \beta_k>$ can be improved further. The mappings are then smoothed and interpolated to form the final lane template for the gel:

$$\alpha_{\text{lane}}: \text{<x, y> pixels} \rightarrow \text{lane}$$

Box 4.3. TRACK_LANE: an expected-based, divide-and-conquer algorithm for tracking gel lanes.

4.3.2. Example

As an example, we will track the gel lanes on the example "bleedthrough" gel from Figure 4.2a. First, we divide the gel into horizontal segments so that we can assume straightness of the lanes in each gel segments. Using our knowledge about the gel layout (there were 34 wells on the gel, 32 of which were loaded with DNA, skipping lanes 17 and 34), we create a lane peak expectation which we refine by mapping it with the detected peaks in the best bottom-up projection profile shown in Figure 4.8a. As shown in the figure, there were 32 peaks detected (marked "*") as lane candidates. However, lane 22 did not contain sufficient signal level to be detected initially, while a candidate peak was detected at the unloaded 34th lane. Using the top-down expectation created from gel layout information, we robustly mapped the lane peaks (marked "o") by locating the missing 22nd peak and ignoring the extraneous 34th peak.

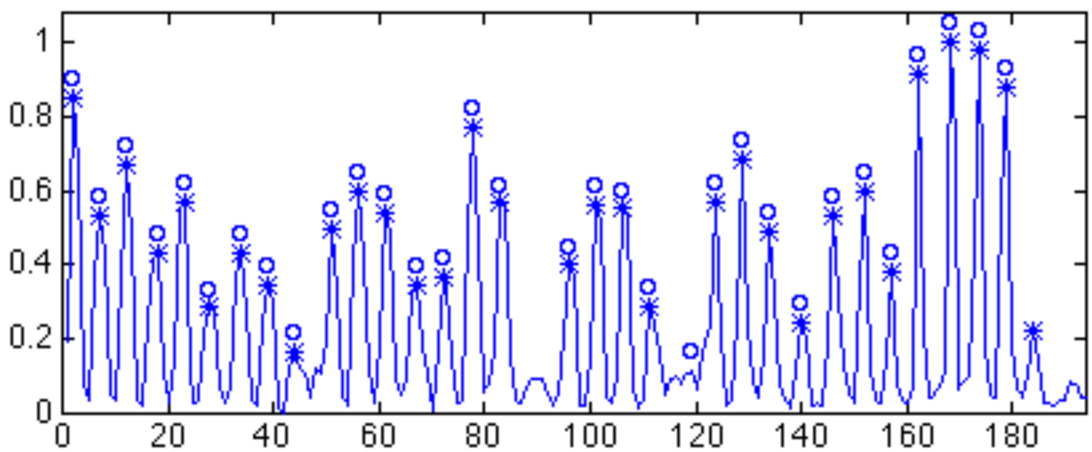


Figure 4.8a. The projection profile of the best horizontal subsection in the gel. The locally detected data peaks are marked "*", while the "o" markers indicate the final mapped lane locations using the gel's expectation layout information.

Using the refined lane template (marked "o" in Figure 4.8a) that we have created from the best cross-sectional projection profile, we proceed to map the lane peaks in the other cross-sections on the gel. Figure 4.8b shows the final mapped lane locations (depicted as dotted lines) of the first four cross-sections on the gel.

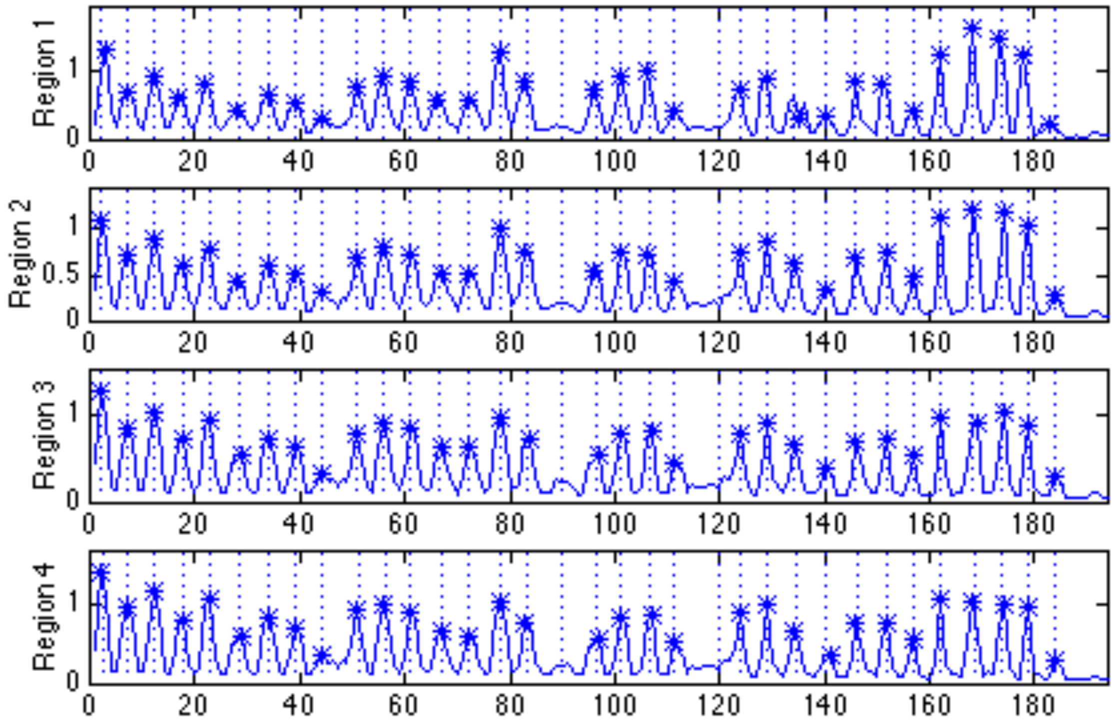


Figure 4.8b. The mapped lane locations (for all 34 lanes, including the unloaded lanes 17 and 34) for the various gel regions. The lane locations are depicted as vertical dotted lines, while the "*"'s mark the lane peaks that were detected initially on the projection profiles.

By joining the mapped lane locations from the different cross sections together, we construct a piece-wise constant lane template for the entire gel. The final lane template is shown in Figure 4.8c together with the MW plane, illustrating that the lane template was fairly accurate even though it was constructed as a two-dimensional concatenation of one-dimensionally cross-sectional projection profiles.

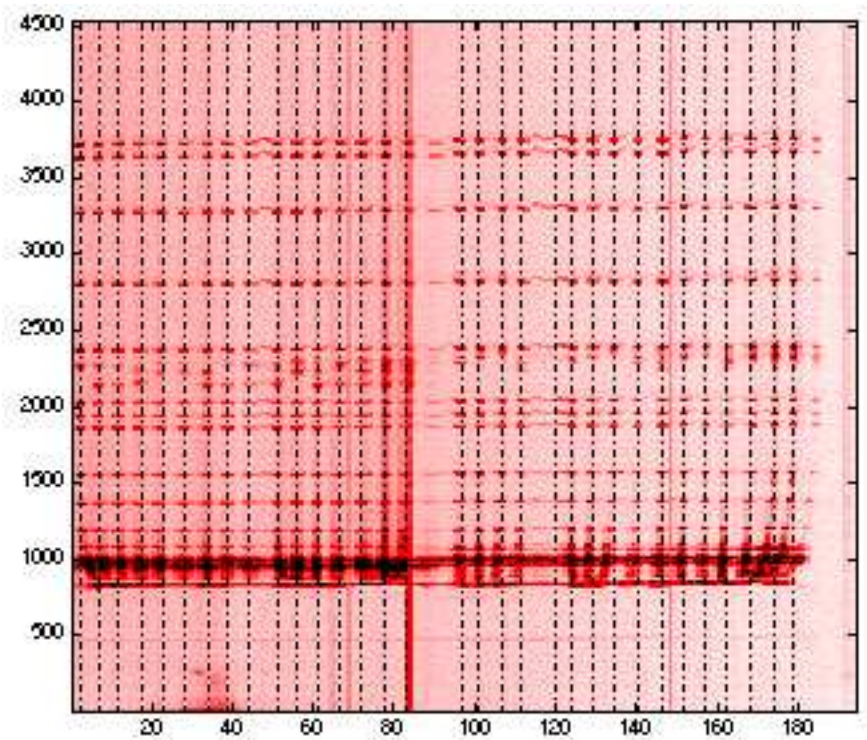


Figure 4.8c. The piece-wise constant lane template (dashed lines) constructed from the lane locations from the various cross sections on the gel. Only the tracks for the 32 loaded lanes are shown over the gel image.

4.4. MW calibration

Once the lanes have been tracked, the two-dimensional gel image can be reduced immediately to a series of one-dimensional vertical electropherograms which can then be calibrated separately. In each lane, we detect the MW peaks and label them with the correct molecular sizes. To filter out extraneous peaks or infer any missing ones, we construct expectations to accurately predict where the MW bands are on the electropherograms.

To predict the expected positions of the MW peaks, the conventional approach is to assume a fixed mobility function such as the local Southern function (Ghosh *et al.*, 1997; Southern, 1979). Here, we adopt a *data-driven* approach by using actual relative pixels of MW peaks from previous gels to predict where the MW peaks would fall on the current gel. The relative pixels record the relative distances of the MW bands. Using the relative pixels of MW peaks learned from previous gels running the same MW standards, we can easily construct a highly reliable MW peak expectation without globally assuming any mobility function.

4.4.1. Algorithm

Lane tracking and MW calibration are actually a pair of *symmetrical* problems: the former involves the mapping of detected peaks horizontally ("row"-wise) to infer vertical points, while the latter involves mapping the peaks vertically (lane-wise) to infer horizontal patterns. As such, we apply the same computational framework:

(a) *Reduce problem.* The two-dimensional complexity of the problem is drastically reduced, since the lane template from TRACK_LANE is used to extract lane electropherograms for one-dimensional MW calibration;

(b) *Build expectation.* We apply relative pixel information that we have learned from previous gels, and customize it with the data on the best lane on the current gel;

(c) *Apply expectation.* We apply the MW peak expectation to search for MW bands in the neighboring lanes. To exploit lane to lane continuity, we adopt the strategy of searching for the MW peaks in the neighboring lanes first, incrementally refining the MW expectation as we propagate away from the initial lanes;

(d) *Iteratively refine.* We iteratively refine the solution by repeating the process using some other lanes to build the initial MW peak expectation in (b).

Box 4.4 gives the detailed description our MW calibration algorithm.

Algorithm: CALIBRATE_MW

Step 1: Build expectation.

(a) Select, from $lane_1$, $lane_2, ..., lane_n,$ a $lane_{best}$ that has the cleanest electropherogram $p_{best}(y)$.

(b) Based on the relative pixels we have learned from previous gels, establish the best mapping between the peaks in $p_{best}(y)$ and the known MW sizes:

$$\varphi_{best}: y \to size$$

Step 2: Apply expectation.

Base on the locally refined relative pixels in φ_{best}, construct individual mappings for the flanking lanes of $lane_{best},$ namely $lane_{best-1}$ and $lane_{best+1}$:

$$\varphi_{best+1}: y \to size$$

$$\varphi_{best-1}: y \to size$$

Repeat Step 2 for the flanking lanes of $lane_{best-1}$ and $lane_{best+1}$, until the MW peaks of all the lanes in the gel are detected and mapped.

Step 3. Iteratively refine.

Repeat Steps 1 and 2 using other $lane_i$'s as $lane_{best}$ to see if the quality of the MW mappings can be improved further. The best set of $<\varphi_{1,}\ \varphi_2 ...,\ \varphi_k>$ collectively forms the MW mapping for all the lanes in the gel:

$$\alpha_{bp}: \text{<lane, y>} \to \text{bp}$$

Box 4.4. CALIBRATE_MW: an expected-based algorithm for MW calibration.

4.4.2. Example

We continue with the "bleedthrough" gel (originally shown in Figure 4.2a) as an example. The MW size standards used on this gel was "GS350", which contains twelve standard DNA size fragments :

	MW standards											
bp	35	50	75	100	139	150	160	200	250	300	340	350

First, we construct a customized MW peak expectation from previous relative pixels for GS350 and the detected peaks on the best electropherogram, shown in Figure 4.9a (the smaller sizes are to the left of the electropherogram). The detected peaks are marked "*", and the mapped peaks are marked "o". Notice that there is a bleedthrough band between the size fragments for 150 bp and 200 bp. Since we are using the relative pixel information from previous gels running GS350, our algorithm is robust enough to avoid the bleedthrough band.

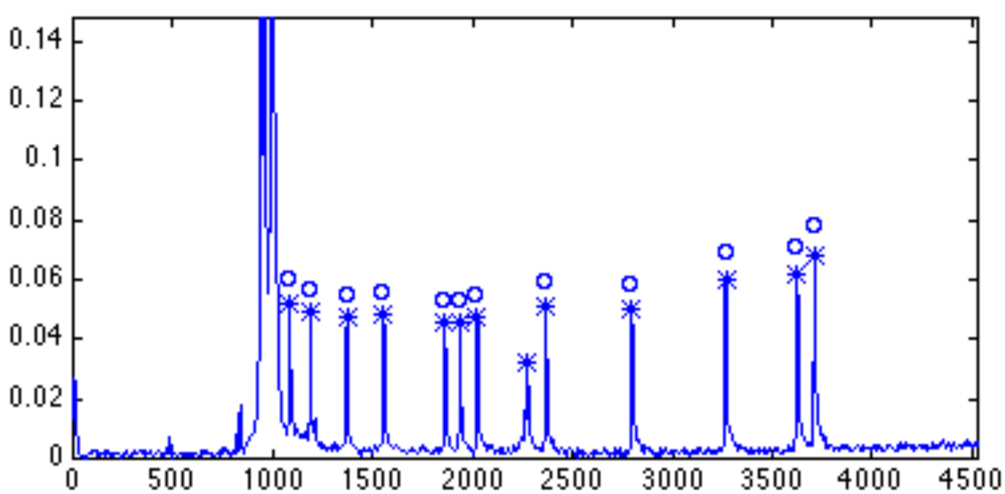


Figure 4.9a. The lane electropherogram for constructing the initial MW peak expectations. The detected peaks are marked "*", and the "o"'s indicate the mapped peaks. There is a bleedthrough band in the middle (the one marked with "*" but without an associated "o"), but the algorithm intelligently avoids this band using expectation information. The high intensity signals on the left are due to the excess primer bands from the markers.

The localized MW peak expectation enables us to robustly detect and label the MW bands on the other lane electropherograms. In Figure 4.9b, we show a lane electropherogram where the bleedthrough bands actually have higher intensities than the actual MW bands. Our expectation-based approach again intelligently avoids the pitfalls.

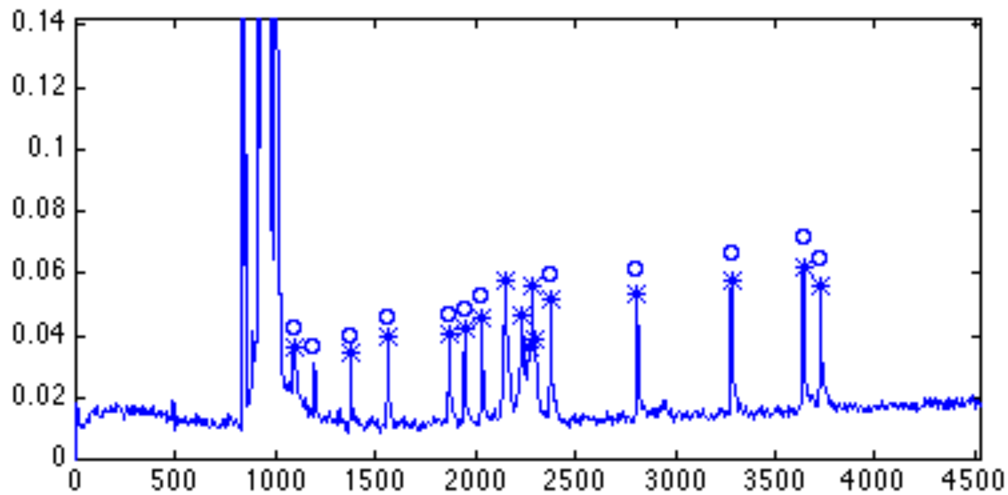


Figure 4.9b. An example in which the bleedthrough bands (in the middle region) are of higher signal intensities than the actual MW bands.

Figure 4.9c shows the final mapped MW peaks for the first 5 lanes, and Figure 4.9d depicts the mapped MW bands for all 32 lanes overlaid on the gel image.

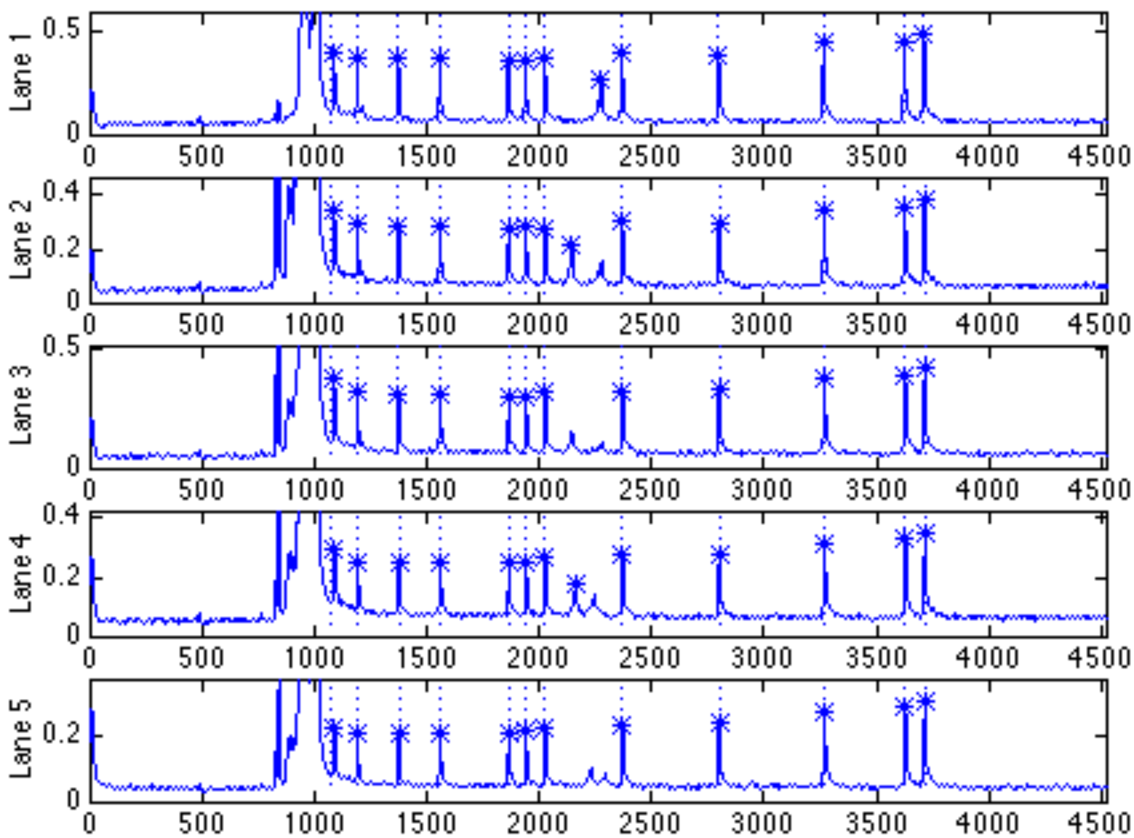


Figure 4.9c. The detected ("*"'s) and mapped (dashed lines) MW bands for the first 5 lanes of the gel. Notice how the bleedthrough bands have been avoided by the expectation-based approach.

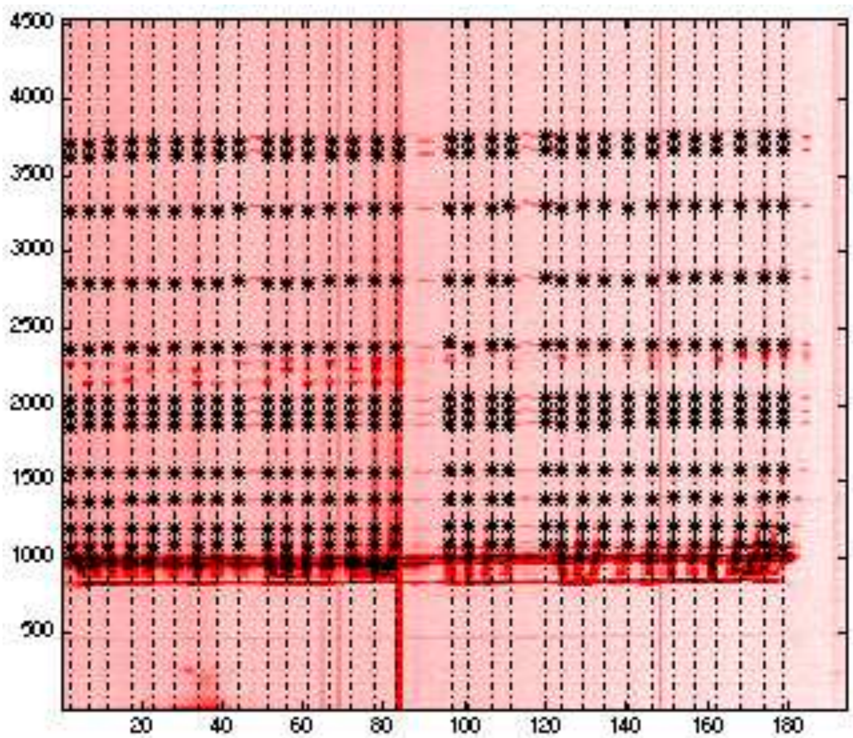


Figure 4.9d. MW calibration of the 32 loaded lanes. The dashed lines show the piece-wise constant lane templates constructed by TRACK_LANE. The "*"'s mark the mapped MW bands in each of the loaded lanes by CALIBRATE_MW.

4.5. Grid refinement

Tremendous speedup is obtained in reducing the two-dimensional grid construction problem into one-dimensional problems of lane tracking and MW calibration. The associated price for such problem simplification is that certain subtle two-dimensional dependencies may be lost in the reduced problem space. In particular, the MW bands are actually two-dimensional regions with finite widths and lengths, and not merely flat peaks along one-dimensional lane profiles. So, ideally, the grid points on the sizing grid should be the *centroids* of the two-dimensionally shaped MW bands. When mapping the MW peaks along the lanes tracked one-dimensionally by TRACK_LANE, we may miss the actual centroids of some of the MW bands since they could lie slightly off the lane templates. To account for such two-dimensional singularities, we perform a final grid refinement by *locally* searching for the peaks' centroids at each grid point.

4.5.1. Algorithm

From the mappings that we have constructed in TRACK_LANE and CALIBRATE_MW, namely:

α_{lane}: <x, y> pixels $\rightarrow$ lane, and

α_{size}: <lane, y> $\rightarrow$ size

we obtain a basic grid (as explained in CONSTRUCT_GRID) using functional composition:

$\alpha_{init} = \alpha_{size} \cdot \alpha_{lane}$: <x, y> pixels $\rightarrow$ <lane, size>

This grid (α_{init}) forms the initial expectation of where the MW bands might lie two-dimensionally. As we have seen, α_{init} is fairly accurate, so we only need to search *locally* in the vicinity of each grid points in α_{init} for the peak centroids. Box 4.4 describes how we refine α_{init} two-dimensionally to form the final sizing grid.

Algorithm: 2D_REFINE_GRID

For each grid point $<x, y>$ in α_{init}, do:

Step 1: Estimate enclosing rectangle of MW peak.

Using α_{init}, we determine an approximate height and width of a rectangular region enclosing the MW peak. For example, we can begin by trying a rectangle region with a 1 bp height and a width that is 75% of the actual lane width.

Step 2: Build local contour.

In the estimated rectangular region, build the local contour for the median pixel intensity value by joining pixels with at least this intensity value together.

Step 3: Adjust enclosing rectangle.

If the contour line does not form an enclosing region, then expand the enclosing rectangle accordingly and repeat Step 2.

If there are more than one enclosed contour lines in the rectangular region, then contract the rectangle towards the contour region that was closest to the expected size of a MW band, and repeat Step 2.

We stop until a single enclosed local contour can be drawn inside the rectangular region.

Step 4: Find centroid.

Compute the centroid $<x', y'>$ of the enclosed contour. Replace the grid point $<x, y>$ in α_{init} with $<x', y'>$, and then proceed to refine the remaining grid points in α_{init}.

Box 4.5. 2D_REFINE_GRID: an algorithm for refining the band localization grid locally and two-dimensionally.

4.5.2. Example

As an example of an initial grid peak which deviates from the actual centroid, let us zoom in on the lower left grid corner (i.e., lane 1, 35 bp) of our example gel. Figure 4.10a shows the three-dimensional view of the gel region near the lower left grid corner. The grid point detected by the one-dimensional algorithms TRACK_LANE and CALIBRATE_MW (marked "*") was slightly off the center of the actual MW band.

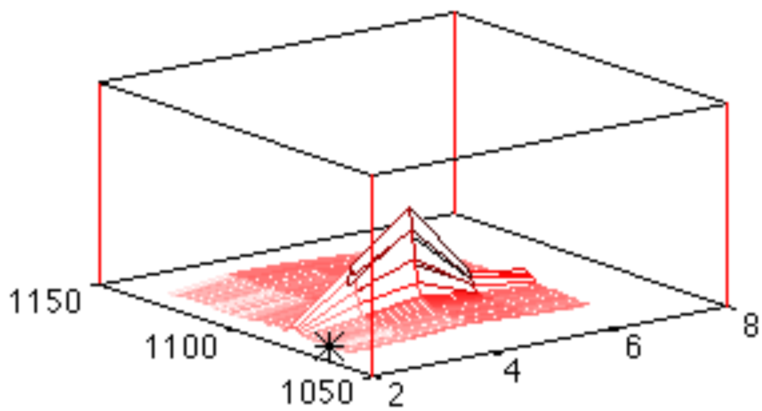


Figure 4.10a. A close-up three-dimensional contour view of the lower left grid corner of the example gel. The initial grid point is marked "*".

To refine the misplaced grid point, we define a bounding rectangle in its vicinity which includes a complete enclosed contour, as shown in Figure 4.10b. We then adjust the grid point by snapping it to the two-dimensional centroid of the enclosed contour (marked "+" in Figure 4.10b) in the bounding box. Figure 4.10c shows the relative positions of the original grid point (marked "*") and the two-dimensionally refined grid point (marked "+") in a three-dimensional contour view of the gel region.

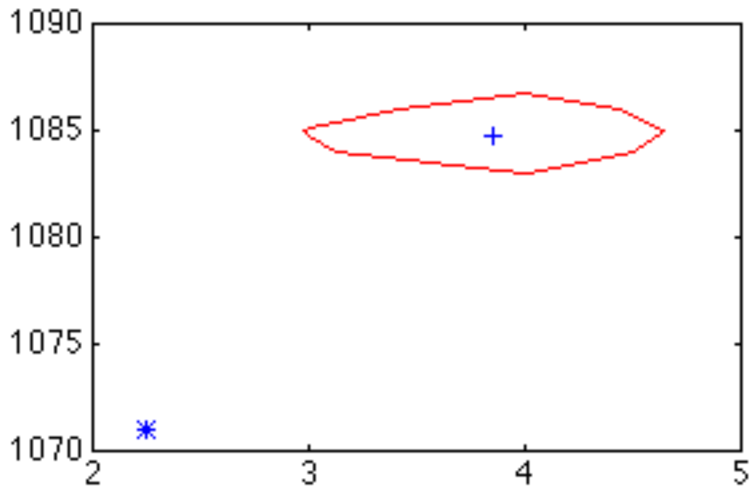


Figure 4.10b. Enclosed contour near the original grid point (marked "*"). The centroid is marked "+".

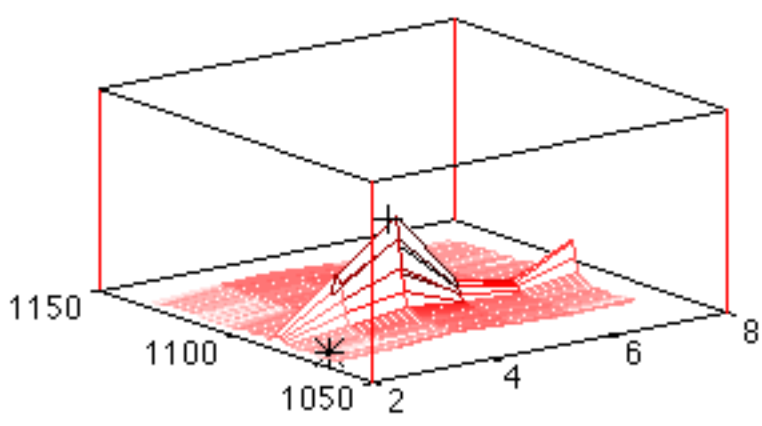


Figure 4.10c. Two dimensional refinement of the grid point (marked "*"). The final grid point is (marked "+") resides on the tip of the contoured MW peak.

4.6. Results

We have shown in great detail how our algorithms efficiently and robustly handled bleedthrough bands in our example gel. The computed grid for this gel is shown in Figure 4.11.

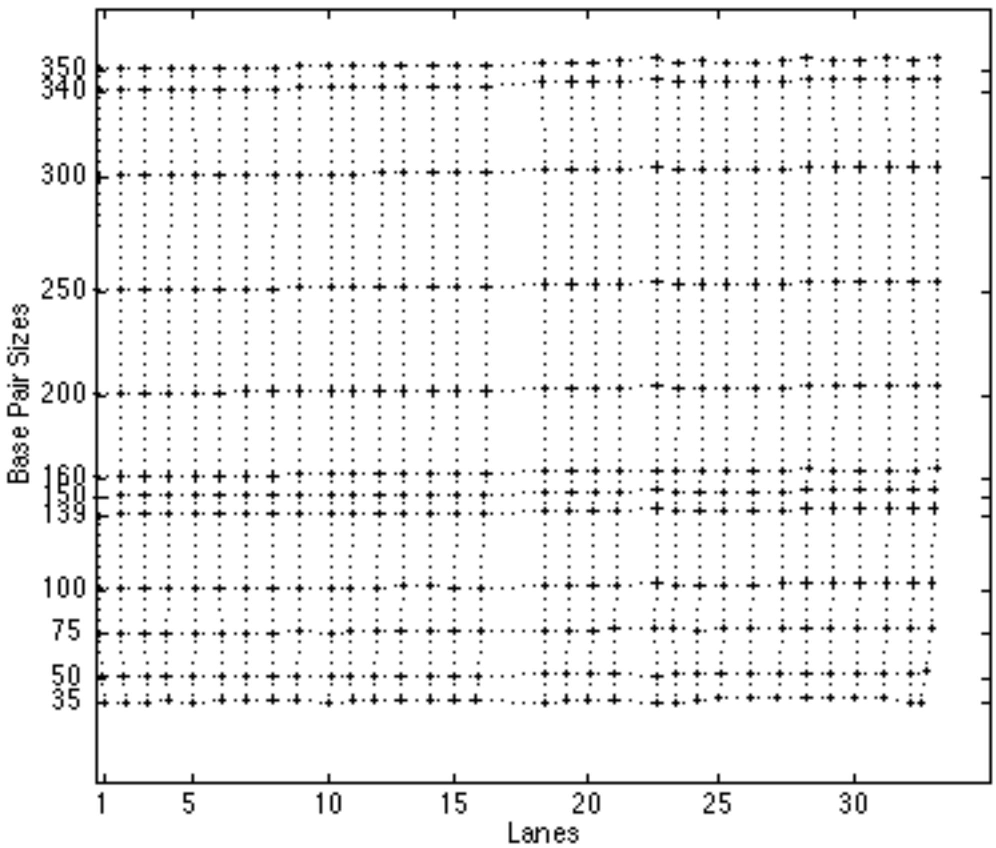


Figure 4.11. The computed sizing grid for the example "bleedthrough" gel originally shown in Figure 4.2a.

Our algorithms are also able to handle complex grid patterns caused by gel smiles (shown in Figures 4.3a and 4.3b) and staggered loading (shown in Figures 4.4a, 4.4b, and 4.4c). By tracking the lanes first, we can calibrate the MW bands separately on the gel lanes, handling the possibly drastic lane-to-lane shifts from gel smiles and staggered loading by not making any strict lane-to-lane continuity[8] assumptions during the independent MW calibration. Figures 4.12a and 4.12b show the computed sizing grids for the gels with gel smiles (shown originally in Figures 4.3a and 4.3b). The sizing grids for the staggered

[8]The only lane-to-lane continuity assumption that we exploit is that the *relative* pixels (and not the *absolute* pixels) of the MW bands are more similar for lanes that are closer together than lanes that are further apart on the same gel.

loaded gels shown in Figures 4.4a, 4.4b, and 4.4c are shown in Figures 4.13a, 4.13b, and 4.13c respectively.

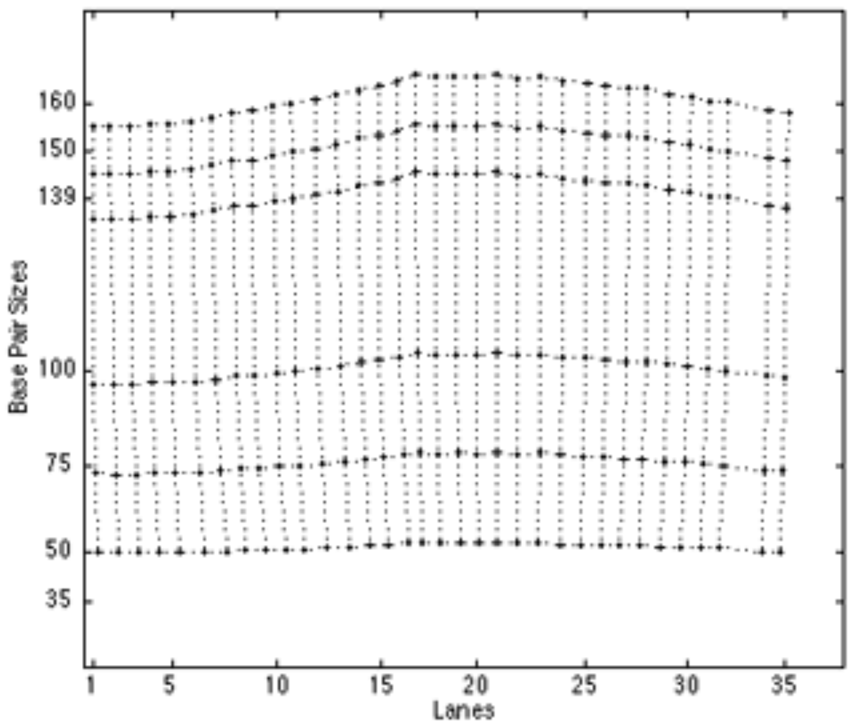


Figure 4.12a. The computed sizing grid for a gel showing "gel smile" artifacts. The original gel image was shown in Figure 4.3a.

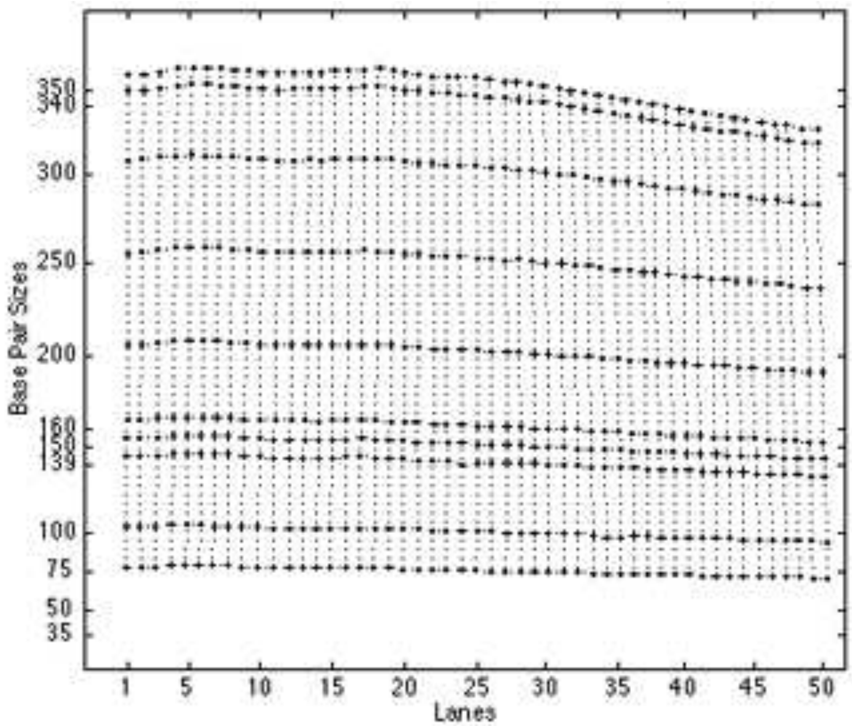


Figure 4.12b. The computed sizing grid for another gel showing a different "gel smile" artifact. The original gel image was shown in Figure 4.3b.

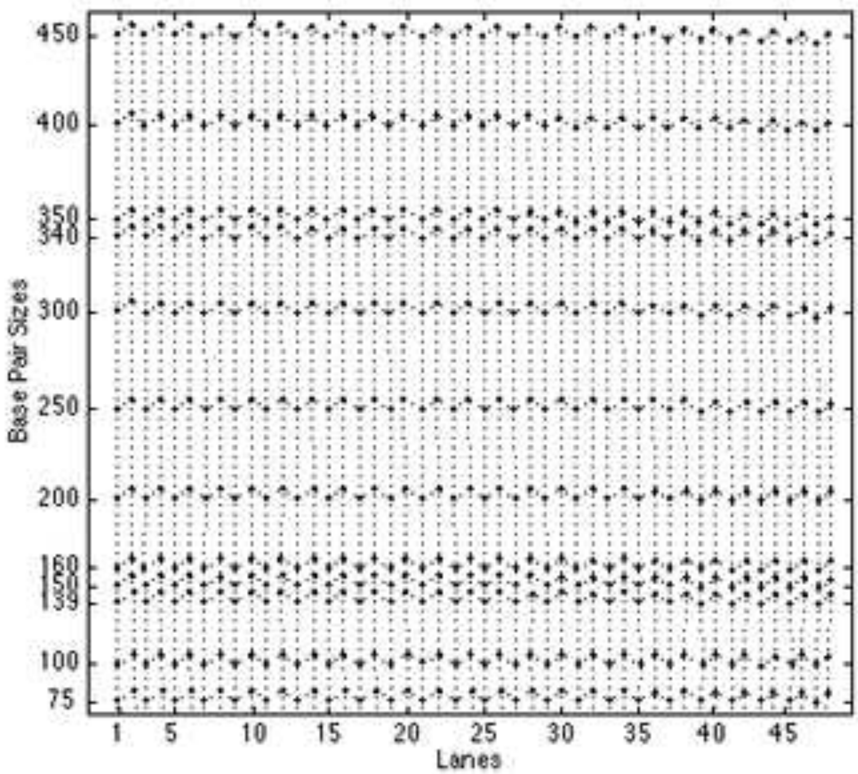


Figure 4.13a. The computed sizing grid for a gel that has a complex staggered loading pattern. The original gel image was shown in Figure 4.4a.

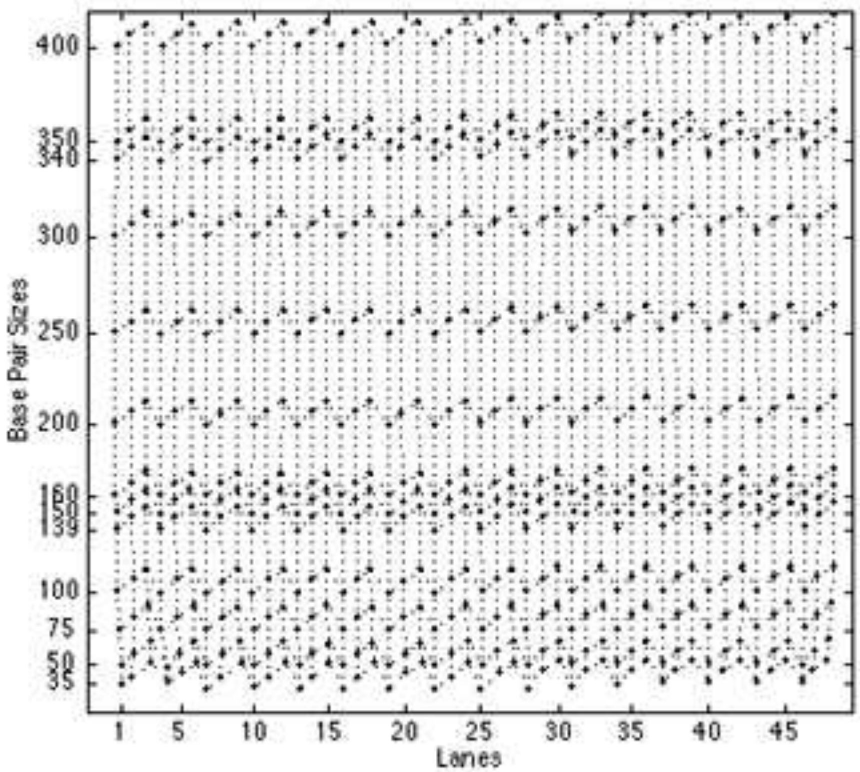


Figure 4.13b. The computed sizing grid for a gel that has a different staggered loading pattern. The original gel image was shown in Figure 4.4b.

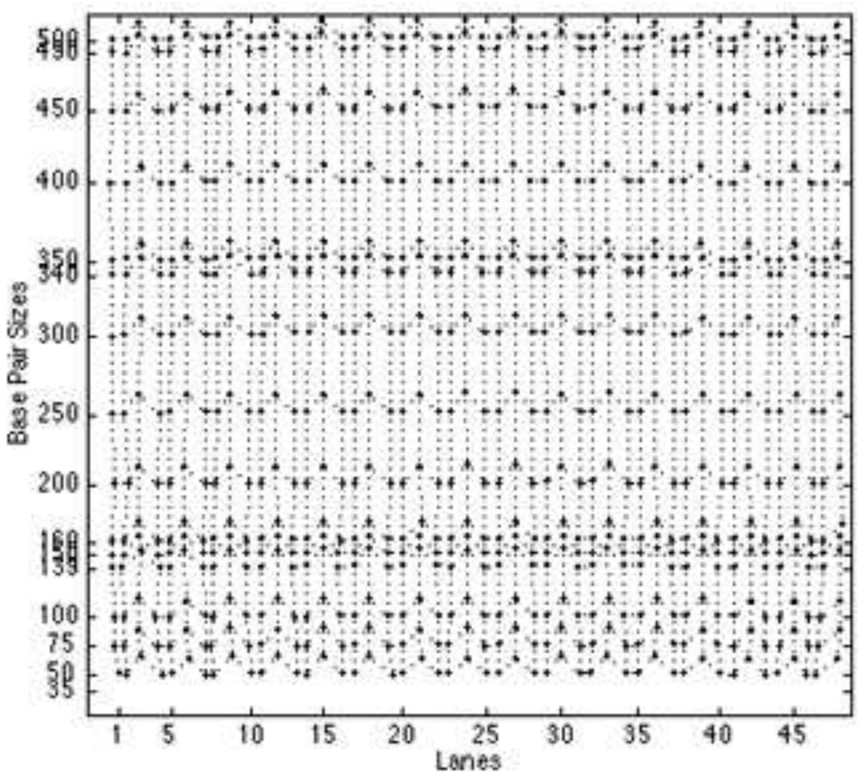


Figure 4.13c. The computed sizing grid for a gel that has yet another different staggered loading pattern. The original gel image was shown in Figure 4.4c.

4.7. Discussion

The grid construction algorithms presented in this chapter are computationally straightforward. Despite their simplicity, these algorithms were very effective in handling real and complex data, as we have seen from the examples. The algorithms were specifically designed to meet the various criteria for practical genotyping problem solving, such as:

- *versatility*: the algorithms must be able to handle vastly different loading patterns;
- *robustness*: the algorithms must not be easily distracted by noise and other pitfalls, such as extraneous or missing MW bands; and
- *efficiency*: the algorithms must be able to scan a gel and construct an accurate two-dimensional sizing grid in minutes.

In solving the practical problem of sizing grid construction, we have adopted various computational strategies. Although they are fundamental computational approaches, they have been critical in handling complex and noisy data in the case of sizing grid construction:

- <u>Divide-and-conquer</u>. Whenever possible, we reduce the problem into simpler ones. Simple problems are more likely to result in simple solutions that are easier to implement and maintain, and are therefore more robust in the practical sense. The reduction of problem also increases the degree of modularity in the system. Because we have separated lane tracking from MW calibration, the computer can handle both data that require lane tracking (e.g. ABI) and data that are already lane-tracked (e.g. Pharmacia), as it can optionally use the lane tracking module with the MW calibration module;

- <u>Apply expectations</u>. When constructing a sizing grid, we invest much effort in creating accurate localized expectations for guiding the search. As the computed expectations are used as initial solutions for the search, accurate expectations can drastically reduce the search required. With good data, the actual solution would be close to the expectation, and the computer will be able to find the solution quickly. With imperfect data, the actual solution deviates further from the expectation, thereby requiring more search effort from the computer to reach the solution. Thus, an associated advantage of expectation-based processing is that the computational requirement is directly proportional to the quality of the data. Additionally, the expectations provide a robust reference frame from which the computer can reliably assess the quality of the data, dynamically allocate its resource to regions of data that need more attention, intelligently attempt recoveries, and helpfully provide meaningful feedback to the user;

- <u>Refine using data</u>. Whenever we form an initial expectation based on some *a priori* assumption (e.g. equal lane widths in TRACK_LANE, initial relative pixels in CALIBRATE_MW), we always refine these expectations locally using the actual data as soon as we can. The data-corrected expectations are more accurate than expectations based on simplifying global assumptions;

- <u>Make few assumptions</u>. By making as few assumptions about the data as possible, we can efficiently handle unexpected characteristics in the data. For example, the practice of staggered loading, which resulted in complex zig-zagged patterns such as those shown in Figure 4.4a, b, and c, started only after we implemented the sizing grid construction module for our system. Because we did not make any *a priori* assumptions on strict lane-to-lane alignment, we were able to extend the MW calibration algorithm to handle the complex loading patterns;

- <u>Use heuristics</u>. In handling real data, the basic computational power of the algorithms must be complemented by heuristics which encode the immense obligatory *field* expertise. The heuristics handle the "details" that are necessary when working with *real* data in a practical system.

This assortment of basic computational strategies turns out to be fairly representative of the algorithms that we have developed for solving the other related problems in genotyping. As such, they will recur frequently in the next few chapters as we describe our other algorithms for solving the remaining problems.

5. Band Quantitation

The construction of the sizing grid allows us to reduce genotyping complexity in two ways. First, using the lane mapping in the grid, we reduce the multi-dimensional gel image into one-dimensional electropherograms; then, using the grid's MW calibration, we apply our knowledge about the markers' expected allele size range to isolate windows of marker data on the electropherograms for analysis.

In this chapter, we describe the next level of problem reduction. The PCR and gel electrophoresis process (Chapter 2) generates electrophoretic signals from size-separated classes of tagged DNA fragments. Each of these DNA fragments has an integral molecular size in base pairs (bp). Therefore, underlying the continuous intensity signals that we observe on the electropherograms are actually series of *discrete* data bands emitted by classes of DNA fragments with integral molecular sizes. The next natural problem reduction step is to *discretize* the continuous signals on the electropherograms into discrete data bands, each indexed with an integral allele size and quantitated with a relative DNA concentration measure. That is, we compute the following mapping function:

β : image data + sizing grid $\rightarrow$ {<bp, concentration>}

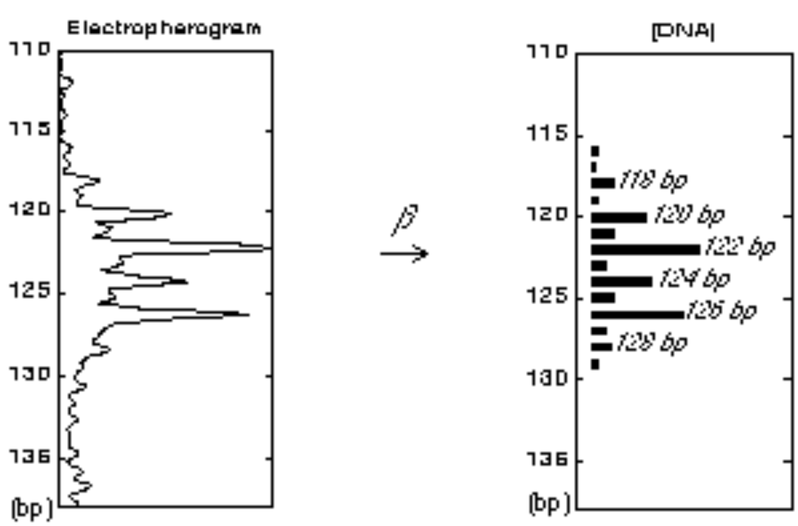


Figure 5.1. Band quantitation. This involves converting the continuous electrophoretic signals (shown on the left) into discrete data bands (shown on the right), assigning an integral size to each of the data bands detected, and determining the relative DNA concentrations of the data bands.

5.1. Problem

There are two main tasks in band quantitation:

1. binning: indexing each marker band with an integral molecular size, and
2. quantitation: determining each marker band's relative DNA concentration (defined as either the area under the peak in the intensity profile, or as the peak height).

For the binning task, the MW sizing grid can only give approximate sizes for the marker bands. There are two potential sources of binning errors:

- *Inadequate resolution.* The commercially available MW size standards (Genescan 500, Bioventures Map) provide only 50 to 20 bp spacing resolution, whereas the data bands from the markers can be as close as 2 bp (for dinucleotide repeat markers) or 1 bp (for mononucleotide repeat markers, or markers exhibiting the "plus-A" artifacts) apart;
- *Inappropriate interpolation.* The sizing grid is constructed using molecular size standards DNA, whose DNA sequences may differ chemically from the marker DNA[9]. As a result, the molecular weight spacing of the marker DNA fragment may not equal the spacing of the size standards. Thus, interpolating within the size standards may lead to inaccurate of the marker fragments.

For the quantitation task, since the fluorescent signal intensity is directly related to the amount of tagged DNA present, we can determine the relative DNA concentrations of the marker bands based on their intensities in the electrophoretic profiles. In a perfect system, the signal for each allele size present would be detected as an unambiguous spike, located exactly at the integral molecular size with a height (intensity) that is directly proportional to the amount of DNA of the particular size. With real data, there are two main problems:

- *Band widths.* Instead of generating unambiguous singular spikes, the DNA fragments produce data bands with two-dimensional shapes that have significant widths. One immediate consequence is that it is difficult to determine the exact integral size for a data band, as its center can lie anywhere within the band width;
- *Band overlap.* Because of the band widths, neighboring marker bands may overlap into one another, making it difficult to determine the individual areas or heights of the bands independently. Band overlap is particularly problematic with markers having

---

[9]The fluorescent molecules attached to the MW DNA fragments are different from those attached to the sample DNA in a multi-dye system. More importantly, the marker DNA is most likely to be chemically different from the MW standards. For example, a repeat unit of a CA-repeat consists of a cytosine (C) and an adenine (A), whereas a corresponding unit in the MW may contain a different pair of molecules formed by any two of the nucleotides adenine, guanine (G), cytosine , and thymine (T).

small repeats (e.g. the dinucleotide repeat markers), since the marker bands tend to be close together.

5.2. Binning by stutter crawling

Traditionally, stutter bands have been considered as noise because they obscure the true allele bands. Geneticists have attempted to minimize PCR stutter in markers experimentally, or failing that, avoided using markers with stutter altogether. Surprisingly, *abundant* stutter bands can provide a perfect solution for the binning problem, since:

(1) stutter bands, which include the true allele bands, provide the precise resolution needed for binning alleles;

(2) there are no discrepancies in the sequence chemistry, since the stutter bands have the same chemical units as the true alleles.

We will show, by a novel technique called *stutter crawling*, how we can use the stutter data to self-calibrate the marker bands, solving two of the problems introduced in Chapter 1:

- Sizing precision: how to calibrate the marker data with the requisite resolution, even when using low resolution MW standards for size calibration;
- Binning consistency: how to ensure that the alleles are binned with consistent integral size labels, minimizing interpolation and rounding errors.

5.2.1. Algorithm

In the laboratory, one can create a high resolution allelic ladder for a marker. This is done by pooling together many DNA samples from a population and then size-separating the pooled mixture on a single gel lane. To ensure that the resulting allelic ladder has the resolution required for disambiguating any two marker alleles, it is best to pool together many different DNA samples – this will sample a diverse set of alleles. Individual DNA samples can then be size-separated in gel lanes adjacent to the allelic ladders. This procedure permits consistent and unambiguous allele binning[10], as shown in Figure 5.2.

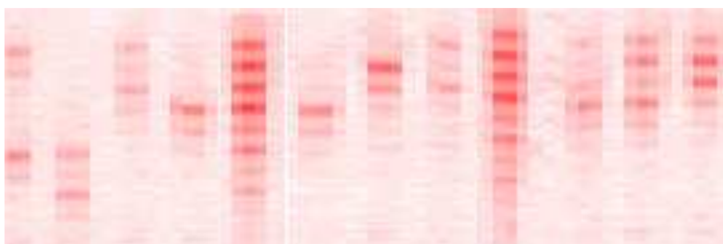


Figure 5.2. Allele binning by experimentally pooling DNA samples. DNA samples from the population are pooled together in single lane run-outs (lanes 5 and 9). The resulting allelic ladders are used for binning individual DNA samples' allele bands in the neighboring lanes, as is shown here. *(Provided by Lillian M. Bloch, Cybergenetics, Inc.)*

The basic idea of stutter crawling is to *computationally simulate* a pooled allelic ladder by superimposing all the available stutter data, as shown in Figure 5.3. One advantage of pooling the data computationally is that we can superimpose stutter data from different lanes, and from different gels — we require only that the data share common experimental conditions. Using additional knowledge (e.g. the marker's repeat size), we intelligently "crawl" along the highly redundant superimposed stutter data to compute an allelic ladder for binning alleles.

Box 5.1 describes the details of our stutter crawling algorithm. We bin the stutter bands in a *breadth-first* fashion across the gel lanes, instead of naively superimposing the stutter trails from all gel lanes and ignoring the lane information completely. To improve the robustness of our stutter crawling algorithm, we use local lane information to verify the

---

[10]In fact, this is a common practice in DNA forensics.

continuity of stutter trails within lanes, before globally crawling from one stutter to the next.

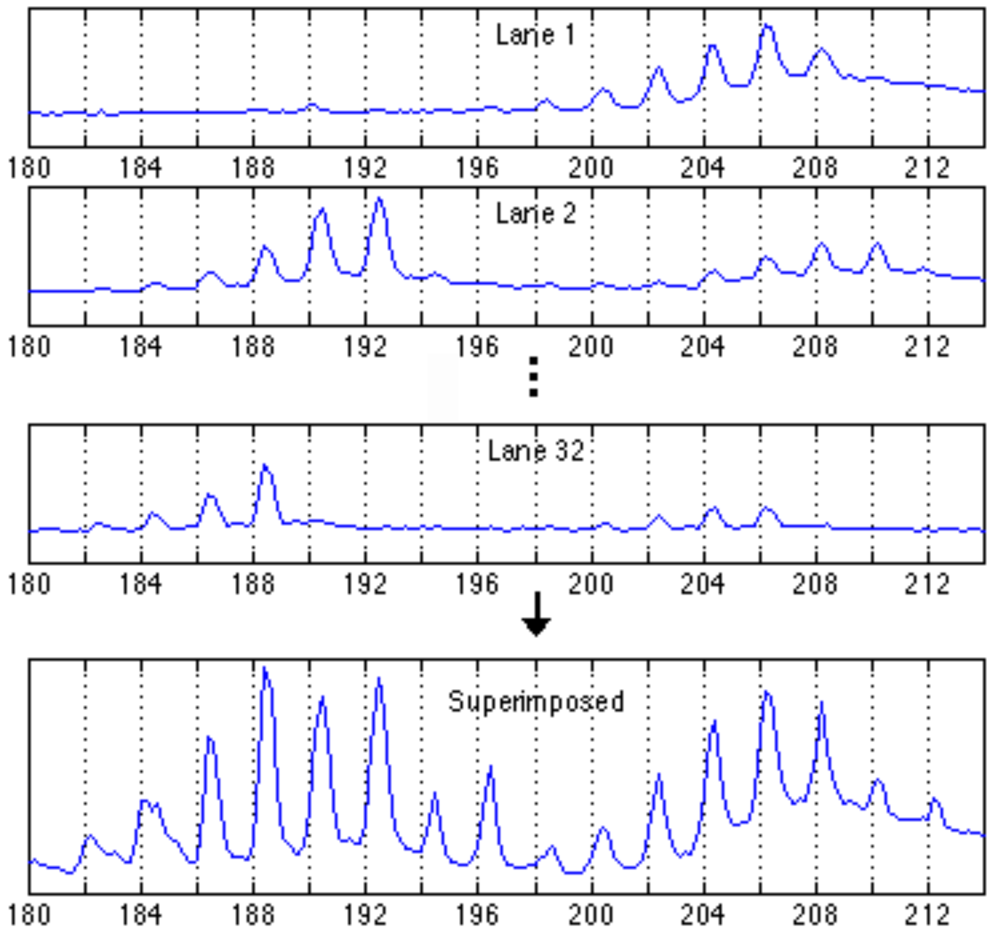


Figure 5.3. Computational pooling of stutter data from different gel lanes. The bottom display pane shows the result of superimposing 32 lanes of electropherograms for the dinucleotide repeat marker D16S511. By pooling a large number of lanes, we would be able to obtain a marker-specific allelic ladder that is of the requisite 2 bp resolution.

Algorithm: STUTTER_CRAWL

Step 1: Align stutter trails.

In each lane, identify potential stutter bands in the marker window. Then, align the stutter bands across lanes and gels by indexing the bands relative to their interpolated MW sizes. Thus, bands do not depend on specific pixel positions.

Step 2: Identify an "anchor" row.

Scanning across all data, identify a row that contains the most stutter bands with high intensities. In other words, determine a MW size m_α that maximizes the total number of high intensity stutter bands indexed by a size within the range of $m_\alpha \pm \varepsilon$. Here, ε is a heuristically determined bin width.

Assign $B_\alpha = \text{round}(m_\alpha)$ as the bin label for all stutter bands along this anchor row.

Step 3: Stutter crawl from anchor row.

(a) Starting from the anchor row, either crawl upwards (in increasing sizes) or downwards (in decreasing sizes) to search for a neighboring row of stutter bands that are all approximately n' bp away, where n is the number of nucleotides in a repeat unit of the marker, and n' is the corresponding interpolated MW size of such a unit. Ideally, $n=n'$. However, because of the chemical difference between the sizing DNA and the marker DNA, as well as the inherent error in real data, n' only approximates n. Initially, we use $n' = n \pm 0.5$.

(b) If a neighboring row with comparable number of stutter bands as those found in the anchor row is found, we add it to the set of binned alleles and assign it with the appropriate bin label $B_\alpha + n$ (if it is upwards from the anchor row) or $B_\alpha - n$ (if it is downwards from the anchor row). We also update locally the expected size (n') of a repeat unit of the marker with the difference between the mean sizes of the two consecutive bins.

(c) Repeat (a) and (b) by crawling from any of the binned alleles. Stop when no more neighboring stutter rows can be binned.

Step 4: Iterate.

Repeat steps 2 and 3 until no more segregated stutter rows can be binned.

Box 5.1. STUTTER_CRAWL: an allele binning algorithm for constructing a high-resolution marker-specific allelic ladder using *all* the stutter data.

5.2.2. Example

Figure 5.4 shows the actual gel[11] image for a dinucleotide repeat marker D16S511. The MW sizing grid (from Bioventures 20-bp ladder) provides an average sizing resolution of 20 bp. This resolution is too low for a dinucleotide repeat marker that has alleles only 2 bp apart. However, our stutter crawling algorithm can exploit the high degree of PCR stuttering in D16S511, and refine the sizing grid into one with the requisite 2-bp resolution.

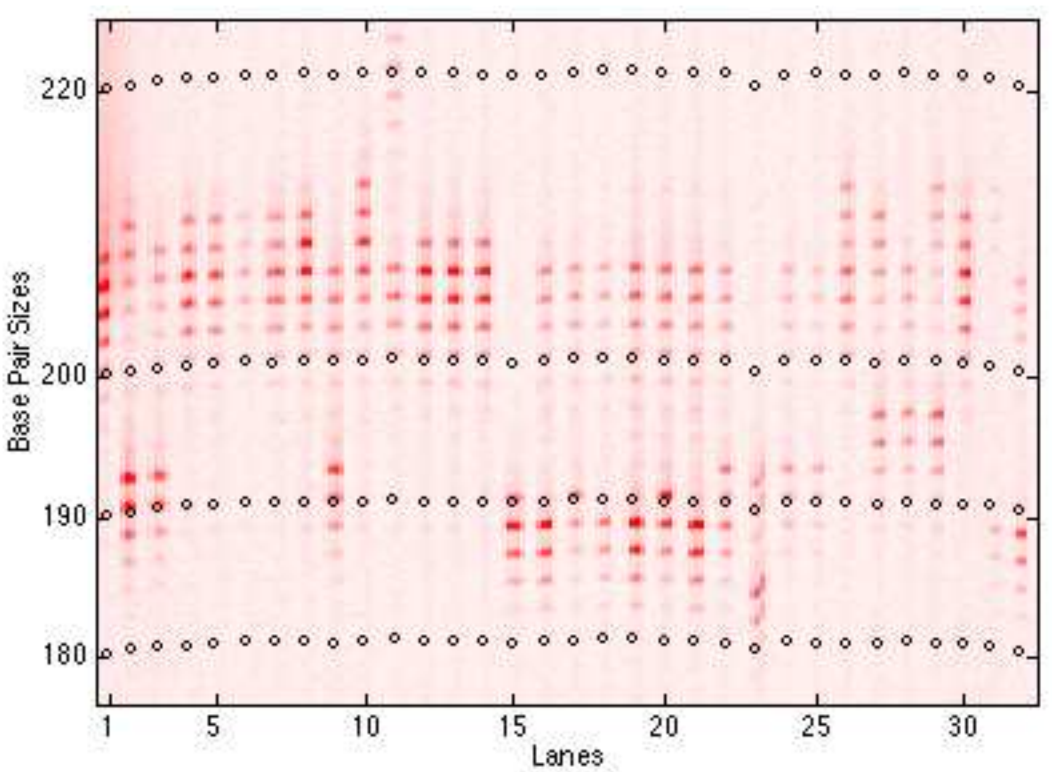


Figure 5.4. The marker image for the TET-labeled dinucleotide D16S511. There are 32 lanes on the gel shown. The "o"s indicate the locations of the MW (Bioventures 20 bp ladder) sizing grid from the TAM dye plane. The highest resolution provided is 10 bp,

[11]This is the gel that we shipped as the demo gel with the FAST-MAP package (see Chapter 8 for information about the FAST-MAP genotyping system).

which is too low for D16S511, a dinucleotide repeat marker requiring 2-bp sizing resolution. *(Data provided by Gordon Bentley, gene/Networks.)*

First, using the MW sizing grid, we align all the detected marker bands. In case we are aligning stutter bands from different gels, we use relative pixels to index the detected marker bands to avoid any local dependency on the absolute image pixel values. With sufficient data, a regular pattern emerges when all available lanes are aligned. Figure 5.5 shows high signal intensity marker bands in darker colors, making regular rows of dark bands visually apparent.

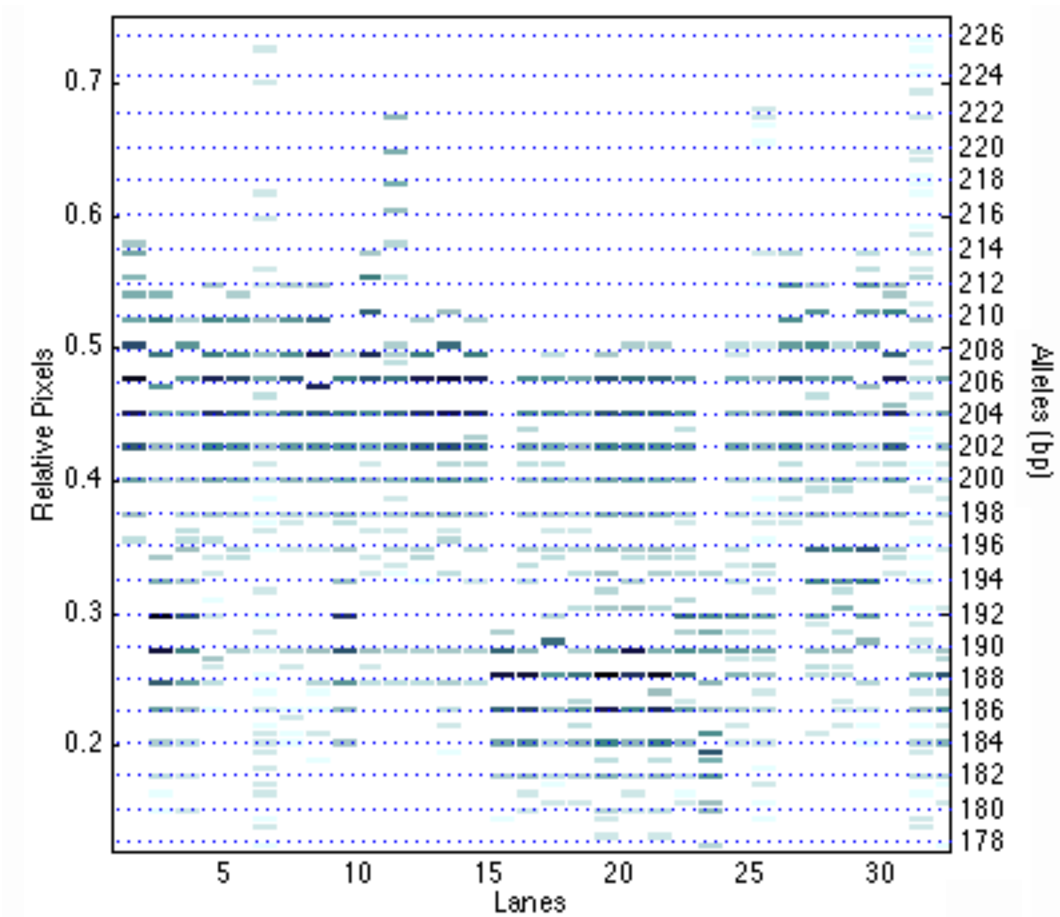


Figure 5.5. Binning of the TET-labeled dinucleotide D16S511 by stutter crawling. The detected marker bands in each of the lanes (32 lanes are shown) are aligned together; the darker bands having higher signal intensity than the lighter ones. The 2-bp ladder resulting from stutter crawling is shown by the horizontal grid lines, which are indexed by their integral allelic labels on the right.

To start stutter crawling on the aligned data, we locate the row with the most dark bands (i.e., the most visually distinct row) as the anchor row. For example, in Figure 5.5, we pick the row marked "204" bp. Picking a row that has many high intensity marker bands ensures that most are actual stutter bands that lie on stutter trails. This choice makes the row highly "extensible" — we can extend the ladder ("crawl") to stutter bands along the many emerging stutter trails. Figure 5.5 shows the 2-bp sizing ladder that we constructed for the gel using stutter crawling.

To investigate the effect of binning by stutter crawling, we show, in Figure 5.6, both the pre-binning MW sizing grid and the post-binning grid on the superimposed electropherogram from the 32 lanes on the gel (shown previously in Figure 5.3). With the original MW sizing grid, the marker bands generally lie slightly off the grid, incurring significant rounding errors that may result in inconsistent allele calling. With the refined grid, the marker bands now "snapped" closely to the grid, minimizing rounding errors that occur when assigning integral allele sizes to the marker bands. The average rounding error (or "bin width") for the 32 genotypes was reduced from 0.29 bp with the MW-sizing grid, to 0.17 bp with the refined grid.

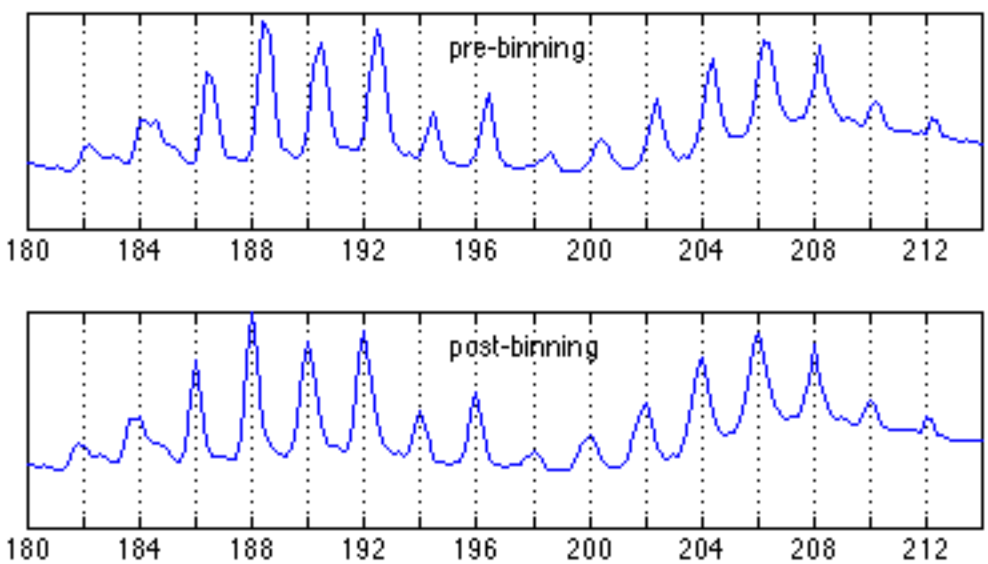


Figure 5.6. Sizing grid refinement. In the top pane, the pre-binning MW sizing grid is overlaid on the superimposed electropherogram of 32 lanes of D16S511 (see also Figure 5.3). The marker bands lie slightly off the MW sizing grid. In the bottom pane, the post-stutter crawling grid is overlaid. The marker bands now snap tightly to the refined grid.

5.2.3. Discussion

To reiterate, the possible chemical difference between the MW DNA and the marker DNA makes the MW sizing grid inadequate for sizing the marker bands — 1 bp of MW DNA may *not* correspond exactly to 1 bp of marker DNA. We illustrate this in Figure 5.7 by plotting the calibrated MW sizes of the allele bands of marker D16S511 against the actual allelic sizes, and compute the slope of the resulting line. With the unadjusted pre-binning MW-sizing grid, the size calibration slope gives a value of 1.92 bp/repeat for marker D16S511. This means that when sized with the Bioventures 20-bp MW ladder, each repeat of D16S511 is actually equivalent to only 1.92 bp of the MW DNA molecules. On the other hand, the size calibration curve adjusted with stutter crawling gives the correct slope of 2 bp/repeat.

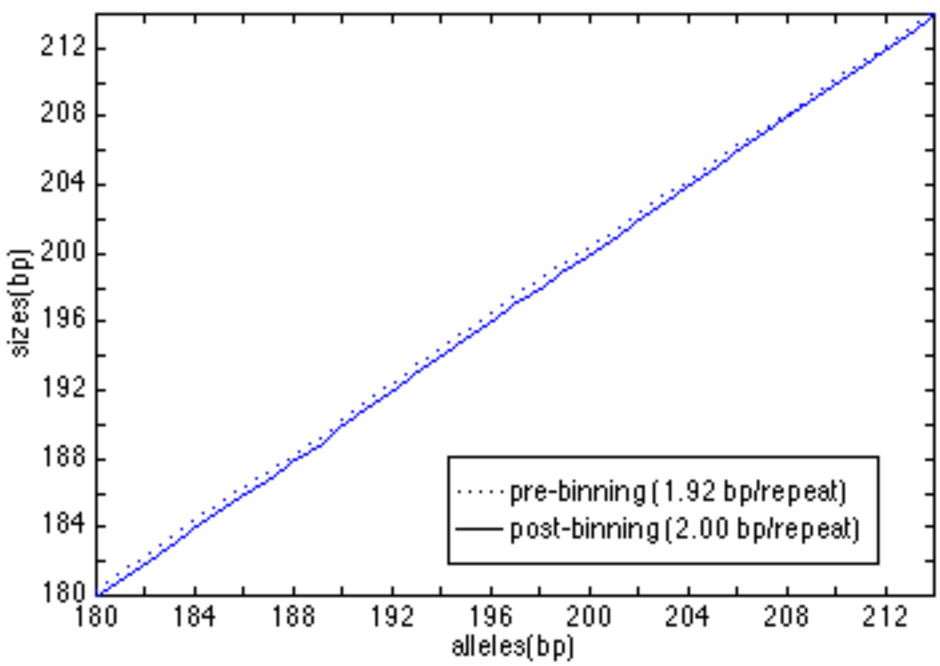


Figure 5.7. The size calibration curves for the dinucleotide D16S511. We plot the interpolated sizes against the actual allele sizes. With the MW-sizing grid ("pre-binning"), the slope gives 1.92 bp/repeat, whereas with the marker-adjusted grid ("post-binning), the sizing slope is an accurate 2.00 bp/repeat.

We show the relative sizing of the other dinucleotide repeat markers in our example gel in Table 5.1. The MW-calibrated allele size differences range from 1.84 bp to 1.98 bp per repeat (instead of 2 bp per repeat). The MW sizing discrepancy is especially pronounced on this gel with markers that have larger allele sizes (e.g. D16S515 and D16S503). The associated rounding error can lead to a miscall when the alleles of a genotype are far apart.

marker	dye	allele window	MW size per repeat
D16S405	FAM	103-161 bp	1.98 bp
D15S165	HEX	176-224 bp	1.96 bp
D16S511	TET	178-238 bp	1.92 bp
D16S503	HEX	290-326 bp	1.88 bp
D16S515	FAM	316-366 bp	1.84 bp

Table 5.1. The MW sizing of the 5 markers on the example gel (32 lanes). All the markers are dinucleotide repeats, with an expected spacing of 2 bp per repeat. The gel was sized with BVMap from Bioventures (20-bp MW ladder).

5.3. Quantitating DNA concentration

The DNA concentrations at the data bands can be quantitated using either the areas of the data bands or the heights of the corresponding peaks in the electropherograms. Because of band overlap, it is often difficult to determine these values independently, since neighboring data bands may overlap one another. Conventional genotyping systems generally ignore the effects of band overlap altogether, using truncated areas or densitometric intensities at the band centers as rough estimates of the DNA concentrations.

A common approach for handling band overlap is to fit the data bands *parametrically* to a model shape function using least square minimization (Galat, 1989; Ribeiro and Sutherland, 1991; Vohradsky and Pánek, 1993). Each fit is then subtracted from the electropherogram, and the process is iterated to fit the remaining data bands. One major advantage of this method is that each band is fitted locally with the model function using individually optimized parametric values. This can account for any local variation in the smearing function. Computationally, the accuracy and efficiency of this method rely heavily on the model function and the data quality. With a wrong model function, the method may not converge to a satisfactory fit. We must therefore carefully select an appropriate data band model.

5.3.1. Data Band Model

After experimenting with a variety of model peak shapes, we found that a function that has a Gaussian left half and a Runga right half fits gel electrophoretic data best (Richards and Perlin, 1995). We call it the *Gauss-Runga* function :

Gauss-Runga data band model function:

$$\Gamma(c,h,\sigma,\upsilon)=\begin{cases} h\cdot e^{-\frac{(c-x)^2}{\sigma^2}} & , \quad x \le c \\ \dfrac{h}{1+\left(\upsilon\cdot(x-c)\right)^2} & , \quad x > c \end{cases}$$

where c is the center of the band, h the height, σ the half-width of the Gaussian left half, and υ the scale factor of the Runga right half. Γ ranges over the x- (base pair) coordinate.

Box 5.2. Gauss-Runga: A hybrid model function that is useful for fitting data bands from gel electrophoresis.

Figure 5.8 shows an example of a Gauss-Runga peak. Assuming that DNA migrates from right to left, the longer Runga tail on the right models the trailing effect of the DNA as it migrates through the gel. In our experience, the Gauss-Runga model function has been very useful in fitting data from current DNA sequencers such as the ABI machines, and the Pharmacia ALF. Of course, a different model function will be used to fit the data should a new sequencer produce data bands with a different shape.

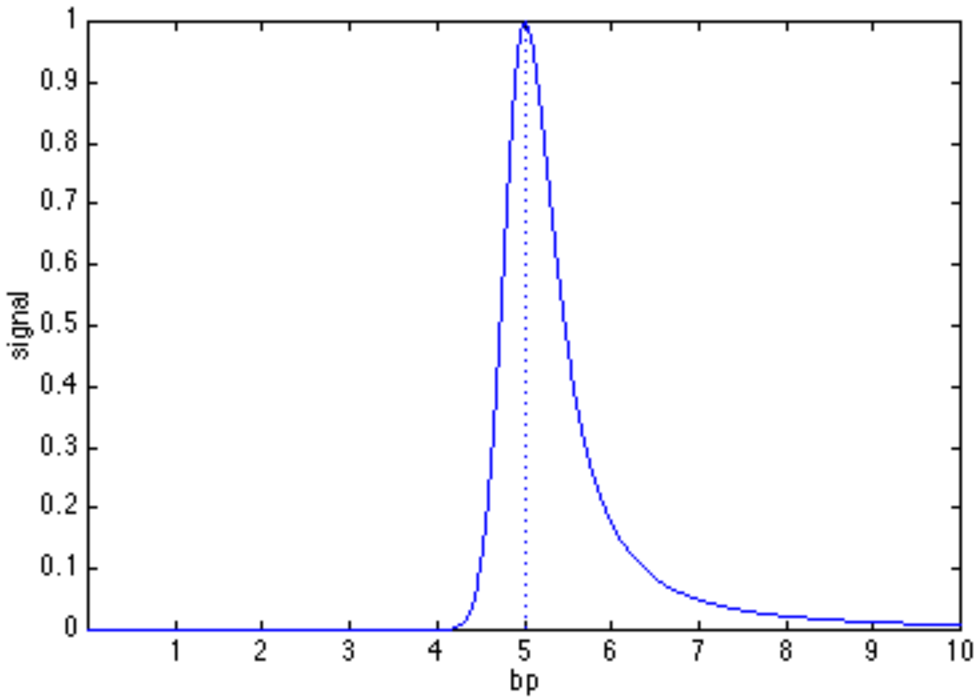


Figure 5.8. Gauss-Runga model function. The center of this peak is at the 5 bp mark. Its Gaussian half width for the left half is 0.33 bp, and its Runga scale is 2.17 for the trailing right half.

5.3.2. Algorithm

To fit a marker band with a Gauss-Runga function, there are four parameters to optimize: the band center *c*, the height *h*, the Gaussian half-width σ, and the Runga scale factor υ. The band center *c* can be initially approximated from the stutter crawling algorithm's allele bins' position. The other three peak parameters (height *h*, Gaussian half-width σ, and Runga scale factor υ) are inter-dependent, and should ideally be determined using global nonlinear optimization. However, the computational requirement for globally optimizing three parameters per peak is impractical for real-time systems. Moreover, such computation would be overkill, since the data bands generally only overlap their immediate neighbors. Based on these computational considerations, we adopt a hybrid "locally-optimize globally-refine" approach for fitting the peaks:

- *locally optimize:* First , we optimize *h,* σ, and υ locally;
- *globally refine:* Then, we apply the global optimizer on *h* alone, as the band height is most sensitive towards band overlap.

We repeat this process to incrementally refine the solution by comparing the overall fit with the observed electrophoretic data. Quantitating each genotyping experiment typically takes several seconds. This reduction in the computational cost is three orders of magnitude less than the original "global" algorithm. With thousands of experiments to quantitate per gel, this improvement is important.

Box 5.3 details the QUANTITATE_BAND algorithm.

Algorithm: QUANTITATE_BAND

Step 1: Estimate an initial fit.

For efficiency and convergence in the least-square minimization process, it is important to accurately estimate the initial fit.

Using the allele bin positions computed by the STUTTER_CRAWL algorithm, we search for data bands in the marker window with nontrivial heights. Starting with the tallest data band detected in the marker window, we determine the initial values for the parameters c, h, σ, and υ by finding a best fit *locally* within the local region $c-0.5 < x < c+0.5$, where c is the expected bin center (from STUTTER_CRAWL) for that allele. When done, we subtract the fitted band $\Gamma(c,h,\sigma,\upsilon)$ from the observed electropherogram, and then iteratively estimate the parameters for the next tallest band in the marker window until the parameters for all the data bands are estimated.

Step 2: Locally refine each parameter individually.

(a) Band centers c: Starting with the most confident (i.e., tallest) band, locally micro-adjust its center c for the best fit relative to the entire electropherogram. Repeat (a), using the next most confident data band.

(b) Gaussian half widths σ: Perform (a), adjusting σ instead of c.

(c) Runga scale factors υ: Perform (a), adjusting υ instead of c.

(d) Heights h: Perform (a), adjusting h instead of c.

Step 3: Globally optimize the band heights.

Since h is the parameter most affected by band overlap, globally optimize all the heights simultaneously. For efficiency, it is important to use a fast nonlinear optimizing algorithm, such as the Levenberg-Marquardt method (Marquardt, 1963). Fortunately, because the computationally tedious steps 1 and 2 produce a close initial estimate, the nonlinear optimization of h typically converges rapidly.

Step 4: Iteratively refine (best-first search).

Repeat Steps 2 and 3 until the sum of squares error of the fit versus the observed electropherogram is minimized.

Step 5: Output band quantitations.

To index each detected data band, assign it the bin label b (such that the band center c falls within the bin widths of allele bin b). Using the height h as the measure of DNA concentration[12], we output the pairs $\{\langle b, h\rangle\}$ to form the transformation function for band quantitation:

$$\beta: \text{ image data + sizing grid} \rightarrow \{\langle b, h\rangle\}$$

Box 5.3. QUANTITATE_BAND: A least-squares minimization band fit using the Gauss-Runga model function.

5.3.3. Example

This example is from D20S195, a FAM-labeled dinucleotide repeat marker loaded on a 34-lane ABI/377 gel, with the GS500 sizing standard in the TAM dye. The marker's electropherogram from one of the gel's lanes is shown in Figure 5.9.

The first step is to identify the marker bands along the continuous intensity profile in the electropherogram. Using the binned sizing grid constructed by the STUTTER_CRAWL algorithm, we search in the expected locations (shown as vertical grid lines in Figure 5.9) for potential data peaks. The maxima of the detected peaks (marked "*" in the figure) are used as the peak centers (parameter c).

[12]When band overlap effects are eliminated mathematically, the band heights work as well as band areas as an estimate of DNA concentration. In fact, with real data, band areas may be less robust than band heights because they are more sensitive to baseline and discontinuity artifacts in the intensity profile. When using band area for the DNA concentration in ß, use the closed functional form of $\Gamma(c,h,\sigma,\upsilon)$ in Box 5.2.

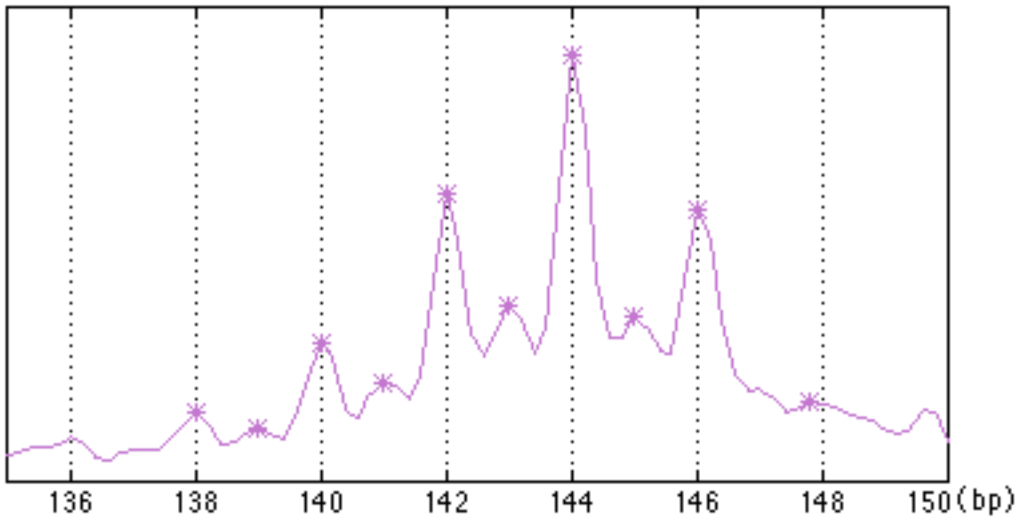


Figure 5.9. D20S195's lane electropherogram gridded by the marker-adjusted sizing grid. Using the pre-binned grid lines as expectations, we quickly and reliably detect the marker peaks (marked as "*"'s in the electropherogram).

Starting from the highest peak detected (at 144 bp), we iteratively estimate the initial values for the other peak parameters for each of the detected peaks: the peak height *h,* the Gaussian half-width σ, and the Runga scale factor υ , as described in Step 1 of the QUANTITATE_BAND algorithm. Figure 5.10 shows the sum of the initial fits.

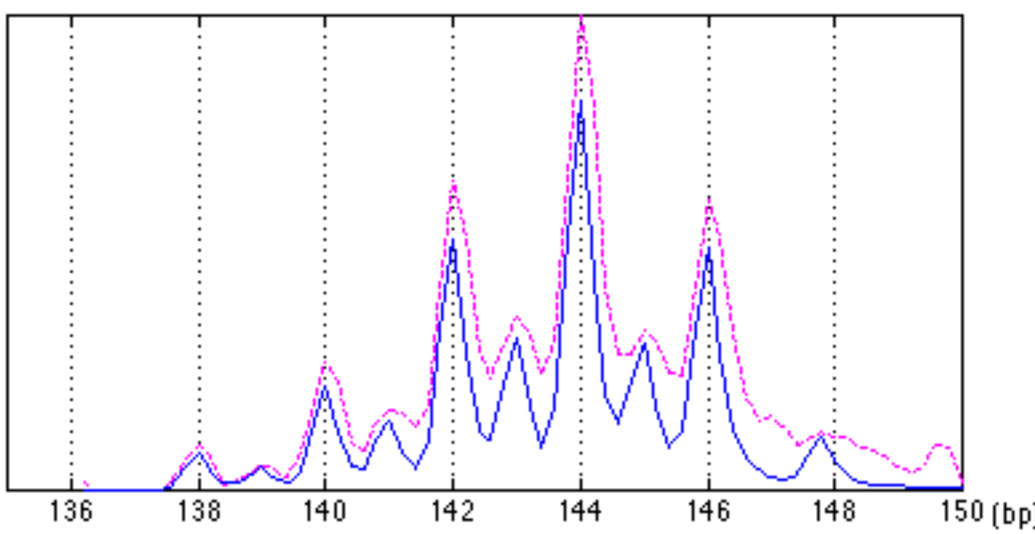


Figure 5.10. Initial sum of fit. The solid line shows the initial sum of the fit from iteratively estimating an initial fit for each detected peaks individually. The original electropherogram is plotted with a dashed line (of a lighter color) above the fitted profile.

With the initial fit, we proceed to Step 2 of QUANTITATE_BAND, where we locally refine each of the peak parameters in the following order: Gaussian half-widths σ, Runga scale factors υ, and peak heights h. Figure 5.11 shows the results of locally refining the Gaussian half-widths σ. Since this parameter affects only the left halves under our Gauss-Runga peak model, the resulting sum of fit shows marked improvement on the left sides of the fitted peaks.

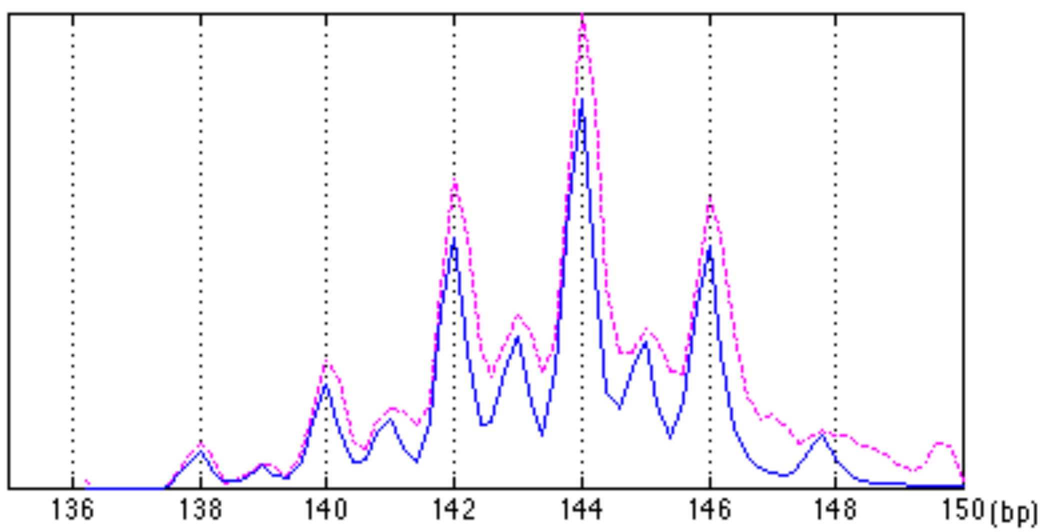


Figure 5.11. Sum of fit after locally optimizing the Gaussian half-widths (σ) of the detected marker peaks. The resulting sum of fit (dark solid line) shows noticeably closer fit to the original electropherogram (light dashed line) on the left sides of the marker peaks. This is because under our peak model, only the left halves are Gaussian.

Similarly, by locally optimizing the Runga parameters, the resulting sum of fit shows improvement on the right halves of the marker peaks, as shown in Figure 5.12.

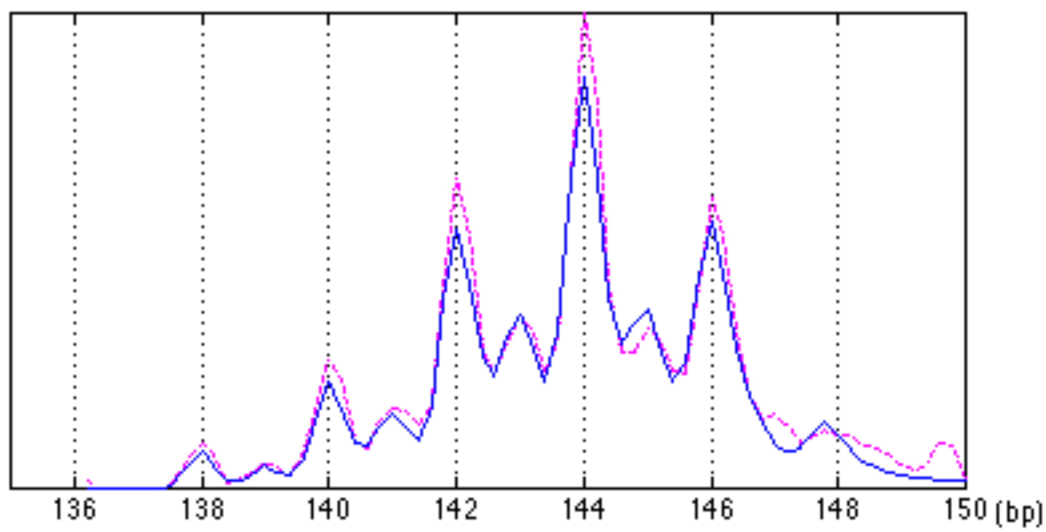


Figure 5.12. Sum of fit after locally optimizing the Runga scale factors (υ) of the detected marker peaks. This time, the resulting sum of fit (solid line) shows noticeably closer fit to the original electropherogram (dashed line) on the right sides of the marker peaks, since the right-half of our peak model is defined by the Runga parameter.

The final local refinement step optimizes the peak heights. As shown in Figure 5.13, the local optimization process has generated a fairly accurate fit.

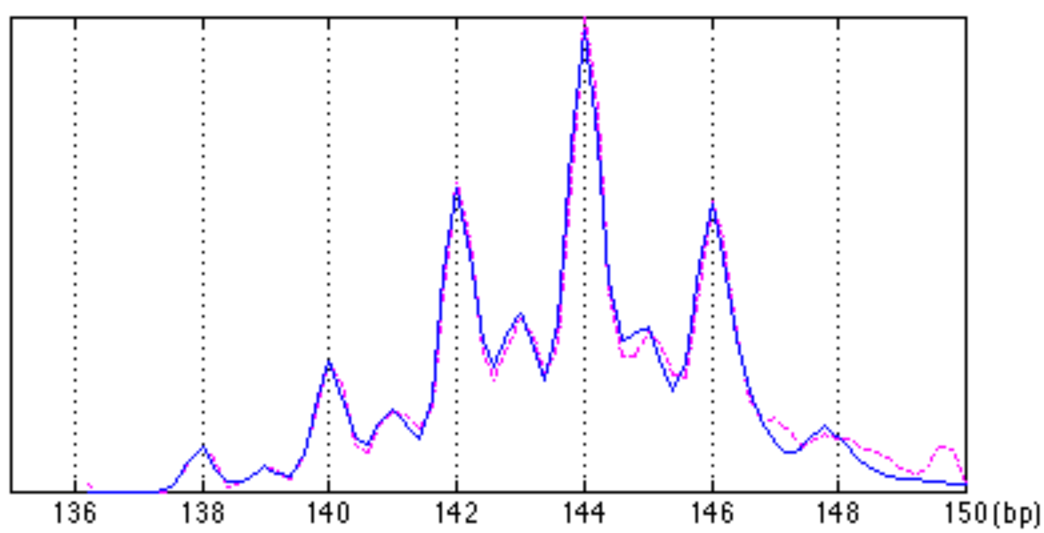


Figure 5.13. Sum of fit (dark solid line) after locally optimizing the peak heights (h) of each detected marker peaks. This is a fairly close fit to the original electropherogram (light dashed line) after one round of locally optimizing peak parameters.

In Step 3 of QUANTITATE_BAND, we refine our initial solution by globally optimizing all the peak heights simultaneously. In principle, global optimization is computation-intensive. However, the good initial fit from the efficient local optimization steps significantly reduced the global search effort. Figure 5.14 shows the globally optimized sum of fit.

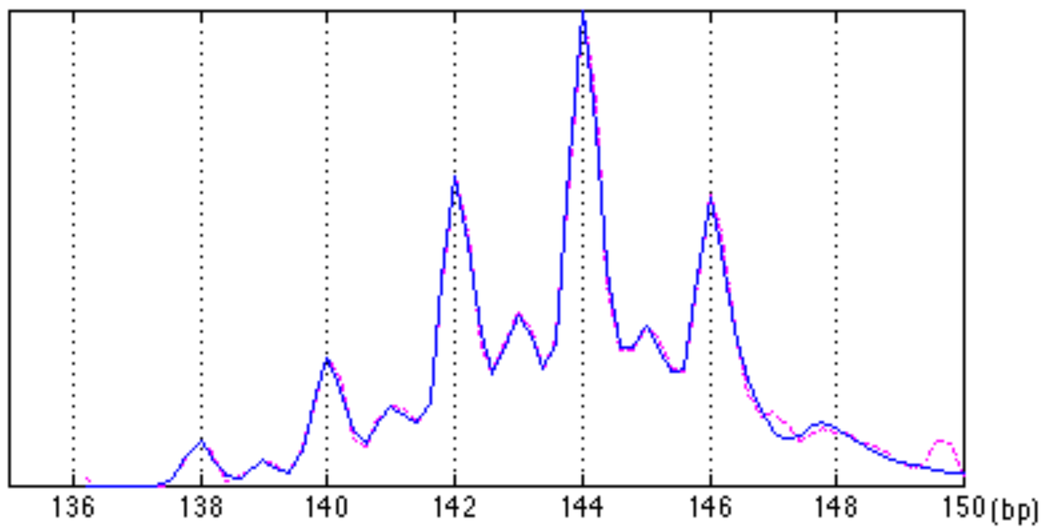


Figure 5.14. Sum of fit (dark solid line) after globally optimizing the peak heights.

For the best fit, we iteratively refine (Step 5 of QUANTITATE_BAND) the solution by repeating the "locally-adjust, globally-optimize" process. In this example, only one iteration was required. The final sum of fit is shown in the top pane in Figure 5.15. For output, we assign an integral size label to each detected band, and compute the relative DNA concentrations. For the integral size labels, we use the molecular sizes from stutter crawling, rounding if necessary. For the DNA concentration, we can use band height or area under each fitted curve (using the closed functional form of $\Gamma(c,h,\sigma,\upsilon)$ in Box 5.2). We show the final output of the band quantitation step in the bottom pane of Figure 5.15.

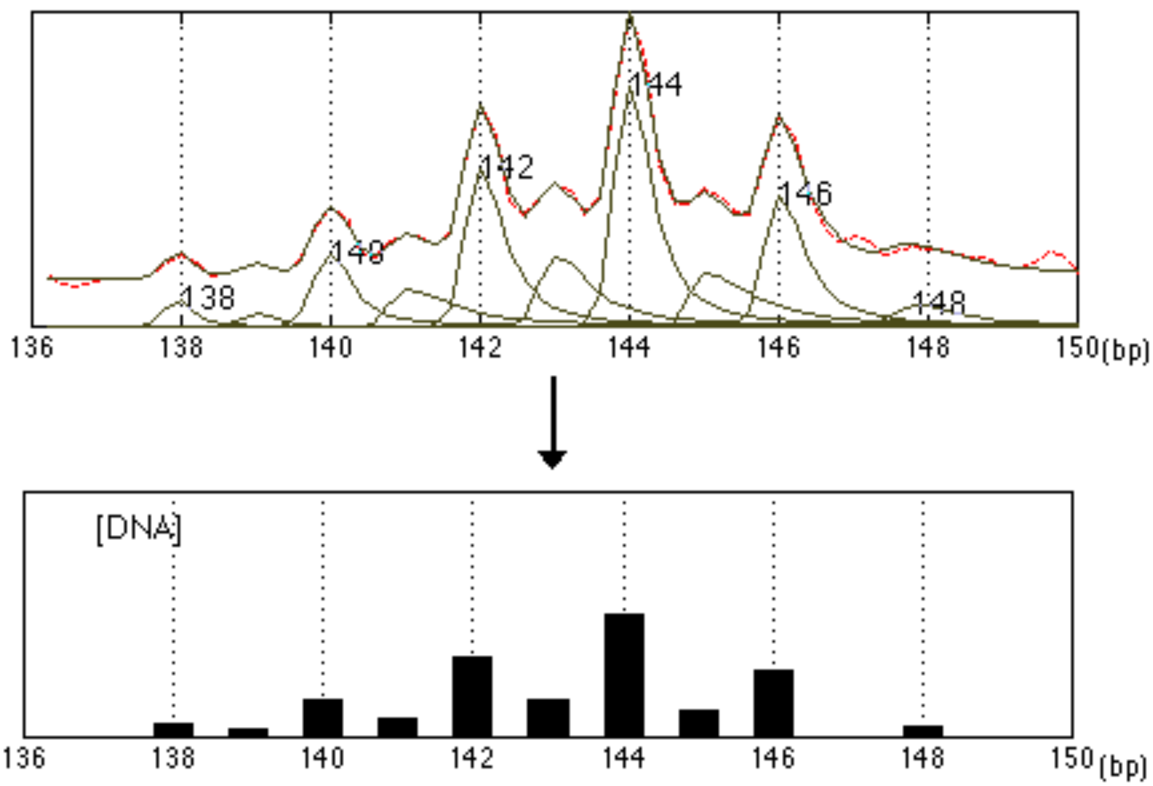


Figure 5.15. Band quantitation output. The top pane shows the final sum of fit (solid line) and the individual peaks beneath it. The bottom pane shows the output of the band quantitation step: each darkened bar represents a discretized data band located at an integral allele size. The heights of the bars depict the computed relative DNA concentrations.

5.3.4. Discussion: Developing algorithms

In this chapter, we have presented algorithms for solving two main problems in band quantitation, namely: *stutter crawling,* which solves the allele binning problem, and *gauss-runga peak fitting,* which solves the DNA concentration quantitation problem. Although these are two straightforward algorithms, the process involved in developing them (as well as the other algorithms described in this dissertation) was not as straightforward. A typical development process involved making a few false starts along the way, discovering new facts about the ever-changing domain, and going through several cycles of algorithm refinements. We describe, in this section, a brief representative account of how we developed the band quantitation algorithms.

Stutter crawling

The conventional approach to the allele binning problem was to first call the alleles using molecular sizes interpolated from the MW sizing grid, and then bin the two alleles by assigning appropriate integer size labels to them (Ghosh *et al.*, 1997). This approach works when the stutter bands are not treated as part of the data, and therefore do not play a role in the allele calling step.

Following the conventional approach, our initial allele binning solution did not involve binning the alleles for all the data bands (including the stutter bands) in the quantitation step. Instead, our initial solution involved quantitating the data bands for DNA concentrations, and then calling the alleles using various stutter deconvolution algorithms (to be described in the next chapter). As it turned out, our allele calling algorithms performed poorly on real data, even though they had worked very well on simulated quantitated data (Section 6.4 and Appendix A).

Initially, it appeared that the poor performance was due to the allele calling algorithms, but our efforts to improve our stutter deconvolution methods were to no avail. On further investigation, the source of the problem turned out to be in the stutter patterns used by the deconvolution algorithms for calling alleles. Because we did not pre-bin the stutter bands, sizing interpolation errors led the computer to learn stutter patterns that were inconsistently sized. Since the computer relied on these compiled patterns as definitive expectations in guiding its search for the correct alleles, our expectation-based allele calling algorithms were adversely affected by the sizing errors.

The stutter band sizing problem led us to develop a novel technique ("stutter-crawling") for binning stutter data. By pre-binning *all* data bands during the quantitation step, the computer was able to learn stutter patterns that are consistently sized, and the power of our stutter deconvolution algorithms in allele calling began to unveil. Our venture into stutter binning also led us to discover an unexpected source for binning errors: in addition to the rounding errors associated with MW sizing interpolation, allele binning can also be affected by a sizing mismatch due to the chemical differences between the MW DNA and the marker DNA (see Table 5.1), when we used MW data to calibrate the marker data.

Our experience with the allele binning problem is representative of the high degree of *interdependency* between the various algorithms described in this dissertation. The real cause of errors in one algorithm (say, allele calling) may have been due to errors in another algorithm (say, band quantitation) that pre-processes the input data to the algorithm. In solving a complex problem reduced into multiple subproblems, it is important to keep a global perspective of the interdependencies between the various subproblems when debugging the algorithms.

There is another aspect of our allele binning experience that is typical of algorithm development in an *emergent* domain such as molecular genetics. While attempting to improve our algorithmic solutions, we also discovered new non-algorithmic causes of errors, such as the chemical factor of the sizing errors. In a complex problem domain that is ever-changing with new facts and discoveries, it is important for the computer scientist to be closely involved with the process of discovery in the problem domain, so that the algorithms designed can be easily adapted to the latest domain changes.

Band quantitation

Our experience in developing the *gauss-runga peak fitting* (QUANTITATE_BAND) algorithm illustrates the power of algorithmic refinements. In our initial QUANTITATE_BAND algorithm, we globally optimize *all* the band parameters (band centers c, Gaussian half widths σ, Runga scale factors ν, and peak heights h). This naive strategy resulted in a huge multidimensional search space consisting of about 50 peak parameters[13] to be simultaneously and globally optimized . Quantitating just one genotyping experiment took hours of computation, even on a powerful computer. Furthermore, despite the intense computation invested, the computer did not always find a satisfactory solution, as there were many local minima in the vast search space, entrapping the computer in its unconstrained global search.

Our first algorithmic refinement was to prune the multidimensional search space, using two common artificial intelligence (AI) approaches: (1) we greatly restricted the range of values that the peak parameters can be optimized to, and (2) we heuristically improved on the accuracy of the initial solution for the global search. Using these two standard AI techniques, we were able to reduce the computation time about ten-fold.

---

[13]For a marker that exhibits PCR stuttering, there are typically 10–15 data bands per genotyping experiment. Each data band is defined by 4 peak parameters (c, σ, ν, and h), resulting in a total of about 40–60 peak parameters.

Our next attempt was to significantly reduce the number of dimensions of the quantitation search space, by effectively reducing the number of peak parameters for global optimization. Instead of applying the global optimizer to all the peak parameters simultaneously, we optimized each of the peak parameters *locally,* and then globally refined only the peak parameter most sensitive to band overlapping artifacts (namely, the peak heights h). We then iteratively refined on the parametric fit to ensure convergence. This third refinement step resulted in another ten-fold speedup, allowing the computer to quantitate each genotyping experiment in seconds.

This exercise in algorithm refinement showed how basic search space reduction techniques in computer science can bring about dramatic improvements in algorithms designed for solving *data-intensive* problems. In the case of QUANTITATE_BAND, we were able to reduce the analysis time at least 100-fold, enabling the computer to analyze a highly multiplexed gel in a couple of hours.

5.3.5. Discussion: "Plus-A" artifacts

Solving a complex interdisciplinary problem requires collaborative problem solving from all involved domains. Although it is our thesis to advocate using *computational* approaches for solving the microsatellite genotyping problem, we must not overlook the importance of not re-solving problems that can be easily eliminated with alternative approaches. In this section, we describe the "plus-A" artifact problem where there is an *experimental* solution that is better than the *computational* counterpart. Interdisciplinary solutions should be expected when solving a complex interdisciplinary problem such as the one addressed in this thesis.

Experimental versus Computational solutions

One advantage of our band quantitating approach, in addition to accounting for band overlap and providing accurate DNA concentration measures, is that it allows us to mathematically excise "plus-A" artifacts from the data. During the PCR amplification of microsatellite markers, polymerases sometimes add a trailing adenosine base to a synthesized DNA strand. This causes the "plus-A" artifact, where not one, but two bands appear for every band[14]. The combination of "plus-A" and "PCR stutter" artifacts can produce a pattern of many bands that are separated by only one base pair, and since the artifact can be highly variable from run to run, it can be unclear which is the correct allele.

To computationally excise the "plus-A" bands from the data, we need to quantitate each marker band individually. This is done in our band quantitation, where we quantitate each marker band with an individualized Gauss-Runga peak model. However, there is still the problem of determining which marker bands to excise from the data. We must make assumptions about the distinguishing characteristics of the "plus-A" bands, such as whether the "plus A" bands are of lower intensity than the actual bands, or whether the "plus A" bands are those of odd-numbered sizes (say).

A more robust approach is to *experimentally* remove the "plus-A" artifact from the data, and restore the two base pair separation. This can be done by forcing the polymerase to *suppress* "plus-A"s, or preferentially *enhance* "plus-A"s (Magnuson *et al.*, 1996). Since new, simple experimental procedures effectively remove the "plus-A" s (using the "G-clamping" or "PIG-tailing" techniques), most genotyping centers now generate clean "plus-A"-free genotyping data. In the case of "plus A" artifact reduction, the experimental approach is more elegant than the computational solution. Nevertheless, band quantitation is still essential for removing the band overlap effects and for quantitating marker bands with accurate relative DNA concentrations. Such exquisite band quantitation is used by the deconvolution genotyping algorithms that we describe in the next chapter, and also for other non-genotyping applications such as computer-based differential display analysis (Jones *et al.*, 1997; Liang and Pardee, 1992; Luehrsen *et al.*, 1997).

---

[14]Our example marker D20S195 has a small degree of "plus-A", producing small "plus-A" peaks in between the major evenly-spaced ones (see Figure 5.9).

6. Allele Determination

In this chapter, we address the final step in microsatellite genotyping: calling the alleles from quantitated electrophoretic data. Problems in allele determination include:

- PCR stuttering: how to eliminate the shadow bands in the data to recover the underlying true alleles?
- Relative amplification: how to account for unequal amplification of alleles in the genotypes?
- Pattern specificity: how to acquire, maintain, and apply marker- and allele-specific patterns intelligently and efficiently?

These problems are PCR limitations that cannot be eliminated experimentally. Therefore, we will show how to resolve them *computationally*. Our final transformation function for the genotyping problem is:

γ: quantitated data with stutter $\rightarrow$ genotypes

6.1. Convolution Model

PCR is analogous to an *amplifier* in conventional signal processing. With perfect fidelity, the amplification of a single allele is expected to produce only a single band on a gel. With an imperfect amplifier, as is in the case of PCR, a distortion response is introduced which causes an allele to generate *multiple* bands instead. In a heterozygotic genotype, if the two alleles are close to each other with respect to their molecular sizes, their trailing shadows overlap and create a convoluted band pattern when size-separated on an electrophoretic gel (Figure 6.1).

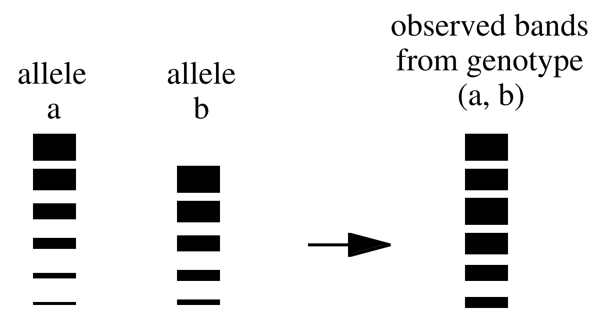


Figure 6.1. Overlapping of PCR stutter bands. Stutter patterns from the alleles in a genotype combine together to form a convoluted band pattern.

Under fixed PCR conditions (including enzyme, cycle times, number of cycles, template and primer concentrations, and buffers), the stutter pattern of each allele of a given marker is known to be *reproducible* (Perlin *et al.*, 1994). This means that we can handle PCR stuttering in microsatellite markers by applying standard techniques from electronic signal processing (Papoulis, 1977) for modeling reproducible responses of an amplifier. Using signal processing techniques, we can accurately model PCR stuttering as a *convolution:*

<u>Convolution model for PCR stuttering</u>:

$$y = A \cdot x$$

A models the marker's distortion responses (i.e. stutter patterns), x represents the underlying genotype, and y denotes the predicted convoluted stutter pattern that would be observed on the gel.

Stutter patterns from a single marker typically vary from one allele to another, as shown in Figure 6.2. Usually, alleles with bigger sizes produce flatter and longer stutter trails than alleles with smaller sizes (for example, compare the stutter patterns for the 155 bp allele with the 135 bp allele in Figure 6.2)[15]. Conventional (linear shift-invariant) convolution model uses a single distortion vector — this does not adequately model PCR stutter. Instead, we use a non shift-invariant convolution with one distortion vector for *each* allele. To represent all the possible stutter patterns in a marker, we write down the stutter pattern for each allele in a *matrix A*. Figure 6.3 shows an example of an actual stutter matrix A for the dinucleotide repeat marker D22S283. Each column in A records the specific stutter pattern of one allele for the marker.

[15]One possible explanation for this stutter variation is that the polymerase tend to "slip" more with the bigger alleles during PCR, creating more stutter bands than with the smaller alleles.

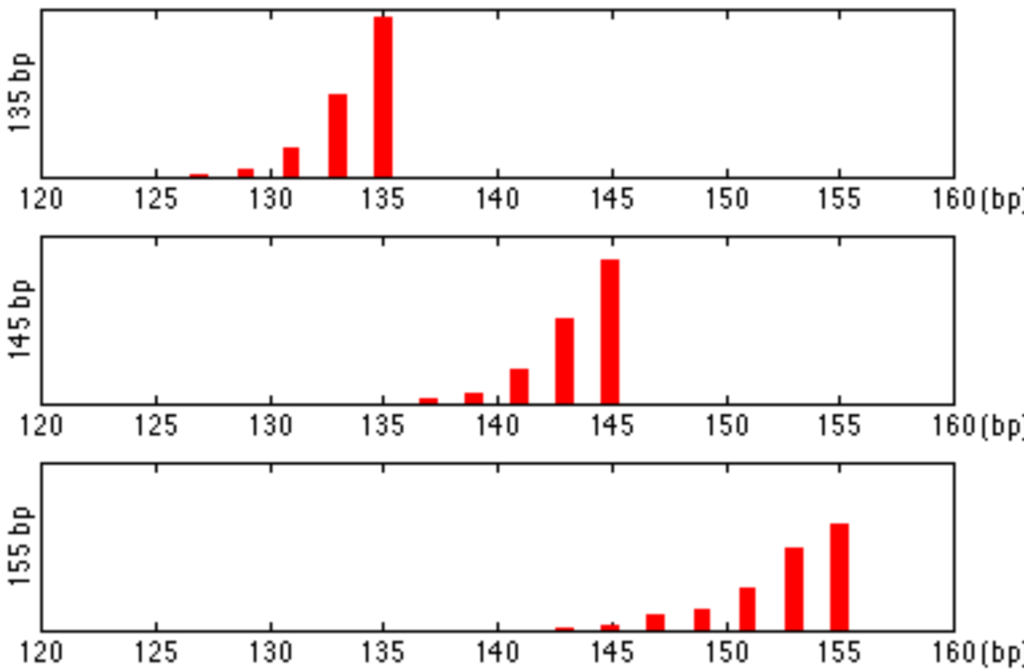


Figure 6.2. Stutter patterns for alleles 135 bp, 145 bp, and 155 bp for TET-labeled marker D22S283. D22S283 is a dinucleotide repeat marker with alleles ranging from 135 to 165 bp. The stutter patterns shown were acquired from an ABI 377 gel running 16 lanes of marker data for D22S283.

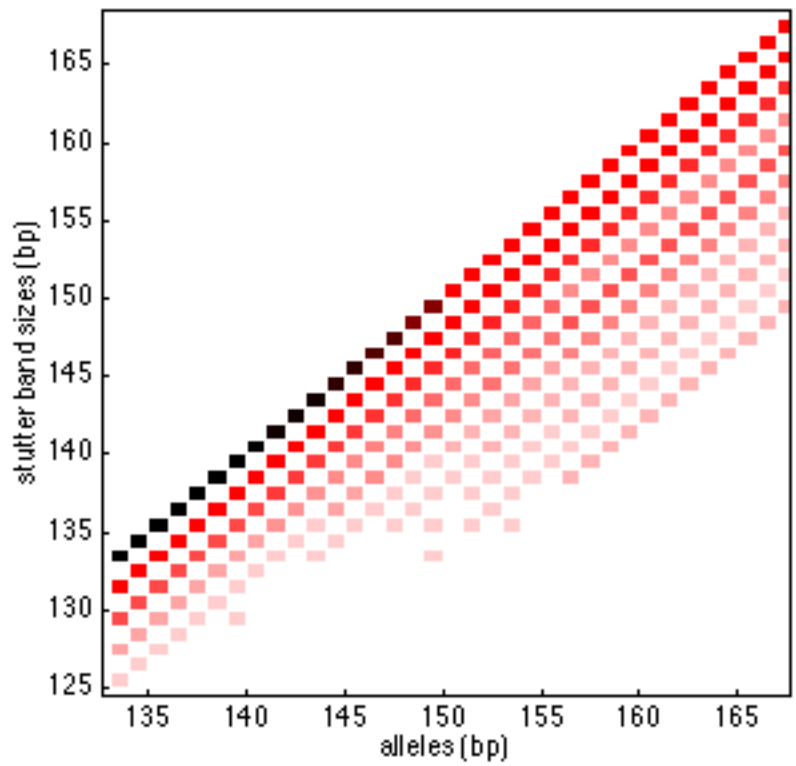


Figure 6.3. The stutter matrix A for marker D22S283 acquired from an ABI 377 gel running 16 lanes of marker data. In particular, note the gradual increase in stutter length from the smaller alleles to the bigger ones.

In our convolution model, the true genotype is represented by a column vector x indexed by the possible marker alleles. Each row entry in x records the number of copies of a particular allele that is present in the genotype. For a heterozygote, x will be a (0,1)-vector with two 1s for the two distinct alleles in the genotype. For a homozygote, x will be a (0,2)-vector with a single entry in the vector that has the value 2.

When the PCR product of a DNA sample is size separated on an electrophoretic gel, the stutter bands from each allele in the genotype combine additively (as shown in Figure 6.1 previously). This convolution process is mathematically equivalent to a simple matrix-vector multiplication:

$$y = Ax$$

where y is the response vector in our convolution model that predicts the relative DNA concentrations in the resulting PCR product. In other words, given the true genotype x, the vector y predicts the relative signal intensities of the shadow bands that would be observed on the gel (see Figure 6.4).

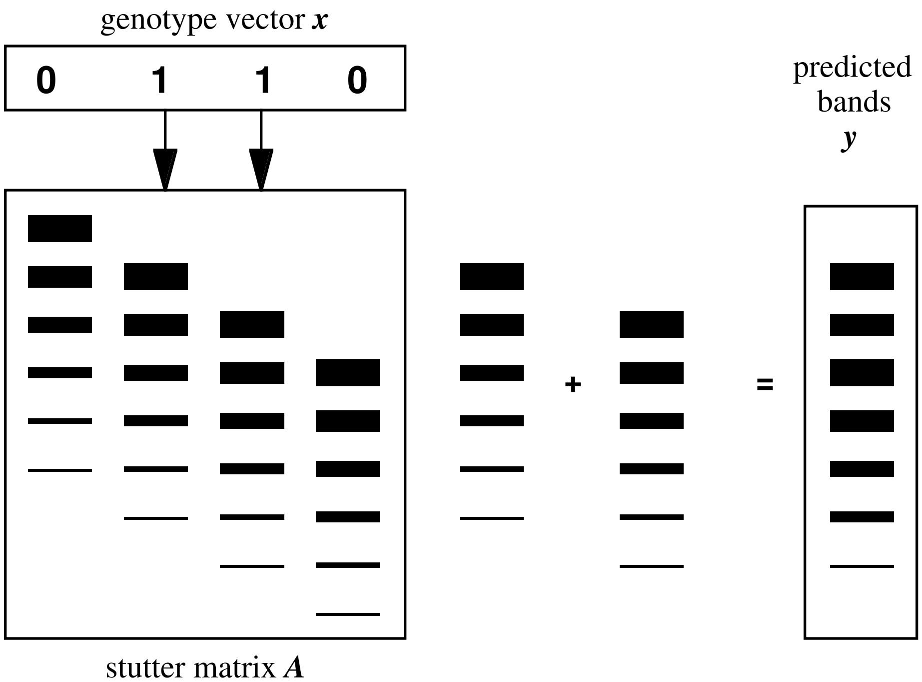


Figure 6.4. Convolution of stutter patterns. The action of the stutter matrix A against two alleles (encoded as "1"s in the genotype vector x) additively superimposes the corresponding PCR amplifier patterns, predicting the data vector data vector y observed on the gel.

6.2. Genotyping Microsatellites by Deconvolution (GMBD)

Our task in genotyping is to recover the genotype vector x from the observed gel data y. Let us assume for now that we also have access to a stutter matrix A for each microsatellite marker to be genotyped. Under our convolution model, we can use *deconvolution* to systematically remove the PCR stutter to recover the true alleles for each genotyping experiment (Figure 6.5). Mathematically, this amounts to a matrix-vector "left division" by the matrix A:

$$x = A \backslash y$$

Here, left division (the MATLAB[16] "\" operation) means solving for x in the least squares sense for the over- or under-determined system of equations $Ax = y$. It can be roughly interpreted as $A^{-1}y$.

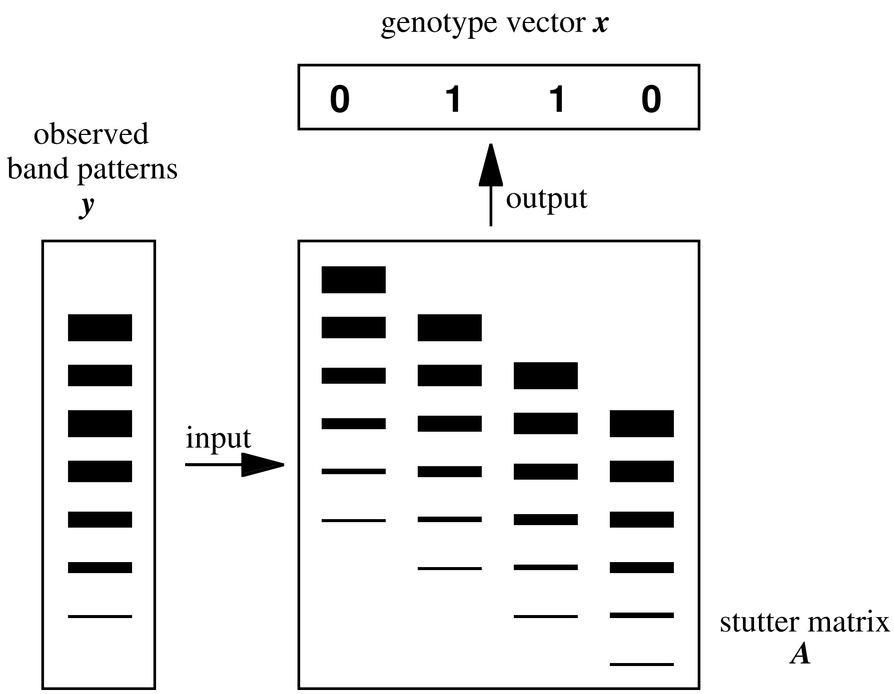


Figure 6.5. The GMBD procedure. The stutter matrix A is divided by the observed (input) data vector y to compute a best-fit (output) genotype vector x.

The stutter matrix A for a marker can be determined from a set of known reference (column) genotype vectors X and the corresponding set of experimentally observed (column) data vectors Y, by a simple extension of our original PCR stutter convolution model $y = Ax$ to the matrix equation:

$$Y = AX$$

where Y, A, and X are all matrices. The stutter pattern matrix A can then be computed using matrix "right division" of the (over or under-determined) linear system:

$$A = Y/X$$

[16]MATLAB is the programming language in which we implemented our automated microsatellite genotyping system FAST-MAP (see Chapter 8).

With the MATLAB "/" operator (matrix right division), Y/X is roughly interpreted as YX^{-1}. Since the stutter patterns are only reproducible under identical experimental conditions, the stutter matrix A must be re-computed should the marker's PCR conditions be changed.

6.3. Relative Amplification

In addition to the stutter artifact, there is a second complication associated with microsatellite PCR amplification. During PCR, the different fragments present in a DNA sample compete with one another for amplification by the polymerase enzyme. An allele of a small molecular size tends to be amplified more efficiently than an allele of a larger size, since there are relatively fewer nucleotides that to duplicate. In a heterozygotic genotype, this results in a *relative amplification* effect between the two alleles (including their stutter bands). Generally, the wider the two true alleles are apart, the higher is the degree of relative amplification between them, as illustrated in Figure 6.6.

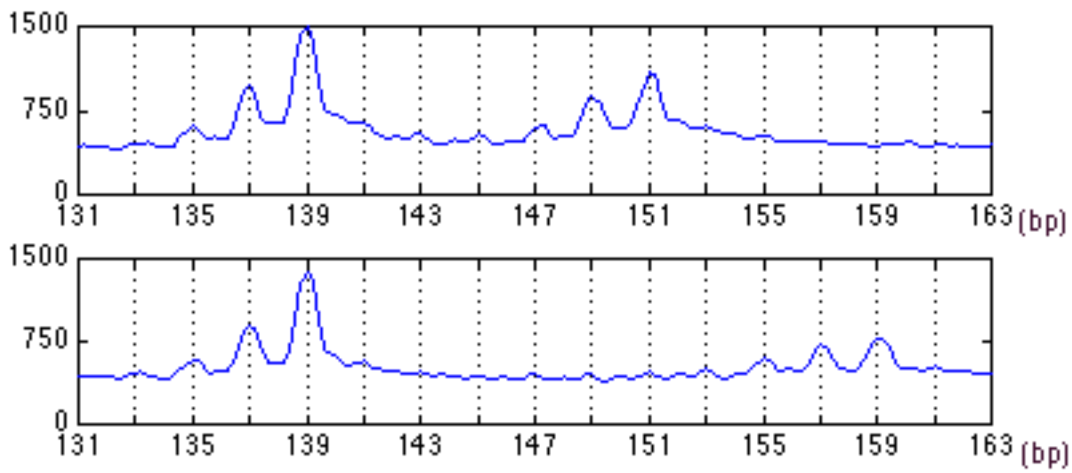


Figure 6.6. Relative amplification of widely separated alleles. Shown are two examples from marker D22S283. Shown here are two genotyping experiments of D22S283 from the same gel. In the top pane, the true alleles are 139 bp and 151 bp. The stutter bands for the allele at 151 bp are slightly less amplified than those for the 139 bp allele. In the bottom pane, the true alleles are even further apart, at 139 bp and 159 bp. The corresponding relative amplification effect between the alleles is even more pronounced.

To model the relative amplification effect in our convolution model, we generalize the genotype vector x from a (0,1)-vector to a real-valued vector. This extension enables the model to account for genotypes with alleles that amplify in a non 1:1 ratio. Instead of being merely a Boolean vector indicating the presence or absence of particular alleles, the genotype vector x is now a weighted vector that records the alleles' relative amplifications. The vector sum for x should still be normalized to the total number of alleles present in the genotype. This means that for diploids, x would sum to 2.

6.4. Deconvolution Methods

For solving the matrix division problem in microsatellite genotyping, we have adapted various deconvolution methods from diverse areas, including signal processing, matrix computation, and artificial intelligence. The algorithms can be organized into two main categories:

(i) Linear shift-invariant algorithms that use a single size-independent PCR stutter pattern vector a *for all alleles.*

- POLY: Polynomial division is used to divide the stutter vector a by the data vector y in order to estimate the genotype vector x.
- FFT: The Fast Fourier Transform is used to deconvolve the data vector y with the stutter vector a to recover the genotype vector x. This is done by dividing the FFT of y by the FFT of a, and then recovering the deconvolved vector x by an inverse FFT.
- WIENER: The FFT method is used with additional Wiener filtering (Press *et al.*, 1992) to filter out possible noise from the observed data. A noise filter Φ, derived directly from the data, is used. Here, we assume that noise arises from low-power interference and does not exceed 15% of the observed data,

  $$|\Phi(f)| = \min(s_p,\ 0.15S_p),$$

  where s_p and S_p are the minimum and maximum values of the data's power spectrum respectively.

(ii) Allele-dependent algorithms that use a marker's size-dependent PCR stutter patterns, recorded in a matrix A.

- GAUSS: Gaussian elimination. Starting from the rightmost band (largest allele size), successively subtract off each allele's stutter pattern.
- SVD: Singular value decomposition. Use SVD to essentially invert the stutter matrix A and apply the inverse matrix to the data vector y to recover the genotype vector x.
- ENUM: Enumeration. Directly enumerate (by exhaustive search) all feasible genotype vectors x to find the one having the least error between observed data vector y and predicted pattern vector Ax.

The linear shift-invariant algorithms POLY, FFT, and WIENER are conventional signal-processing algorithms that assume (usually incorrectly) that the stutter pattern does not vary with allele size. The allele-dependent algorithms GAUSS, SVD, and ENUM, on the other hand, are specifically designed to account for stutter patterns that vary with allele size. Thus, we would expect the algorithms from the latter category to outperform those from the former category in analyzing actual data.

A comparative study

To decide on the deconvolution methods to use for GMBD in our genotyping system, we conducted a preliminary comparative study (Perlin *et al.*, 1995) on all six algorithms with simulated data (Appendix A). As a measure of the effectiveness of an algorithm in removing the PCR stutter artifacts, we compared the methods based on the extent with which they *re-center* the allele distribution onto the correct genotype (see Figure 6.6). Note that we did not include relative amplification in our 1995 simulations, which focused on pure stutter artifacts.

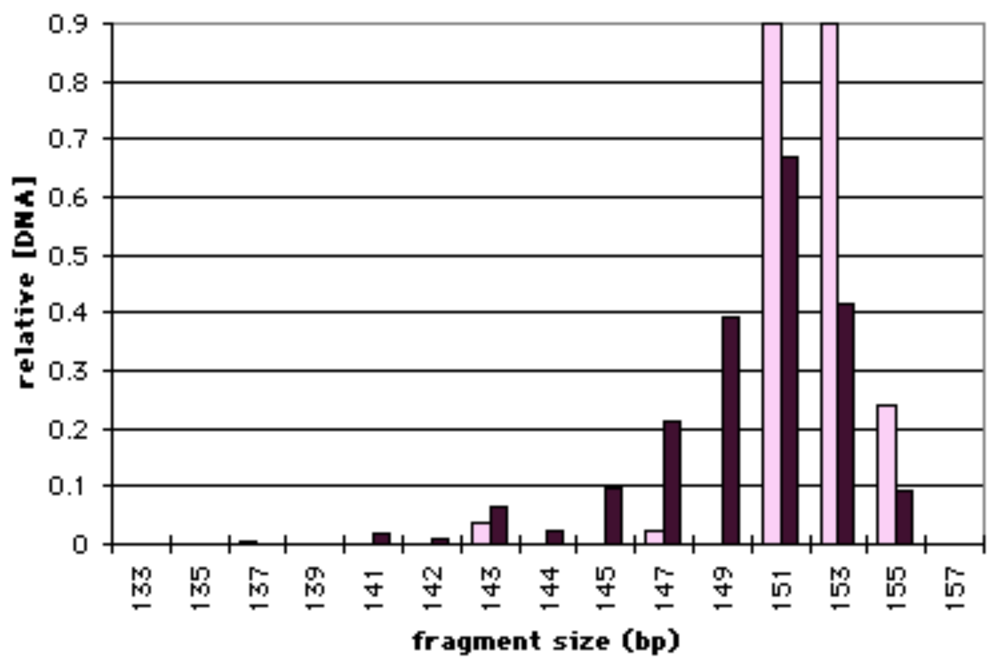


Figure 6.6. Effect of deconvolution in removing PCR stutter artifact for alleles differing by 2 bp. Shown are the allele distribution of the uncorrected data *y* without deconvolution (blackened bars), and the distribution of the corrected data *x* with deconvolution (unblackened bars). The allele distributions are normalized to sum to 2, i.e., the number of alleles present. Note that the distribution *x* corrected by deconvolution (in this case, SVD) is largely centered on the correct two alleles at 151 bp and 153 bp. The data shown is from microsatellite marker D22S283, size separated in one lane of a 34-lane ABI/377 gel.

Table 6.1 shows the results of our comparative study. It confirms that allele-dependent algorithms are more effective than allele-independent deconvolution algorithms, especially in the presence of severe stutter. In particular, ENUM performed perfectly in our study, since it always re-centers the allele mass to exactly two (correct) alleles. SVD and GAUSS were also highly effective in re-centering the allele distribution. Since SVD and GAUSS (unlike ENUM) do not rely on any assumptions on the number of discrete alleles present in the PCR mixture, they are also useful for detecting abnormalities or contamination in the DNA samples. For example, if there had been three alleles in the genotype, SVD would probably have detected this by returning a deconvolved vector x having three non-zero entries, each with an allele mass of approximately 0.67 units.

	Noise	
	0%	10%
Moderate stutter (5 bands):		
Input y	0.635	0.635
POLY	0.991	0.950
FFT	0.990	0.948
WIENER	0.959	0.920
GAUSS	1.000	0.981
SVD	1.000	0.955
ENUM	1.000	1.000
Severe stutter (10 bands):		
Input y	0.468	0.469
POLY	0.977	0.921
FFT	0.977	0.919
WIENER	0.896	0.852
GAUSS	1.000	0.979
SVD	1.000	0.938
ENUM	1.000	1.000

Table 6.1. The re-centering effects, measured as the fraction of allele distribution re-centered on the correct genotype, for the six deconvolution algorithms on simulated data. The simulation studies were conducted with 300 simulated genotypes of closely spaced alleles that were separated by 0 to 3 dinucleotide repeat units.

6.5. Processes in GMBD

The transformation in allele determination is summarized as:

$$\gamma : \text{ quantitated stutter data } \rightarrow \text{Genotypes}$$

To perform this transformation, we need information about the stutter patterns as well as the relative amplifications of the microsatellite marker. This knowledge is acquired from prior data analysis of the marker, and is organized in a systematic marker library for efficient computer retrieval and maintenance. The GMBD transformation γ therefore involves two consecutive processes:

- Marker library construction
  $$\gamma_1 : \text{ stutter data + [known genotypes]} \rightarrow \text{pattern matrix } A \text{ + ratio table } \rho$$
- Genotyping by deconvolution
  $$\gamma_2 : \text{ stutter data + pattern matrix } A \text{ + ratio table } \rho \rightarrow \text{genotypes}$$

Before we proceed to describe the algorithms for γ_1 and γ_2 in detail, to better understand the processes involved in GMBD, we first examine an actual program trace of our genotyping system on a marker. In the program trace below, the gel "`3/19/97_rp_Gel`" is a 34-well ABI/377 gel, 16 lanes of which (lanes 11 through 26) were loaded with marker panel "`panel3`". Out of the 11 markers in the panel, we focus on marker "`D22S283`", a dinucleotide repeat marker with an allele window of 135 to 165 bp (in fact, D22S283 has been the example marker in this chapter). Here is the trace of our genotyping system analyzing D22S283 on a Macintosh PowerBook 3400c:

```
Marker D22S283  (5 of 11 markers in panel3): [22-Nov-97 17:17:43]
------

Scanning 3/19/97_rp_Gel for marker windows..................Done.
Scanning 3/19/97_rp_Gel for marker bands....................Done.
Binning marker bands........................................Done.
Quantitating D22S283 3/19/97_rp_Gel lane 11.....a.b.a.b..Done.
Quantitating D22S283 3/19/97_rp_Gel lane 12.....a.b..Done.
Quantitating D22S283 3/19/97_rp_Gel lane 13.....a.b.a.b.a.b..Done.
Quantitating D22S283 3/19/97_rp_Gel lane 14.....a.b..Done.
Quantitating D22S283 3/19/97_rp_Gel lane 15.....a.b.a.b.a.b..Done.
Quantitating D22S283 3/19/97_rp_Gel lane 16.....a.b.a.b..Done.
Quantitating D22S283 3/19/97_rp_Gel lane 17.....a.b.a.b..Done.
Quantitating D22S283 3/19/97_rp_Gel lane 18.....a.b.a.b.a.b..Done.
Quantitating D22S283 3/19/97_rp_Gel lane 19.....a.b.a.b..Done.
Quantitating D22S283 3/19/97_rp_Gel lane 20.....a.b..Done.
Quantitating D22S283 3/19/97_rp_Gel lane 21.....a.b..Done.
```

```
Quantitating D22S283 3/19/97_rp_Gel lane 22.....a.b.a.b..Done.
Quantitating D22S283 3/19/97_rp_Gel lane 23.....a.b..Done.
Quantitating D22S283 3/19/97_rp_Gel lane 24.....a.b.a.b..Done.
Quantitating D22S283 3/19/97_rp_Gel lane 25.....a.b.a.b..Done.
Quantitating D22S283 3/19/97_rp_Gel lane 26.....a.b.a.b..Done.
Estimating genotypes...........................................Done.
Using the following 14 estimated genotypes to construct initial
stutter library for D22S283:
   3/19/97_rp_Gel Lane 11 : <151, 159> (qual = 0.80)
   3/19/97_rp_Gel Lane 12 : <145, 151> (qual = 0.81)
   3/19/97_rp_Gel Lane 13 : <145, 159> (qual = 0.83)
   3/19/97_rp_Gel Lane 14 : <151, 151> (qual = 0.93)
   3/19/97_rp_Gel Lane 15 : <145, 159> (qual = 0.76)
   3/19/97_rp_Gel Lane 17 : <139, 151> (qual = 0.85)
   3/19/97_rp_Gel Lane 18 : <151, 153> (qual = 0.82)
   3/19/97_rp_Gel Lane 19 : <147, 151> (qual = 0.85)
   3/19/97_rp_Gel Lane 20 : <151, 153> (qual = 0.85)
   3/19/97_rp_Gel Lane 21 : <139, 139> (qual = 0.85)
   3/19/97_rp_Gel Lane 22 : <149, 153> (qual = 0.80)
   3/19/97_rp_Gel Lane 23 : <145, 153> (qual = 0.84)
   3/19/97_rp_Gel Lane 25 : <145, 149> (qual = 0.89)
   3/19/97_rp_Gel Lane 26 : <143, 153> (qual = 0.79)
Bootstrapping initial library A using estimated
genotypes...........:......................:.................:..
................:.........:..........:........:........Done.
Genotyping D22S283 3/19/97_rp_Gel lane 11....Done.
Genotyping D22S283 3/19/97_rp_Gel lane 12....Done.
Genotyping D22S283 3/19/97_rp_Gel lane 13....Done.
Genotyping D22S283 3/19/97_rp_Gel lane 14....Done.
Genotyping D22S283 3/19/97_rp_Gel lane 15....Done.
Genotyping D22S283 3/19/97_rp_Gel lane 16....Done.
Genotyping D22S283 3/19/97_rp_Gel lane 17....Done.
Genotyping D22S283 3/19/97_rp_Gel lane 18....Done.
Genotyping D22S283 3/19/97_rp_Gel lane 19....Done.
Genotyping D22S283 3/19/97_rp_Gel lane 20....Done.
Genotyping D22S283 3/19/97_rp_Gel lane 21....Done.
Genotyping D22S283 3/19/97_rp_Gel lane 22....Done.
Genotyping D22S283 3/19/97_rp_Gel lane 23....Done.
Genotyping D22S283 3/19/97_rp_Gel lane 24....Done.
Genotyping D22S283 3/19/97_rp_Gel lane 25....Done.
Genotyping D22S283 3/19/97_rp_Gel lane 26....Done.
Updating stutter library.......Done.
Done processing marker D22S283 at 22-Nov-97 17:51:45.
```

First, the computer scans the gel for regions ("marker windows") containing D22S283's genotyping data for analysis (`"Scanning for marker windows...."`). To do so, the computer uses the MW sizing grid information and marker information such as the expected allele size window, as well as gel information such as the layout of the loaded lanes. After the computer has located marker windows in each lane's electropherogram, the computer scans for marker bands in the data windows (`"Scanning for marker bands...."`). The marker bands are then aligned and binned using our

STUTTER_CRAWL algorithm to refine the MW sizing grid to the marker's actual allelic ladder ("`Binning marker bands....`").

Using the binned allelic ladder as a sizing reference, the computer then quantitates the marker bands in each of the genotyping experiment ("`Quantitating lane 11.....a.b.a.b...`") with the QUANTITATE_BAND algorithm. To obtain an quantitated fit that is as close to the actual data bands as possible, the computer iteratively refines its data peak parameters (for the Gauss-Runga peak model) using least-square methods. Each "`a.b`" pair in the program trace indicates one such iteration. As shown in the example, only two or so iterations are generally needed to reach convergence for clean data.

Since we are analyzing marker D22S283 for the first time, the computer does not have any previous marker library that it can use for allele determination. Therefore, the computer bootstraps by estimating the alleles ("`Estimating genotypes....`"), and then constructs a marker library from the estimated genotypes ("`Bootstrapping initial library....`"). To obtain an initial marker library that is as accurate as possible, the computer iteratively refines the marker library constructed during bootstrapping process. Each refining iteration is indicated with a colon ":" in the program trace.

After constructing a marker library for D22S283, the computer then proceeds to the allele determination step ("`Genotyping lane 11....`") using the marker library. When the computer has completed calling the alleles on all the genotyping experiments, it learns from the data by refining D22S283's marker library using the final allele calls ("`Updating stutter library....`"). The computer automatically augments its internal knowledge base with new stutter pattern and relative amplification information from the data. In this way, the computer improves its performance over time as it analyzes more data for marker D22S283.

6.6. Algorithms: Marker Library Construction

In this section, we describe the algorithms for constructing marker libraries. A marker library stores the relevant characteristics of the marker's expected allele stutter patterns that the computer can use in genotyping by deconvolution. The library consists of two components: A, a stutter pattern matrix, and ρ, a relative amplification ratio table.

Preferably, both A and ρ should be determined with known reference genotypes. If not (as is often the case), they can be compiled from a bootstrapping technique.

As previously described, we construct the stutter pattern matrix A using matrix "right division" of the linear system:

$$A = Y/X$$

where Y contains quantitated stutter data and X contains known or computed genotypes. Both X and Y are matrices, with each column representing respectively the genotype or the data of a genotyping experiment. The relative amplification ratios are computed from the quantitated data and their known genotypes, using the expected stutter patterns in A.

Box 6.1 describes the overall algorithm for constructing a marker library. Details on the actual structures of the marker libraries used in our genotyping system are described in Appendix C.

Algorithm: CONSTRUCT_LIBRARY

Step 1: Construct stutter matrix A.

Case 1: If both the stutter data Y and the reference genotypes X are given, construct A using algorithm CONSTRUCT_A.

Case 2: If the reference genotypes X are not available, construct A on only the stutter data Y using algorithm BOOTSTRAP_A.

Step 2: Construct ratio table ρ.

Construct the amplification ratio table ρ using algorithm CONSTRUCT_ρ.

Box 6.1. CONSTRUCT_LIBRARY: An algorithm for constructing a marker library.

6.6.1. Stutter matrix construction

The stutter matrix A can be constructed from the quantitated data Y and the associated known genotypes X (weighted by their relative amplifications) by solving the under- or over-determined linear system $Y = AX$. If X does not cover all the possible marker alleles, we fill in the stutter patterns for the absentee alleles with copies of stutter patterns from the nearest alleles. As more data become available, we refine the stutter matrix by replacing the copied patterns with the actual stutter patterns from the new data (using the REFINE_A algorithm).

Box 6.2 below describes the CONSTRUCT_A algorithm for constructing a stutter matrix A from a given set of quantitated stutter data Y and their corresponding genotypes X.

Algorithm: CONSTRUCT_A

Step 1: Normalize columns in X and Y.

Normalize the columns in X and Y such that every column sums to the expected number of alleles in the genotype represented by that column. Usually, individual reference genotypes are used[17], so the column sums are 2 (for diploids).

Step 2: Numerically compute A.

Given Y and X, compute an initial A using matrix "division":

$$A = Y/X$$

The symbol "/" represents solving the linear system $AX=Y$ with numerical least-square algorithms (e.g. Gaussian elimination, Cholesky factorization, or QR-factorization (Press *et al.*, 1992)). Many numerical programming systems support this operation as part of the language. For example, in Matlab, we simply type A = Y/X.

[17]In principle, we can also used *pooled* genotype for constructing A. That is, we can pool together individual DNA samples in a single genotyping experiment, as long as we normalize the corresponding columns in X and Y to the appropriate allele sums.

Step 3: Normalize columns in A.

Depending on the algorithms used in the matrix division, the resulting matrix A may contain negative entries. Zero out any non-positive entries in the stutter matrix, and normalize each column in A to sum to 1, since each column contains the stutter pattern for a single allele.

For each row i in X that has no non-zero entries, zero out the entire i-th column of matrix A, if it is not already all zeros.

Step 4: Fill in columns in A for absentee alleles.

For each all-zero column in A, replace with a copy of the stutter pattern from the closest non-zero column.

Box 6.2. CONSTRUCT_A: An algorithm for constructing a stutter pattern matrix from reference genotype data.

Bootstrapping

The reference genotypes X are often not available for stutter matrix construction when we first genotype a marker (either a marker from a new marker panel or an old marker under a new experimental condition). In such situations, we must bootstrap the construction of the marker library without any reference genotypes.

First, we generate approximate reference genotypes from the data by re-using an old stutter matrix from a similar marker (or simply a generic stutter matrix) and use our deconvolution methods to construct an initial X. Then, using X on Y, we construct an estimated stutter matrix A. As with all our algorithms, as a final step, we iteratively refine A by minimizing the least-square error between the predicted patterns AX and the observed data Y. Box 6.4 below describes this bootstrapping process in greater detail.

Algorithm: BOOTSTRAP_A

Step 1: Construct an initial A.

First, either obtain an initial A from a previously compiled marker that is similar to the new marker, or construct A by filling each column with a generic stutter pattern.

Step 2: Construct an initial X by SVD.

Given A and Y (normalized), compute a corresponding X using the SVD deconvolution method. Normalize the entries in X so that each column contains at most two non-zero entries, and that each column sums to 2.

Step 3: Iteratively refine A.

Using algorithm REFINE_A, refine A iteratively.

Box 6.3. BOOTSTRAP_A: An algorithm for constructing a stutter matrix without using reference genotypes.

Refining

Some of the possible alleles for a marker may not be included in the initial reference genotypes for constructing the stutter matrix. Then, the computer estimates the stutter patterns for these alleles by using the stutter patterns from the nearest reference alleles. When data for these alleles later become available, the computer should automatically update its stutter matrix A with the actual stutter patterns. This update allows the computer to learn and improve over time and data, as well as gracefully adjust to changes in the marker's behavior (by biasing towards the more recent data, if necessary).

Box 6.5 shows our algorithm for refining the stutter matrix A from new data in X and Y.

Algorithm: REFINE_A

Step 1: Construct a new A .

Using algorithm CONSTRUCT_A, construct a new stutter matrix A_{new} from X and Y.

Step 2: Compare A_{new} against A.

(a) Using algorithm SVD_DECONVOLVE on A_{new} and Y, compute new genotypes in X_{new}.

(b) Compute err_{new}, the total sum of squares error between the matrix product $A_{new}X_{new}$ and the observed data Y.

Step 3: Iteratively refine.

If err_{new} is less than the original sum of squares error between AX and Y, replace A with A_{new} , and repeat Steps 1 and 2.

Box 6.4. REFINE_A: An algorithm for incrementally improving the stutter matrix A.

6.6.2. Relative amplification ratio table construction

We define the relative amplification ratio for a genotype as the ratio of the total DNA (including stutter bands) from the smaller allele to the total DNA from the larger allele. In other words, the relative amplification of a genotype is the ratio in a weighted genotype vector x between the vector value of the smaller allele to that of the larger allele. We store a *range* of possible amplification ratios for each genotype in a marker's amplification ratio table, since the relative amplification ratios vary slightly from one genotyping experiment to another (even if they have the same genotype).

To compute the entries in the relative amplification table ρ, we use the same reference genotype data X and Y used for constructing the stutter pattern matrix A. For each genotype in *X,* we compute the best relative amplification ratio for the allele pair that minimizes the least-square error between the observed data vector in *Y* and the expected stutter pattern re-constructed in *AX.*

Since the reference data typically does not include all possible marker alleles, there is usually not reference data for *every* possible pair of alleles. To compute the relative amplification ratios, we could fill in the missing amplification ratios either by interpolation or using the relative amplification ratio values from the nearest reference allele pairs, as was done for the absentee alleles in stutter matrix *A*. However, we have observed that amplification ratios depend primarily on allele size *difference* rather than actual allelic values. We therefore make the following assumption:

$$\rho\,(a_1, a_2) \cong \rho\,(b_1, b_2) \text{ where } |a_1 - a_2| = |b_1 - b_2|$$

This simplifying assumption allows us to compress ρ from an $O(n^2)$ table for an n-allele marker to an $O(n)$ table indexed by the allelic differences. Figure 6.7 shows an example of a compressed amplification ratio table for D22S283. With the simplified ρ, there are now only $O(n)$ relative amplification ratios that have to be determined. Moreover, any missing values can easily be linearly interpolated (a partially-filled two-dimensional table ρ is more complex to interpolate). Although we could expand ρ out to its full-blown n^2-entry table as more data become available, our experience has been that the compressed version of ρ is adequate for GMBD computations.

In Box 6.5, we describe CONSTRUCT_ρ , our algorithm for computing the relative amplification ratios from the quantitated stutter data *Y,* the reference genotypes *X,* and the stutter matrix *A* .

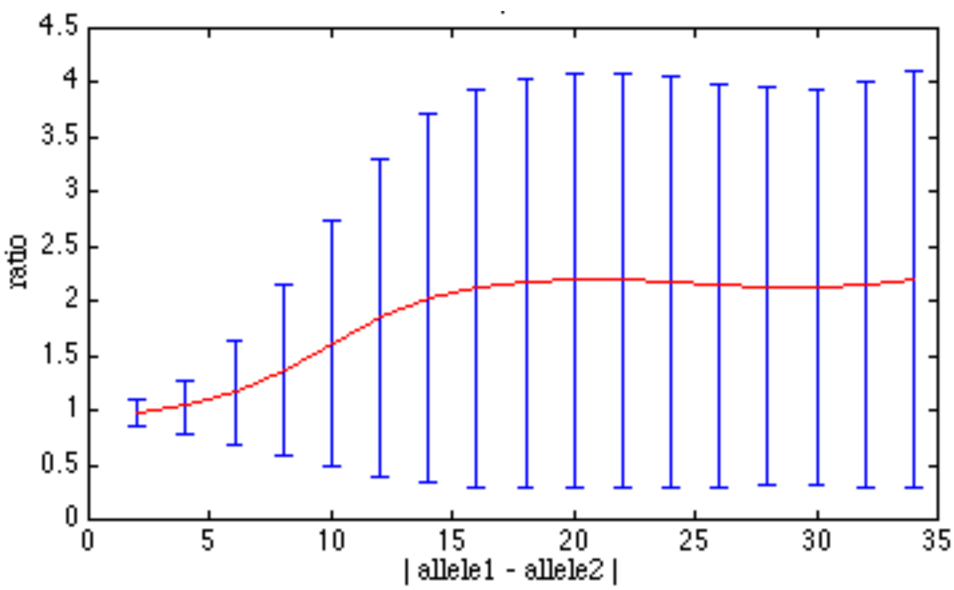


Figure 6.7. Relative amplification ratio table. Shown is the compressed form of the relative amplification ratio table for TET-labeled dinucleotide marker D22S283 (allele range 135-165 bp), as compiled from an ABI 377 gel running 16 lanes of marker data. The vertical error bars show an expected range of relative amplification ratios for each allele pair, while the plotted line shows the mean amplification ratios. (The stutter matrix for D22S283 was shown in Figure 6.3.)

Algorithm: CONSTRUCT_ρ

Step 1: Sort the known genotypes.

Group the columns in X according to their allele size differences.

Step 2: Compute amplification ratios for each observed allele size difference.

(a) For each column in X that has (non-zero) allele size difference δ, compute a new genotype vector (say, x_j). Compute x_j using the SVD deconvolution method with stutter matrix A and the data vector y_j from the corresponding column in Y. The ratio of the two genotype entries in x_j gives the relative amplification ratio, υ_j , for this reference genotype.

(b) Minimally adjust each υ_j so that the predicted vector Ax_j adjusted by υ_j has least sum of squares error with the observed data vector y_j .

(c) Let μ_δ be the mean of the computed υ_j 's, and σ_δ the corresponding standard deviation. Set $\rho(\delta) \leftarrow [\mu_\delta - \sigma_\delta, \mu_\delta + \sigma_\delta]$.

Step 3: Interpolate amplification ratios for the unobserved allele size differences.

Fill any empty entries in ρ using spline interpolation.

Box 6.5. CONSTRUCT_ρ: An algorithm for constructing a relative amplification ratio table for a marker from its quantitated stutter data Y, reference genotypes X, and the stutter matrix A.

6.6.3. Example

Let us step through the marker library construction for the marker D22S283. We showed the actual program trace for the entire allele determination process for D22S283 in Section 6.5. Here, we focus on the construction of a marker library for D22S283. For simplicity, we use a smaller example with only 8 reference lanes.

To construct a marker library for D22S283, we need both the quantitated data Y and the corresponding reference genotypes X. First, we write down the band quantitations from each lane's electropherogram in the columns of data matrix Y. Each column of Y is normalized to sum to 2, the expected number of alleles present in the genotype:

Data matrix Y

(bp)	lane 11	lane 12	lane 13	lane 14	lane 15	lane 16	lane 17	lane 18
131	0	0	0	0	0	0	0	0
133	0	0	0	0	0	0.0158	0	0
135	0	0	0	0	0.01	0.1010	0	0
137	0	0.0235	0	0	0.027	0.3271	0	0
139	0	0.0499	0.0109	0	0.0477	0.6166	0	0.0331
141	0.0263	0.1428	0.1317	0	0.1319	0	0.0205	0.0531
143	0.0485	0.3436	0.3781	0.0627	0.3341	0.056	0.0660	0.1413
145	0.1044	0.5710	0.6112	0.1143	0.5314	0.0523	0.1004	0.3208
147	0.1794	0.1986	0	0.2822	0.0726	0.1110	0.2141	0.5948
149	0.3399	0.2634	0.037	0.6029	0.0532	0.2586	0.4001	0.3037
151	0.5230	0.3762	0.0451	0.8264	0.0686	0.3757	0.6807	0.3986
153	0	0	0.0594	0.1115	0.0836	0.0859	0.4235	0.0978
155	0.1434	0.0310	0.1058	0	0.1266	0	0.0947	0.0568
157	0.2417	0	0.2411	0	0.206	0	0	0
159	0.3337	0	0.3273	0	0.2581	0	0	0
161	0.0597	0	0.0524	0	0.0492	0	0	0
total	2	2	2	2	2	2	2	2

To construct the stutter matrix A, we also need a corresponding reference genotype matrix X. This is the first time analyzing marker D22S283, so there are no previously analyzed genotypes available, nor do we know the actual genotypes for Y. Thus, we bootstrap by using GMBD on the quantitated data with a *generic* marker library to compute a set of estimated genotypes for Y as the initial X. Since D22S283 is a dinucleotide repeat marker, we use a typical stutter matrix A containing a generic stutter pattern (for dinucleotide repeat markers) of three data bands with relative DNA concentrations 0.571, 0.286, and 0.143, as shown in Figure 6.8. In Figure 6.9, we show the default stutter matrix A and the default relative amplification ratio table ρ for bootstrapping dinucleotide marker library construction.

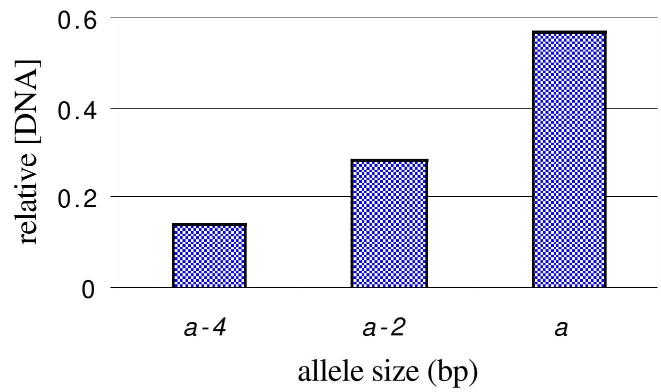


Figure 6.8. A generic stutter pattern for any allele *a* of a dinucleotide repeat marker. This generic stutter pattern is used in bootstrapping dinucleotide marker library construction.

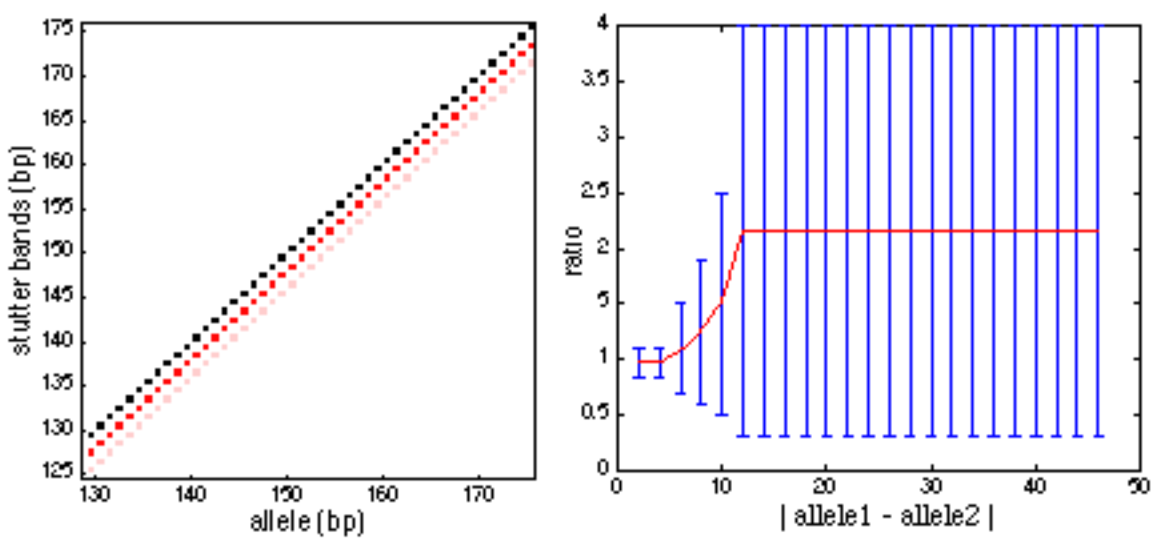


Figure 6.9. A default dinucleotide repeat marker library for bootstrapping. Shown on the left is the default stutter matrix *A,* and on the right is the default relative amplification ratio table (compressed form) ρ.

Using the default dinucleotide marker library *A* and ρ on the quantitated data matrix *Y,* we estimate the initial reference genotypes *X* using the SVD algorithm (described in the next section). Here is a program trace:

```
Estimating genotypes........................................Done.
Using the following 8 estimated genotypes to construct initial
stutter library for d22s283:
   3/19/97_rp_Gel Lane 11 : <151, 159> (qual = 0.80)
   3/19/97_rp_Gel Lane 12 : <145, 151> (qual = 0.81)
   3/19/97_rp_Gel Lane 13 : <145, 159> (qual = 0.83)
   3/19/97_rp_Gel Lane 14 : <151, 151> (qual = 0.93)
   3/19/97_rp_Gel Lane 15 : <145, 159> (qual = 0.76)
   3/19/97_rp_Gel Lane 17 : <139, 151> (qual = 0.85)
   3/19/97_rp_Gel Lane 18 : <151, 153> (qual = 0.82)
   3/19/97_rp_Gel Lane 19 : <147, 151> (qual = 0.85)
```

From the 8 estimated genotypes, we construct a reference genotype matrix X_{ref}:

<u>reference genotype matrix X_{ref}</u>

(bp)	lane 11	lane 12	lane 13	lane 14	lane 15	lane 16	lane 17	lane 18
139	0	0	0	0	0	1	0	0
141	0	0	0	0	0	0	0	0
143	0	0	0	0	0	0	0	0
145	0	1	1	0	1	0	0	0
147	0	0	0	0	0	0	0	1
149	0	0	0	0	0	0	0	0
151	1	1	0	2	0	1	1	1
153	0	0	0	0	0	0	1	0
155	0	0	0	0	0	0	0	0
157	0	0	0	0	0	0	0	0
159	1	0	1	0	1	0	0	0
total	2	2	2	2	2	2	2	2

For each genotype in X_{ref}, we adjust the relative amplification between the alleles to minimize the sum of squares errors between the predicted patterns AX_{ref} and the observed patterns Y. The reference genotype matrix X_{ref} is then updated with these amplification ratios, giving a weighted genotype X:

<u>weighted genotype matrix X</u>

(bp)	lane 11	lane 12	lane 13	lane 14	lane 15	lane 16	lane 17	lane 18
139	0	0	0	0	0	1.04	0	0
141	0	0	0	0	0	0	0	0
143	0	0	0	0	0	0	0	0
145	0	1.08	1.16	0	1.17	0	0	0
147	0	0	0	0	0	0	0	1.26
149	0	0	0	0	0	0	0	0
151	1.15	0.92	0	2.00	0	0.96	1.10	0.74
153	0	0	0	0	0	0	0.90	0
155	0	0	0	0	0	0	0	0
157	0	0	0	0	0	0	0	0
159	0.85	0	0.84	0	0.83	0	0	0
total	2	2	2	2	2	2	2	2

With the weighted genotype matrix X and the quantitated data matrix Y, we solve for A in the linear system $AX = Y$. Here is the initial solution from Matlab's `"A = Y/X"` operation:

initial stutter matrix from $A = Y/X$

bp	139	141	143	145	147	149	151	153	155	157	159
131	0	0	0	0	0	0	0	0	0	0	0
133	0.02	0	0	0	0	0	0	0	0	0	0
135	0.10	0	0	0	0	0	0	0	0	0	0
137	0.31	0	0	0.02	0	0	0	0	0	0	-0.01
139	0.59	0	0	0.04	0.02	0	0	0	0	0	-0.01
141	-0.01	0	0	0.11	0.04	0	0.01	0.01	0	0	0.01
143	0.03	0	0	0.29	0.09	0	0.03	0.04	0	0	0.02
145	-0.01	0	0	0.47	0.22	0	0.06	0.04	0	0	0.03
147	-0.03	0	0	0.04	0.38	0	0.15	0.06	0	0	-0.01
149	-0.02	0	0	0.01	0.07	0	0.29	0.09	0	0	0.02
151	-0.02	0	0	0	0.08	0	0.41	0.25	0	0	0.06
153	0.06	0	0	0.03	0.06	0	0.03	0.44	0	0	0.02
155	-0.01	0	0	0.01	0.04	0	0.01	0.10	0	0	0.14
157	0	0	0	-0.01	0	0	0	0	0	0	0.28
159	0	0	0	-0.01	0	0	0.01	-0.01	0	0	0.38
161	0	0	0	0	0	0	0	0	0	0	0.07
total	1	0	0	1	1	0	1	1	0	0	1

After removing the negative entries in *A*, and normalizing the values in each column to sum to 1 (for non-zero columns), we have:

<u>normalized *A*</u>

bp	139	141	143	145	147	149	151	153	155	157	159
131	0	0	0	0	0	0	0	0	0	0	0
133	0.01	0	0	0	0	0	0	0	0	0	0
135	0.10	0	0	0	0	0	0	0	0	0	0
137	0.31	0	0	0.02	0	0	0	0	0	0	0
139	0.58	0	0	0.04	0.03	0	0	0	0	0	0
141	0	0	0	0.12	0.05	0	0.01	0.02	0	0	0
143	0	0	0	0.31	0.12	0	0.03	0.04	0	0	0
145	0	0	0	0.50	0.29	0	0.06	0.04	0	0	0
147	0	0	0	0	0.51	0	0.16	0.06	0	0	0
149	0	0	0	0	0	0	0.31	0.10	0	0	0.03
151	0	0	0	0	0	0	0.43	0.28	0	0	0.07
153	0	0	0	0	0	0	0	0.47	0	0	0.02
155	0	0	0	0	0	0	0	0	0	0	0.16
157	0	0	0	0	0	0	0	0	0	0	0.31
159	0	0	0	0	0	0	0	0	0	0	0.42
161	0	0	0	0	0	0	0	0	0	0	0
total	1	0	0	1	1	0	1	1	0	0	1

Note that the columns for the alleles 141, 143, 149, 153, and 157 do not contain any stutter patterns. This is because these alleles were not included in the set of reference genotypes *X* used in constructing *A*. To form a complete stutter matrix *A*, we fill in the stutter patterns for these absentee alleles using stutter patterns from the nearest non-zero columns. Figure 6.10 shows both the initial stutter matrix *A*, and the stutter matrix with filled-in columns (italicized).

stutter matrix A with filled-in columns

bp	139	*141*	*143*	145	147	*149*	151	153	*155*	*157*	159
131	0	*0*	*0*	0	0	*0*	0	0	*0*	*0*	0
133	0.01	*0*	*0*	0	0	*0*	0	0	*0*	*0*	0
135	0.10	*0.01*	*0.02*	0	0	*0*	0	0	*0*	*0*	0
137	0.31	*0.10*	*0.04*	0.02	0	*0*	0	0	*0*	*0*	0
139	0.58	*0.31*	*0.12*	0.04	0.03	*0.01*	0	0	*0*	*0*	0
141	0	*0.58*	*0.31*	0.12	0.05	*0.03*	0.01	0.02	*0*	*0*	0
143	0	*0*	*0.50*	0.31	0.12	*0.06*	0.03	0.04	*0.02*	*0*	0
145	0	*0*	*0*	0.50	0.29	*0.16*	0.06	0.04	*0.04*	*0*	0
147	0	*0*	*0*	0	0.51	*0.31*	0.16	0.06	*0.04*	*0.03*	0
149	0	*0*	*0*	0	0	*0.43*	0.31	0.10	*0.06*	*0.07*	0.03
151	0	*0*	*0*	0	0	*0*	0.43	0.28	*0.10*	*0.02*	0.07
153	0	*0*	*0*	0	0	*0*	0	0.47	*0.28*	*0.16*	0.02
155	0	*0*	*0*	0	0	*0*	0	0	*0.47*	*0.31*	0.16
157	0	*0*	*0*	0	0	*0*	0	0	*0*	*0.42*	0.31
159	0	*0*	*0*	0	0	*0*	0	0	*0*	*0*	0.42
161	0	*0*	*0*	0	0	*0*	0	0	*0*	*0*	0
total	1	*1*	*1*	1	1	*1*	1	1	*1*	*1*	1

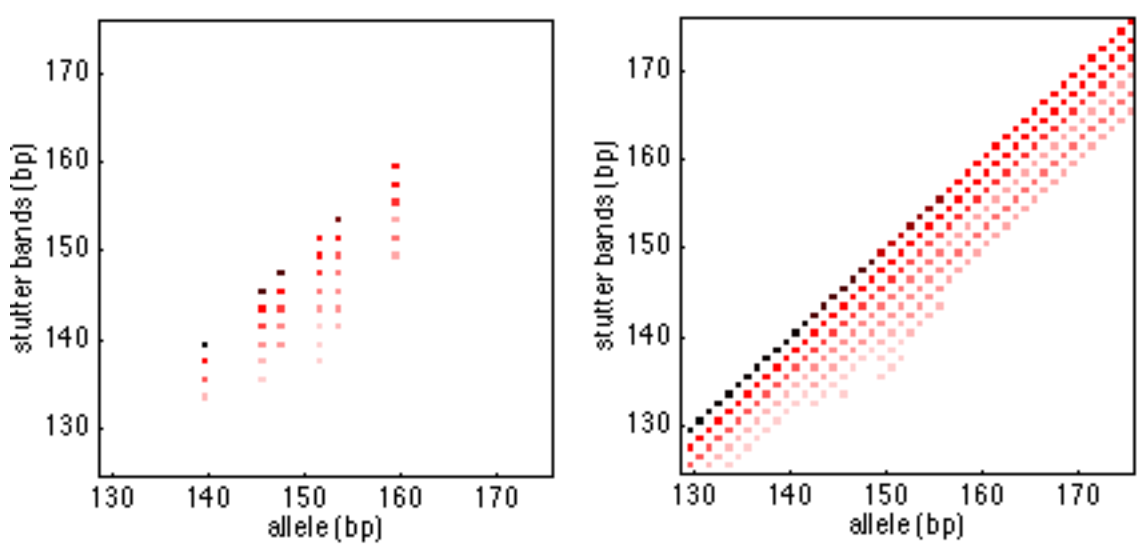


Figure 6.10. The computed stutter matrix A. Shown (on the left) is the initial stutter matrix from solving the linear system $AX = Y$, given X and Y. Because some of the marker alleles were absent in X, their stutter patterns have to be copied from the nearest reference alleles to construct a completely filled stutter matrix A, as shown (on the right).

After computing the stutter matrix *A,* the next step is to compute the relative amplification ratio table ρ from *A, X,* and *Y*. For each genotype in *X*, we compute the best relative amplification ratio between the allele pair in the genotype which minimizes the sum of squares error between the predicted patterns in *AX* and the actual observed data *Y*. We then collate the relative amplification ratios from genotypes that share the same allelic differences to compute the possible *range* of amplification ratios for that allelic difference. The missing values are interpolated. We show in Figure 6.11 the initial relative amplification ratio table ρ computed from our reference data *Y*.

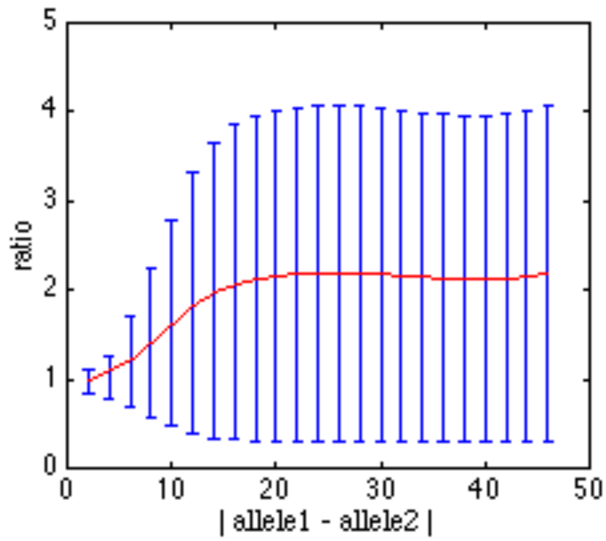


Figure 6.11. The initial relative amplification ratio table ρ for D22S283. Individual relative amplification ratios are computed from *X,* and *Y* using *A*. The ratios are then collated and interpolated to form a range of ratios for each allele difference value.

Finally, we repeat the construction process to iteratively refine the computed A and ρ for the library that best models the observed data Y in the least-squares sense. The resulting stutter matrix A is:

<u>stutter matrix *A* for D22S283</u>

bp	139	141	143	145	147	149	151	153	155	157	159
131	0.02	0	0	0	0	0	0	0	0	0	0
133	0.04	0.02	0.01	0	0	0	0	0	0	0	0
135	0.12	0.04	0.03	0.02	0	0	0	0	0	0	0
137	0.31	0.12	0.05	0.03	0.02	0.01	0	0	0	0	0
139	0.51	0.31	0.12	0.06	0.03	0.03	0.01	0	0	0	0
141	0	0.51	0.30	0.14	0.06	0.04	0.03	0	0	0	0
143	0	0	0.50	0.30	0.14	0.07	0.04	0.02	0	0	0
145	0	0	0	0.44	0.30	0.12	0.07	0.06	0.02	0	0
147	0	0	0	0	0.44	0.29	0.12	0.03	0.06	0.02	0
149	0	0	0	0	0	0.44	0.29	0.16	0.03	0.06	0.02
151	0	0	0	0	0	0	0.44	0.31	0.16	0.03	0.06
153	0	0	0	0	0	0	0	0.42	0.31	0.16	0.03
155	0	0	0	0	0	0	0	0	0.42	0.31	0.16
157	0	0	0	0	0	0	0	0	0	0.42	0.31
159	0	0	0	0	0	0	0	0	0	0	0.42
161	0	0	0	0	0	0	0	0	0	0	0
total	1	1	1	1	1	1	1	1	1	1	1

Figure 6.11 shows the final stutter pattern matrix A and relative amplification ratio table ρ for marker D22S283.

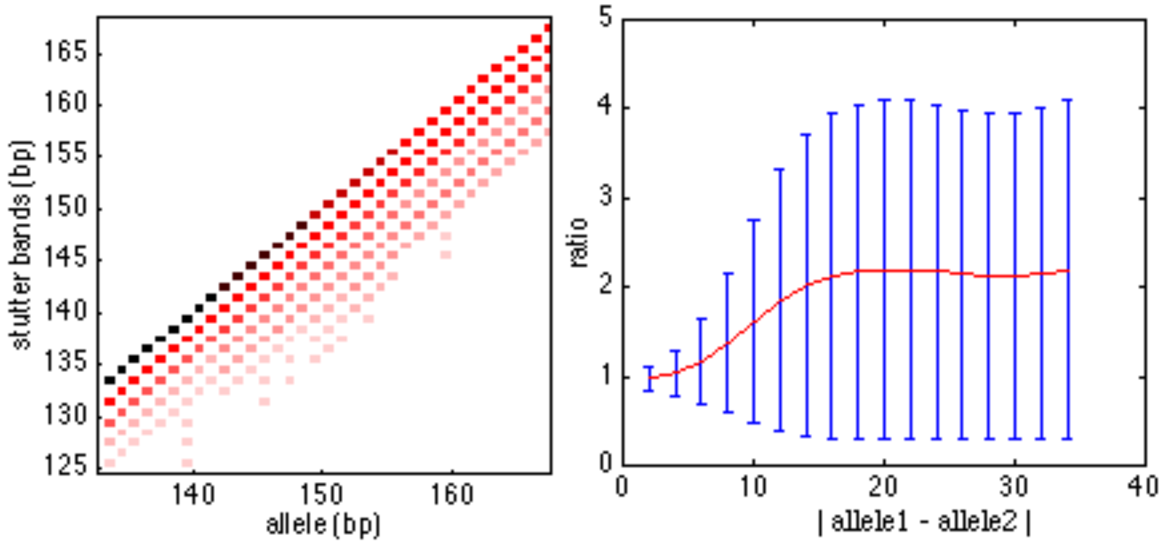


Figure 6.11. Computed marker library for marker D22S283. On the left is the final stutter matrix A constructed from the 8 reference lanes. On the right is the corresponding relative amplification ratio table ρ. Compare these with the marker library constructed with 16 reference lanes: the stutter matrix A shown in Figure 6.3 and the amplification ratio table ρ shown in Figure 6.7.

To verify that stutter matrix A can reliably predict stutter patterns from the genotypes, we multiply stutter matrix A with the *weighted* genotype matrix X (to generate the predicted stutter patterns), and then compute the squares of the differences between the predicted and observed stutter patterns. The small sum of squares in each column verifies that the stutter patterns predicted by A are indeed close to the observed patterns in Y.

Predicted pattern matrix AX

(bp)	lane 11	lane 12	lane 13	lane 14	lane 15	lane 16	lane 17	lane 18
131	0	0	0	0	0	0.0208	0	0
133	0	0	0	0	0	0.0416	0	0
135	0	0.0217	0.0233	0	0.0235	0.1248	0	0
137	0	0.0326	0.0350	0	0.0353	0.3224	0	0.0254
139	0.0115	0.0744	0.0700	0.0200	0.0706	0.5400	0.0110	0.0455
141	0.0345	0.1798	0.1633	0.0600	0.1648	0.0288	0.0330	0.0984
143	0.0460	0.3628	0.3500	0.0800	0.3531	0.0384	0.0620	0.2073
145	0.0805	0.5425	0.5134	0.1400	0.5178	0.0672	0.1310	0.4325
147	0.1380	0.1110	0	0.2400	0	0.1152	0.1590	0.6473
149	0.3505	0.2682	0.0169	0.5800	0.0167	0.2784	0.4630	0.2160
151	0.5570	0.4070	0.0507	0.8800	0.0501	0.4224	0.7630	0.3277
153	0.0255	0	0.0253	0	0.0250	0	0.3780	0
155	0.1360	0	0.1352	0	0.1336	0	0	0
157	0.2635	0	0.2619	0	0.2588	0	0	0
159	0.3570	0	0.3549	0	0.3507	0	0	0
161	0	0	0	0	0	0	0	0
total	2	2	2	2	2	2	2	2

Squared differences between Y and AX

(bp)	lane 11	lane 12	lane 13	lane 14	lane 15	lane 16	lane 17	lane 18
131	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
133	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
135	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
137	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
139	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00
141	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
143	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
145	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01
147	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.00
149	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01
151	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01
153	0.00	0.00	0.00	0.01	0.00	0.01	0.00	0.01
155	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00
157	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
159	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00
161	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
sum	0.01	0.01	0.02	0.02	0.03	0.02	0.03	0.05

6.7. Algorithms: Genotyping by Deconvolution

From our preliminary study of different deconvolution methods, the best algorithms for genotyping microsatellite by deconvolution (GMBD) are SVD, GAUSS, and ENUM. SVD and GAUSS turn out to be algorithmically similar[18], which explains the comparable performance of both methods on simulated data. ENUM, on the other hand, differs from SVD and GAUSS in that it searches exclusively in the more constrained space of genotypes containing *exactly* two alleles.

A useful strategy for improving robustness in handling real data is to use different methods to call the alleles, and then intelligently determine a consensus from the independent calls. For GMBD, we have chosen the SVD and ENUM algorithms. We omit GAUSS as it is algorithmically similar to SVD, and we prefer SVD for its efficiency, robustness, and extendibility[19].

In Box 6.6, we present our GMBD algorithm DECONVOLVE_GENOTYPE that uses the SVD and ENUM algorithms.

---

[18]Both SVD and GAUSS invert A and then apply it to$Y$. The difference is that SVD uses singular value deomposition to invert A in one single operation, whereas GAUSS uses a Gaussian elimination procedure to remove the stutters iteratively.

[19]We will see in the next chapter that SVD can be extended to perform pooled GMBD without much modification.

Algorithm: DECONVOLVE_GENOTYPE

Step 1: Call the alleles using multiple independent algorithms.

(a) Normalize the data vector y so that the vector sum is 2, as there are two alleles in the DNA sample.

(b) Use algorithm SVD_DECONVOLVE to determine the alleles on the data vector y with the marker library $\langle A, \rho \rangle$. Call the resulting genotype vector x_{svd}.

(c) Use algorithm ENUM_DECONVOLVE to determine the alleles, producing the genotype vector x_{enum}.

Step 2: Obtain a consensus.

If x_{svd} agrees with x_{enum}, then output it as the consensus genotype vector.

If not, let x_{enum_i} be the i-th best candidate genotype vector from ENUM_DECONVOLVE. Select from the top candidates the x_{enum_i} with the smallest total sum of squares error between Ax_{enum_i} and y and between x_{enum_i} and A/y. Output this x_{enum_i} as the final consensus genotype vector.

Box 6.6. DECONVOLVE_GENOTYPE: An algorithm for genotyping microsatellites by deconvolution. It uses two independent deconvolution methods for genotyping, and then computes the best consensus genotype from these candidate solutions.

6.7.1. SVD

SVD, or *singular value decomposition,* is a fast and robust algorithm from computational linear algebra for solving over- and under-determined linear systems (such as $Ax=y$). We have already used it on several occasions in our library construction algorithm as a quick and reliable genotyping method. Beside its computational efficiency, SVD can be useful in detecting any extra alleles (that may be present due to contamination or other data anomalies) since it does not make any assumptions about the expected values in the genotype vector x. In fact, it is this "no-assumptions" characteristic that gives SVD its versatility in performing pooled GMBD (see the next chapter), when there are indeed extraneous (more than 2) alleles present in the DNA samples. Box 6.7 presents our SVD deconvolution algorithm in more detail.

Algorithm: SVD_DECONVOLVE

Step 1: Deconvolve with SVD.

Solve for x_{svd} using the singular value decomposition of the stutter matrix A on the normalized data vector y (Golub and Van Loan, 1989; Lawson and Hanson, 1974; Press et al., 1992).

Step 2: Call the alleles.

Depending on the quality of the observed data y, there may be more than two non-zero entries in the resulting vector x_{svd}. To actually call the alleles, we iterate over possible genotypes based on these non zero entries in x_{svd}, to search for the allele pair which fits best with data vector y.

Let a_i denote the i-th entry in the vector x_{svd} , and $x^{i,j}$ be a genotype vector created with a_i as the i-th entry and a_j as the j-th entry.

Select from x_{svd} , a limited set of indices with nontrivial a_i 's (say, $x_i > 0.25$). Create from this set of indices all possible homozygotic ($i=j$) and heterozygotic ($i \neq j$) genotype vectors $x^{i,j}$s, provided that the ratio of a_i to a_j is in the range specified by $\rho(a_i, a_j)$. Output the genotype vector $x^{i,j}$ with the least sum of squares deviation between $Ax^{i,j}$ and y.

Box 6.7. SVD_DECONVOLVE: A genotyping algorithm using singular value decomposition.

6.7.2. ENUM

ENUM is a search-based GMBD algorithm. Unlike SVD, it assumes that there are at most two distinct alleles in the genotype, and then searches for the best genotype vector (x) that produces an expected stutter pattern (Ax, where A is the stutter matrix) that has the least sum of squares error against the observed data vector (y). To handle relative amplification, ENUM performs a local hill-climbing search for the best amplification ratio to apply at each proposed genotype. In contrast with SVD, ENUM's local searching is more cautious and sensitive to alleles with low amplifications. SVD is faster and more robust against background noise because of its global view. The two algorithms work very well together, complementing each other's potential limitations.

Box 6.8 shows the details of the ENUM algorithm.

Algorithm: ENUM_DECONVOLVE

Step 1: Identify candidate alleles.

Let $y(i)$ be the i-th entry in the normalized data vector y, and $a_{c(i)}$ be the corresponding allele where $c(i)$ is the corresponding column in the stutter matrix A for the allele. The set $\{a_{c(i)} : y(i)>0\}$ represents the candidate alleles[20] on which we enumerate the possible genotype vectors.

Step 2: Enumerate over candidate alleles.

Let $\alpha_{i,j}$ be the mean of the amplification ratio range in $\rho(a_{c(i)}, a_{c(j)})$. We enumerate over the set of candidate alleles by creating genotype vectors $x^{i,j}$ with $\alpha_{i,j}$ as the $c(i)$-th entry and $1/\alpha_{i,j}$ as the $c(j)$-th entry initially.

Step 3: Locally search for amplification ratios.

Adjust each $x^{i,j}$ locally by performing a hill-climbing search for the best amplification ratio allowed by the range $\rho(a_{c(i)}, a_{c(j)})$. Use the sum of squares error between the predicted vector $Ax^{i,j}$ and the observed data vector y as an error measure in the search.

Step 4: Search for the genotype vector having the best fit.

For each candidate vector $x^{i,j}$, compute the sum of squares error between the predicted $Ax^{i,j}$ and the observed y. Keep track of a set of top candidate genotype vectors for determining a consensus later in the algorithm DECONVOLVE_GENOTYPE. Output the genotype vector $x^{i,j}$ that has the least sum of squares error.

Box 6.8. ENUM_DECONVOLVE: A search-based genotyping algorithm.

[20] For efficiency, we can prune the candidate allele set even further. For example, we can restrict it to contain alleles with $y(i) > min(1/\rho)$ (the smallest allele mass allowable by the amplification ratio table).

6.7.3. Example

To explore the difference between SVD and ENUM, we review an example from marker D16S405, a FAM-labeled dinucleotide repeat marker in the allele range 103–161 bp. Shown in Figure 6.12, the genotyping experiment was run out on lane 9 of a 34-lane ABI/377 gel[21]. The top pane of the figure shows the electropherogram, the middle pane shows the quantitated relative DNA concentrations of the data bands, and the third pane shows the alleles called by the computer.

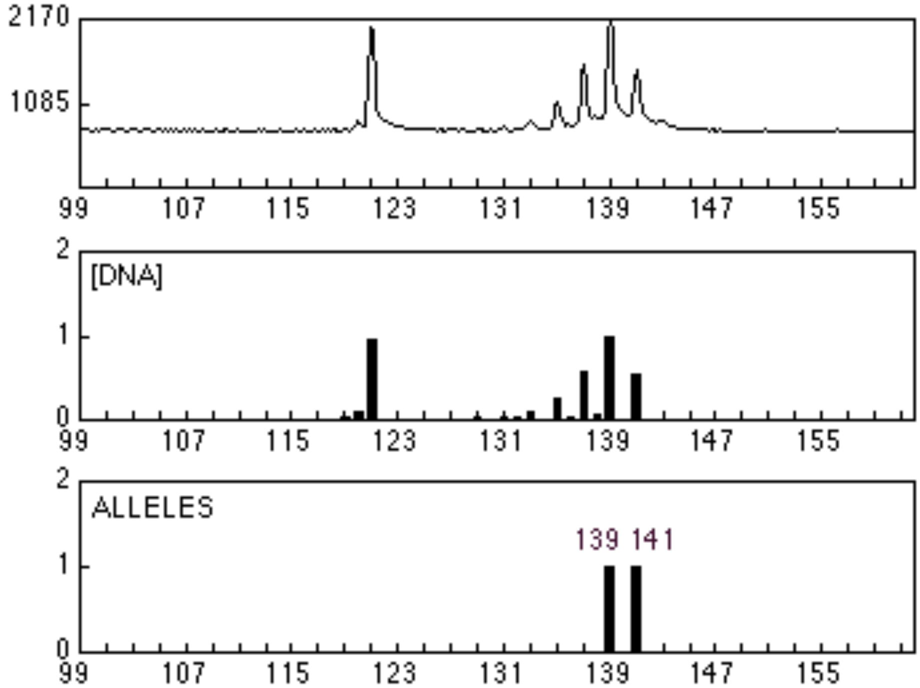


Figure 6.12. An example genotyping experiment from marker D16S405, a FAM-labeled dinucleotide repeat marker in the allele range 103-161 bp. This genotyping experiment was run out on lane 9 of a 34-lane ABI/377 gel.

What makes this example interesting is the presence of a singular spurious band in the electropherogram. This spurious band has a similar peak shape (e.g., with respect to the peak width and height) as the data bands in the experiment. It also has an allele size (121 bp) with the expected parity (odd) of the alleles for marker D16S405. As such, one cannot distinguish this band from the other bands based on peak shape or allele parity information. A naive allele determination algorithm that calls the two highest peaks in the electropherogram would have called the 121 and 139 bp alleles as the genotype. However,

[21]In fact, this is the "demo gel" that we ship with FAST-MAP, our automated genotyping system described in Chapter 8.

to the expert human genotyper, it is clear that the 121 bp band is an artifactual band — there are no stutter bands associated with it, and PCR stuttering is expected for D16S405 (based on data from the other lanes). Like a human expert, our deconvolution algorithm learns about the characteristic patterns for marker D16S405, and expects to find stutter patterns associated with a true allele band. As a result, the computer was able to intelligently call the alleles at 139 and 141 bp.

To see how SVD and ENUM complement each other in this example, let us look at Figure 6.13, which shows the different computer calls. On the top pane is the computer's final call. This final call is based on a consensus of the candidate calls shown in the panes below. The second pane shows the deconvolution of data y with stutter matrix A by the SVD algorithm. Because SVD does not restrict its call to at most two distinct alleles, it was able to detect three alleles at 121, 139, and 141 bp. However, as an individual's genotype can contain at most 2 different alleles, SVD must select two of these three in its final call. It does this by selecting the two with the tallest amplification values, which, unfortunately, erroneously included the 121 bp band. On the other hand, the 121 bp could have actually been an allele from DNA sample contamination or dye bleedthrough, and SVD would detect such cases and alert the geneticist. The ENUM algorithm, which tries out all possible allele pair combinations for one that fits the data best, assumes at most two distinct alleles in the genotype. If there were more than two alleles present in the DNA sample, ENUM could not detect them. Thus, while SVD is sensitive in detecting extra alleles, ENUM is robust against spurious noise. In this case, ENUM produces the correct allele call as its top choice (ENUM1), with the other two paired combinations of the three alleles as its second and third choices (ENUM2 and ENUM3).

Since the alleles called by SVD differ from those called by ENUM1, the computer must select a consensus from the three candidate ENUM allele calls. The selection is based on two criteria: (1) how well the predicted stutter pattern, computed by multiplying the proposed genotype vector with the stutter matrix A, fits the observed data, and (2) how closely the proposed genotype vector, weighted by an appropriate relative amplification ratio, correlate with the genotype vector obtained by SVD deconvolution. For each of the three weighted genotype vectors x_{enum_i} in ENUM, the computer computes the sum of squares error between the observed data vector y and the predicted pattern vector Ax_{enum_i}, as well as the sum of squares error between x_{enum_i} and SVD's genotype vector x_{svd}. In this example, ENUM1 has the least total sum of squares errors, so the computer outputs ENUM1 as its final allele call.

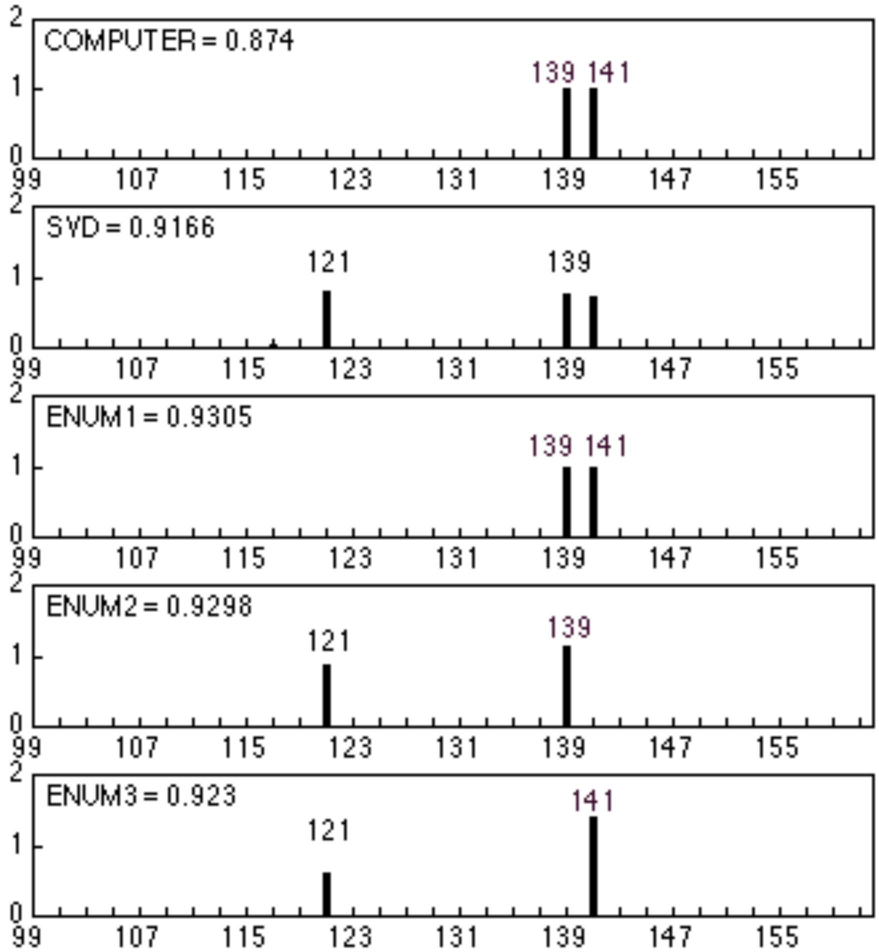


Figure 6.13. The performance of the SVD and ENUM deconvolution algorithm on the example shown in Figure 6.12. The darkened bars in each pane show the alleles called, and the heights correspond to their relative amplifications. The pane labeled "COMPUTER" shows the final consensus call made by the computer based on the results from the independent deconvolution algorithms. The pane marked "SVD" shows the call made by SVD, while "ENUM1", "ENUM2", and "ENUM3" display the top three choices made by ENUM. The real number associated with each pane indicates the computer's confidence in the corresponding genotype.

Our GMBD algorithms based their allele calls on pattern matching. Two types of pattern information are employed: for each marker, the stutter patterns stored in the stutter matrix *A,* and the relative amplifications the amplification ratio table ρ. We have shown in the previous example how our GMBD algorithms use stutter pattern expectations to robustly call the alleles; let us now look at how the algorithms handle relative amplifications between the alleles in the genotypes.

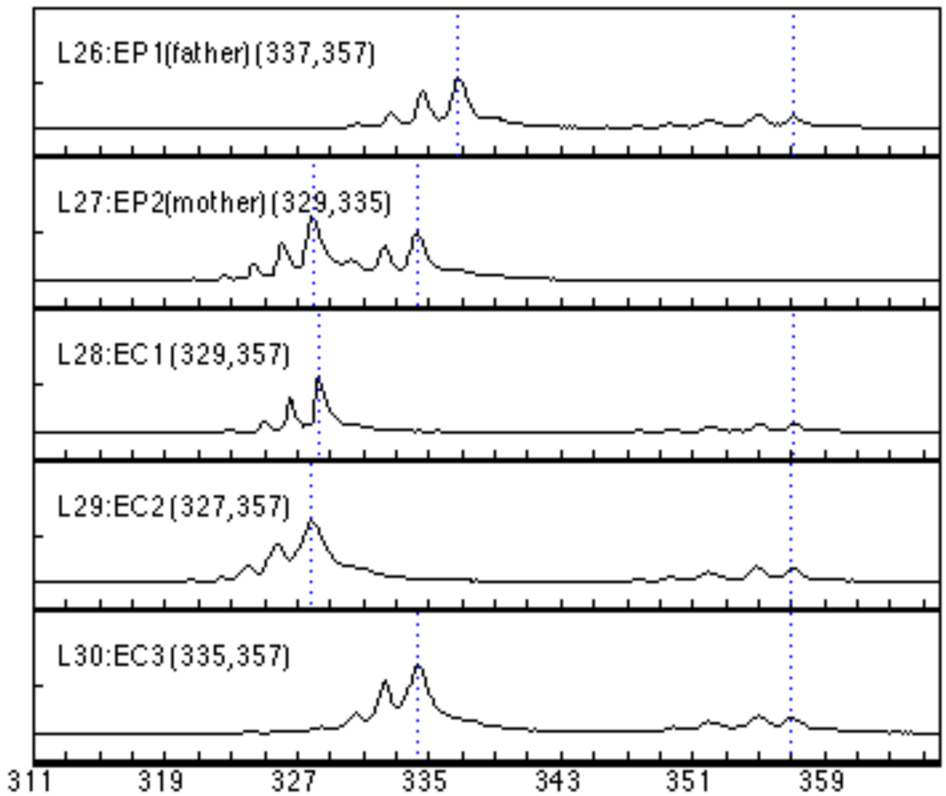


Figure 6.14. Relative amplification in marker D16S405. Shown are the electropherograms for marker D16S405 from a family. The genotyping experiments were run out on lanes 26 through 30 on the ABI/377 gel from the previous example. The vertical dotted lines indicate the alleles called by the computer. Each pane is labeled by the lane number (e.g. L26), the sample ID (e.g. EP1), and the corresponding genotypes.

Figure 6.14 shows the data for marker D16S405 from a nuclear family (the genotyping experiments for this family were run out on the same gel as the previous example). When the alleles in a heterozygotic genotype are far apart (e.g. all the individuals in the family except for the mother EP2), the amplification of the data bands from the smaller allele can differ greatly from the data bands for the larger allele. It is conceivable that even an expert human genotyper may miss the less amplified allele in a genotype when there is a pronounced relative amplification between the alleles (e.g. EC1 in lane 28). However, by using *both* stutter patterns and relative amplifications as pattern expectations, the computer was able to accurately call all the alleles in the family shown. Note that the computer does not make use of any pedigree information to assist its allele determination (we want to leave the pedigree as an independent information source for the user to confirm allele calls); all the calls by the computer were based solely on the stutter matrix A and the amplification ratio table ρ that the computer had compiled for marker D16S405 from previous processing.

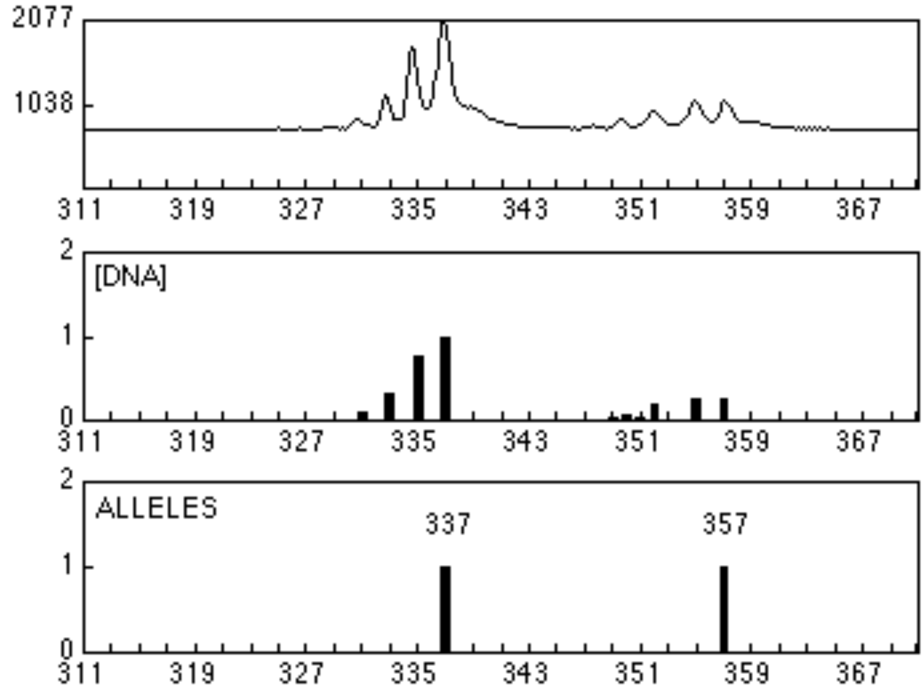


Figure 6.15. Relative amplification. When the alleles in a genotype are widely separated (with respect to their sizes), they may not be amplified in an 1:1 ratio. Here, the bands for the bigger allele (357 bp) are considerably less amplified than the bands for the smaller allele (337 bp).

Figure 6.15 shows the detailed view of the genotyping data for the father EP1 (lane 26). As you can see, the total amount of DNA for the bands originated from the smaller allele (337 bp) is much greater than the total DNA for the bands from the bigger allele (357 bp). A naive algorithm that assumed equal amplification ratios for the two alleles would miss the less amplified 357 bp allele. Our GMBD algorithms employ the relative amplification knowledge from the marker library, and know that from past experiences, the larger sized allele in a genotype that is 10 bp apart (assuming we are using a compressed amplification ratio table) is typically much less amplified than the smaller sized allele. With this information, the computer can construct a highly accurate expectation that fits the data. For example, in Figure 6.16, the relative amplifications (as indicated by the heights of the darkened bars) for the proposed genotypes in ENUM1, ENUM2, and ENUM3 are adjusted so that the individual alleles were allocated the appropriate proportions of total amplified DNA. As a result, the computer was able to call the 357 bp allele, even though it was much less amplified than the 337 bp allele.

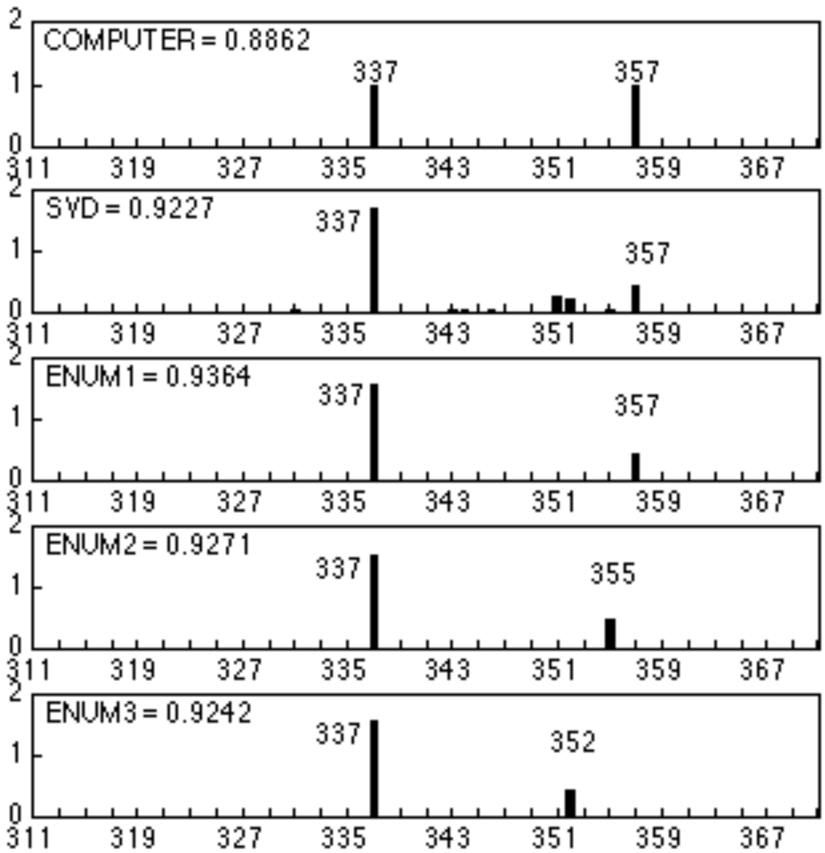


Figure 6.16. Adjustment for relative amplification. Using the expected amplification ratios from the marker library for D16S405, the computer adjusts the relative amplification between alleles in each proposed genotype accordingly. For example, the larger-sized alleles for the genotypes in ENUM1, ENUM2, and ENUM3 were all given a smaller share of the total DNA fraction (indicated by the heights of the darkened bars) because they were expected to be less amplified than their smaller-sized counterparts.

6.8. Discussion

An important computational approach that is central to this dissertation is to *use all the data available.* When analyzing real data, data *quantity* can be a useful antidote to insufficient data *quality*[22]. The inherent data redundancy in large amount of data can be used to detect data consistency for filtering out random data artifacts present in less perfect data.

To maximize the amount of data that can be exploited for redundancy, we adopted a non-conventional approach, treating the PCR stutter bands as *data*. Traditionally, PCR stuttering has been considered to be noise, and geneticists tried to suppress the artifactual bands (Odelberg and White, 1993) from the genotyping data. With our systematic approach of accounting for the stutter bands, the stutter bands become an integrated part of the data, critical for robust and consistent processing. In the previous chapter, we made use of the inherent continuity of stutter bands to overcome (by stutter crawling) the inadequate sizing resolution of the MW size standards. In this chapter, we used the expectation of stutter bands to unambiguously detect spurious peaks, even if they looked just like real data bands (see the D16S405 example in Figure 6.12). To illustrate the advantage of stuttering further, we show, in Figure 6.17 below, a similar case for marker D6S1050, a tetranucleotide repeat marker that does not exhibit PCR stuttering. Again, an additional band is present in the electropherogram that is indistinguishable from actual allele bands in shape and size . However, without stutter bands, it is impossible for man or machine to distinguish the true allele bands from the lone noise band[23].

---

[22]Of course, no amount of data would be sufficient to recover data that are of very bad quality.
[23]From the data shown, the other possible scenario in which there are actually two noise bands and only one allele band (a homozygotic genotype) is less likely because one of the bands (the actual allele band) would have been much taller (say, twice as tall) than the other bands.

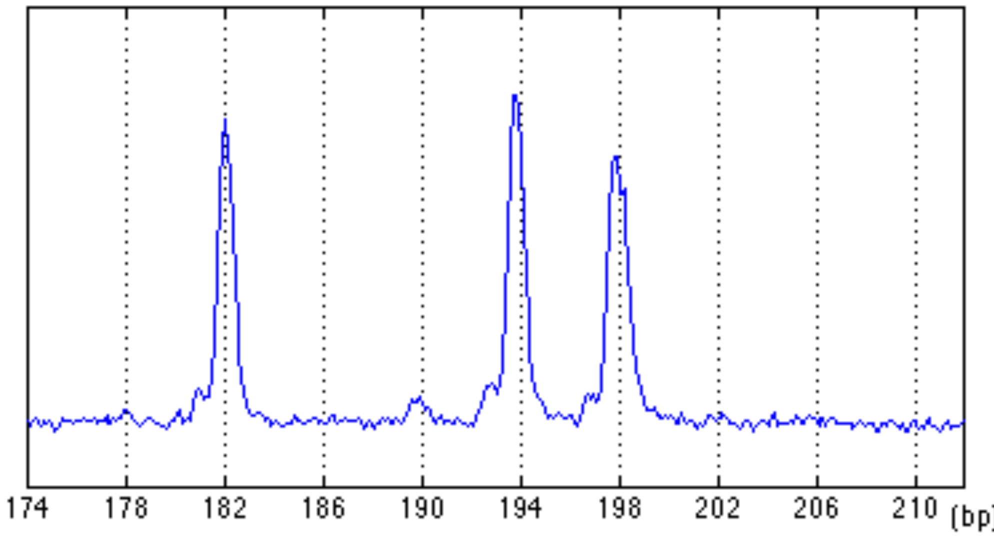


Figure 6.17. An additional band amidst the allele bands for marker D6S1050, a HEX-labeled tetranucleotide repeat marker that has an allele range of 168-215 bp.

The implication of our results is quite important. Geneticists have recently moved away from dinucleotide repeat markers because of PCR stuttering. Instead, they have resorted to trinucleotide and tetranucleotide repeat markers, mainly because of the lack of stutter bands in these markers. However, to use these larger repeats that do not exhibit PCR stuttering, one has to pay the price of using markers that are more complex, less informative, less stable, less densely distributed over the genome, and consume more "real estate" on the gel than the dinucleotide repeat markers. Our work in this dissertation showed that it is actually not the stuttering but the lack of stutter bands that is a true disadvantage when it comes to microsatellite genotyping. The geneticists can now resume to using the useful dinucleotide repeat markers (or even the mononucleotide repeat markers, as we will show in Chapter 9) for more productive genetic studies.

7. New functionality: Pooled genotyping

Allele frequency methods such as linkage disequilibrium and association studies are emergent methods for studying the genetics of complex diseases (Ghosh and Collins, 1996; Lander and Schork, 1994; Risch and Merikangas, 1996). Instead of tracing the inheritance patterns among individuals in families, these methods look for causative genes along the genome by testing, for each marker, whether a particular allele occurs at a higher frequency among the affected population than the unaffected individuals[24]. If there is a significant collective loss in the allelic variation (as indicated by the allele frequencies) in the chromosomal regions among the affected population, then the causative genes for the disease are likely to reside in the proximity of these regions.

However, linkage disequilibrium typically extends over very short genomic distances, requiring *high-resolution* genome-wide association tests involving tens of thousands of genetic markers. In theory, we can (naively) determine the allele frequencies of all the markers by genotyping every individual in the population, but the scope will be incapacitating in practice. Moreover, it is probably an overkill to obtain all the individual genotypes as only the allele *frequencies* are required for association studies. What we need is an accurate and efficient method to screen a large number of marker polymorphisms (alleles) without having to genotype each and every one in the population individually.

One such method is to pool DNA samples together in a single experiment for PCR and gel readout (LeDuc *et al.*, 1995; Risch and Merikangas, 1996). This method is easily workable with genetic markers that do not exhibit PCR stuttering (e.g. tetranucleotide repeat markers): the allele frequencies of the pooled samples can simply be read off the relative allelic concentrations (peak heights or areas) in the electropherograms[25]. With PCR stuttering, the observed band pattern for each pooled experiment will be a convolution of a large number of stutter patterns. The observed band pattern from a pool size of, say, 10 DNA samples can be a convolution of up to 20 different stutter patterns. Even an experienced human genotyper would not be able to decipher such convoluted patterns.

---

[24]The individuals can be siblings (as in sib-pair analysis), family members (as in familial clustering analysis), or unrelated individuals from a population (preferably isolated and homogenous).

[25]This is actually an over-simpification, as the effect of relative amplification (which is also present in the larger repeats) must also be taken into account for. Of course, with stutterless markers, the adjustment of relative amplification is much more straightforward than markers with convoluted stuttering.

The larger repeats (e.g. tetranucleotide repeat markers), while typically stutter-free, are less abundant and can therefore provide only *limited resolution* for genome scans. Eventually, the smaller repeats (e.g. dinucleotide repeat markers) must be used to attain the fine resolution required for detecting linkage disequilibrium. However, these repeats usually exhibit PCR stuttering, and pooled sample genotyping is not feasible unless the PCR stutter bands can be eliminated from the data. In this chapter, we will show how the problem of PCR stuttering in pooled genotyping can be solved *computationally,* using only a simple generalization of our original convolution model for single-sample genotyping.

7.1. Convolution model

The overall pattern observed on a gel run-out for a pooled sample is the sum of the band patterns from the genotype of each individual sample present in the pool. Under our matrix convolution model, we can model PCR stuttering in pooled DNA as:

$$y = \sum_i A \cdot x_i$$

where each x_i represents the genotype vector of a DNA sample in the pool.

Using the distributive law of matrix operations, we re-write the above relation as:

$$y = A \cdot \left(\sum_i x_i \right)$$

Ignoring the effects of relative amplification[26] for now, the vector $\sum_i x_i$ is the allele distribution in the DNA sample mixture. As with single-sample genotyping, we can recover this allele distribution vector by deconvolution:

$$\sum_i x_i = A \setminus y$$

---

[26]In the presence of relative amplification, we can still recover the allele distribution $\sum_i x_i$ from $A\backslash y$ by adjusting it with the relative amplification ratios ρ (see step 2 of Algorithm POOLED_SVD).

To compute the allele frequency from the allele distribution vector, we normalize by dividing each entry in $\sum_i x_i$ with *2n,* where *n* is the number of equimolar DNA samples in the pool.

7.2. Pooled GMBD

An immediate corollary of the above generalized convolution model is that we can apply convolution methods that are similar to those we have designed for the single-genotype problem to the multiple-genotype problem. In fact, all the algorithms except ENUM (POLY, FFT, WIENER, SVD, and GAUSS) can be applied directly in pooled GMBD, as none of them are algorithmically constrained by the actual number of alleles or genotypes present. Although ENUM had assumed that there are at most two distinct alleles, we can easily generalize it to handle *2n* alleles, where *n* is the number of samples pooled. The real problem with ENUM is that direct enumeration of the $O(m^n)$ possible genotype vectors for an *n*-sample pool genotype of an *m*-allele marker is computationally prohibitive. To circumvent this computational threshold, we employ a hill-climbing search instead of exhaustive enumeration. In fact, hill-climbing is capable of producing comparable results if we start from a good initial estimate. We call this sixth algorithm SEARCH to distinguish it from ENUM:

- SEARCH: Starts with a good initial solution (e.g. from SVD) and then locally performs a hill-climbing search procedure to find a better solution.

A comparative study

To verify the feasibility of pooled GMBD and to compare the various deconvolution methods in pooled genotyping, we tested the algorithms on simulated pooled genotype data (Perlin *et al.*, 1995). We generated a test set of 300 simulated pooled genotypes, each comprising 100 alleles (50 individuals, each with 2 alleles). The alleles were randomly drawn from each marker based on their frequency distribution. Again, note that we did not include relative amplification in our simulations of pooled DNA stutter data.

For each of the 300 pooled genotypes, we applied our pooled GMBD algorithms to compute the allele frequency vectors. For comparison, we computed the average sum of squares errors between the estimated and the actual allele frequency vectors. We show the results in Table 7.1.

The result support our notion of using stutter-based deconvolution in quantitative PCR-based pooled genotyping. As expected, the allele-dependent deconvolution algorithms (GAUSS, SVD, SEARCH) performed better than the allele-independent algorithms (POLY, FFT, WIENER) . In fact, the difference was more pronounced with pooled genotyping, since variation in stutter patterns are accentuated as more alleles were combined. Just as SVD and ENUM were best with single-sample genotyping, with pooled genotyping SVD and SEARCH (the hill-climbing version of ENUM) performed better than the other deconvolution algorithms investigated.

	Noise			
	0%	5%	10%	15%
Moderate stutter (5 bands):				
POLY	.362	.387	.374	.437
FFT	.192	.203	.221	.276
WIENER	.192	.202	.220	.274
GAUSS	.000	.031	.065	.101
SVD	.000	.029	.058	.093
SEARCH	.000	.021	.048	.079
Severe stutter (10 bands):				
POLY	.650	.662	.688	.736
FFT	.492	.518	.538	.582
WIENER	.492	.515	.543	.589
GAUSS	.000	.033	.084	.138
SVD	.000	.033	.081	.125
SEARCH	.000	.025	.071	.116

Table 7.1. The average sum of squares errors for pooled DNA deconvolution algorithms on simulated data. The simulation studies were conducted with 300 pools of 50 simulated genotypes (50×2 alleles) from markers with 10 to 25 normally distributed alleles.

7.3. Algorithms

For convenience, we compute the allele distribution vectors ($\sum_i x_i$), instead of the allele frequencies. The frequencies can be recovered by dividing the entries in $\sum_i x_i$ by $2n$, where n is the number of equimolar DNA samples in the pool. This seemingly trivial choice actually has an important algorithmic consequence — it greatly reduces our search space from the real-valued vector space of allele frequencies to the far more constrained integer-valued vector space of allele distributions.

To mathematically remove the convoluted stutters, we use the marker libraries constructed from single-sample genotype data to deconvolve the pooled data. Again, we use multiple methods (e.g. SVD and SEARCH) to compute the allele distributions, and then heuristically determine a consensus from the various calls. Box 7.1 shows the POOLED_GENOTYPE algorithm in more detail.

Algorithm: POOLED_GENOTYPE

Step 1: Compute the allele distribution using two independent algorithms.

(a) Normalize the data vector y so that the vector sum is $2n$, where n is the number of samples in the DNA pool.

(b) Use algorithm POOLED_SVD to compute an allele distribution vector x_{svd}.

(c) Use algorithm POOLED_SEARCH to compute an alternate allele distribution vector x_{search}.

Step 2: Obtain a consensus.

If x_{svd} agrees with x_{search}, then output x_{svd} as the consensus allele distribution vector.

If not, let x_{search_i} be the i-th best candidate genotype vector from algorithm POOLED_SEARCH.

Select from the top candidates the x_{search_i} with the smallest total sum of squares errors between Ax_{search_i} and y and between x_{search_i} and A/y. Similarly, compute the sum of squares error for x_{svd}.

If the sum of squares error for x_{search_i} is less than that for x_{svd}, output x_{search_i} as the allele distribution vector for the pooled data vector y. Otherwise, output x_{svd}.

Box 7.1. POOLED_GENOTYPE: An algorithm for genotyping pooled microsatellite DNA samples by deconvolution. It uses two independent deconvolution methods for genotyping, and computes the best consensus from the answers.

SVD

In essence, singular value decomposition performs a *weighted* re-centering of the allele mass using the stutter patterns at the various alleles. As such, SVD is not restricted to the number of alleles present, and is therefore equally applicable to single-sample GMBD as to pooled GMBD. That is, the algorithms POOLED_SVD and SVD_DECONVOLVE are essentially the same. To output an integral allele distribution vector, we start with the allele that has the highest value in the SVD vector and iteratively round the vector values until we have accounted for all the alleles the pooled sample. Box 7.2 describes POOLED_SVD algorithm in greater detail.

Algorithm: POOLED_SVD

Step 1: Deconvolve with SVD.

Solve for x_{svd} using the singular value decomposition of the stutter matrix A on the (normalized) data vector y.

Step 2: Iteratively round for the integral allele distribution.

To convert the real-valued vector x_{svd} into an integer-valued allele distribution vector x_{svd_round}, we iteratively round the entries, starting with the most confident allele, until we have found all $2n$ alleles.

(a) Let k be the index to the entry in x_{svd} that has the largest value. Set

$$x_{svd\_round}(k) \leftarrow \mathrm{round}(x_{svd}(k))$$

where $x_{svd}(k)$ and $x_{svd\_round}(k)$ denote the k-th entries of x_{svd} and x_{svd_round} respectively.

(b) Let i be the index to the entry in x_{svd} that has the next largest value. If i is smaller than k, set

$$x_{svd\_round}(i) \leftarrow \mathrm{round}(x_{svd}(i) \times \mathrm{mean}(\rho(a_i, a_k)))$$

Otherwise, set

$$x_{svd\_round}(i) \leftarrow \mathrm{round}(x_{svd}(i) \div \mathrm{mean}(\rho(a_k, a_i)))$$

This accounts for relative amplification between the alleles at i and k.

(c) Repeat (b) until all the $2n$ alleles have been accounted for, or until there are no non-zero entries in x_{svd} to be rounded. In the latter case, add the number of unaccounted alleles to $x_{svd\_round}(k)$. Output the allele distribution vector x_{svd_round}.

Box 7.2. POOLED_SVD: A pooled GMBD algorithm using singular value decomposition.

SEARCH

For pooled genotyping, the exhaustive search employed in the ENUM algorithm for single-sample GMBD is no longer feasible. As such, we use more intelligent search strategies to overcome the added complexity in pooled genotyping.

Given a good initial solution, local search strategies can produce good results efficiently. In our POOLED_SEARCH algorithm, we use a hill-climbing search strategy that starts with the SVD solution as an initial estimate. At each search step, the computer explores the search space by micro-adjusting the distribution vector: the computer creates a new allele distribution from the old vector by redistributing one unit of an allele to another, and then evaluating whether the new allele distribution produces an expected band pattern that fits better with the actual observed pattern. With large number of samples in the pooled DNA, or with markers having a large number of different alleles, even this search step can be computationally intensive. As such, it is important to carefully implement POOLED_SEARCH using efficient search strategies such as those commonly found in artificial intelligence (e.g. beam search or A*) (Rich and Knight, 1991). Box 7.3 describes the POOLED_SEARCH algorithm in detail.

Algorithm: POOLED_SEARCH

Step 1: Identify candidate alleles.

Let $y(i)$ be the i-th entry in the normalized data vector y. Let $a_{c(i)}$ be the corresponding allele, where $c(i)$ is the column in the stutter matrix A for the allele. The set $\{a_{c(i)} : y(i)>0\}$ represents the maximal set of candidate alleles[27] that may appear in the allele distribution vector.

Step 2: Search for a better allele distribution vector.

(a) Let BEST be the set of the best allele distribution vectors that seen thus far. If a beam search of width w is employed, restrict BEST to contain only the best w vectors. Initially, BEST = $\{x_{svd\_round}\}$.

(b) Remove an allele distribution vector x_{best} from BEST that was not processed before. Create NEW, a set of new allele distribution vectors by iteratively re-distributing an allele from x_{best} over the candidate alleles. Exclude from NEW any vectors that were analyzed before.

(c) For each vector x_{new} in NEW, compute the sum of squares error between the expected stutter data Ax_{new} and the observed data vector y. Additionally, minimally adjust the values in x_{new} as permitted by the amplification ratio table ρ to further minimize the sum of squares error. Associate with x_{new} the smallest error.

(d) If there are any x_{new} 's in NEW that have a smaller error than x_{best} , then add them to the set BEST. If not, put x_{best} back into BEST, and labeled it "processed".

(e) Repeat until there are no more unprocessed vectors in BEST, or until a search bound has been reached. Output the best vector in BEST as the final allele distribution vector.

Box 7.3. POOLED_SEARCH: A search-based genotyping algorithm for pooled samples.

---

[27] As in ENUM_DECONVOLVE,we may want to prune this candidate set even further. For example, we can restrict it to contain only alleles with $y(i) > min(1/\rho)$ (the smallest allele mass allowable by the amplification ratio table).

7.4. Example

First, individual DNAs for D16S403, a FAM-labeled dinucleotide repeat marker with an allele range of 125-155 bp, were PCR amplified and size separated on an ABI/377 sequencer. We used these experiments to determine the alleles in each DNA sample, and also to construct the binning, stutter, and relative amplification calibrations for D16S403. Pools of 2, 4, and 8 individual PCR products were constructed, and size separated on the ABI.

In Figure 7.1, we show the results of pooled GMBD on one of the two-sample pools. The topmost pane shows the electropherogram from the gel readout of the pooled experiment. The second pane shows the relative DNA concentrations of the data bands using our band quantitation methods. In the third pane, we show the actual allele distribution for this experiment, computed from the individual genotypes of the two samples in the pool. Since the two genotypes are both heterozygotes, the allele distribution shows four different alleles. The computer's call is in the fourth pane labeled "COMPUTER". In this example, our POOLED_GENOTYPE algorithm correctly calls all four alleles present in the pool, despite the peak overlap, plus-A, and PCR stutter artifacts present in the data.

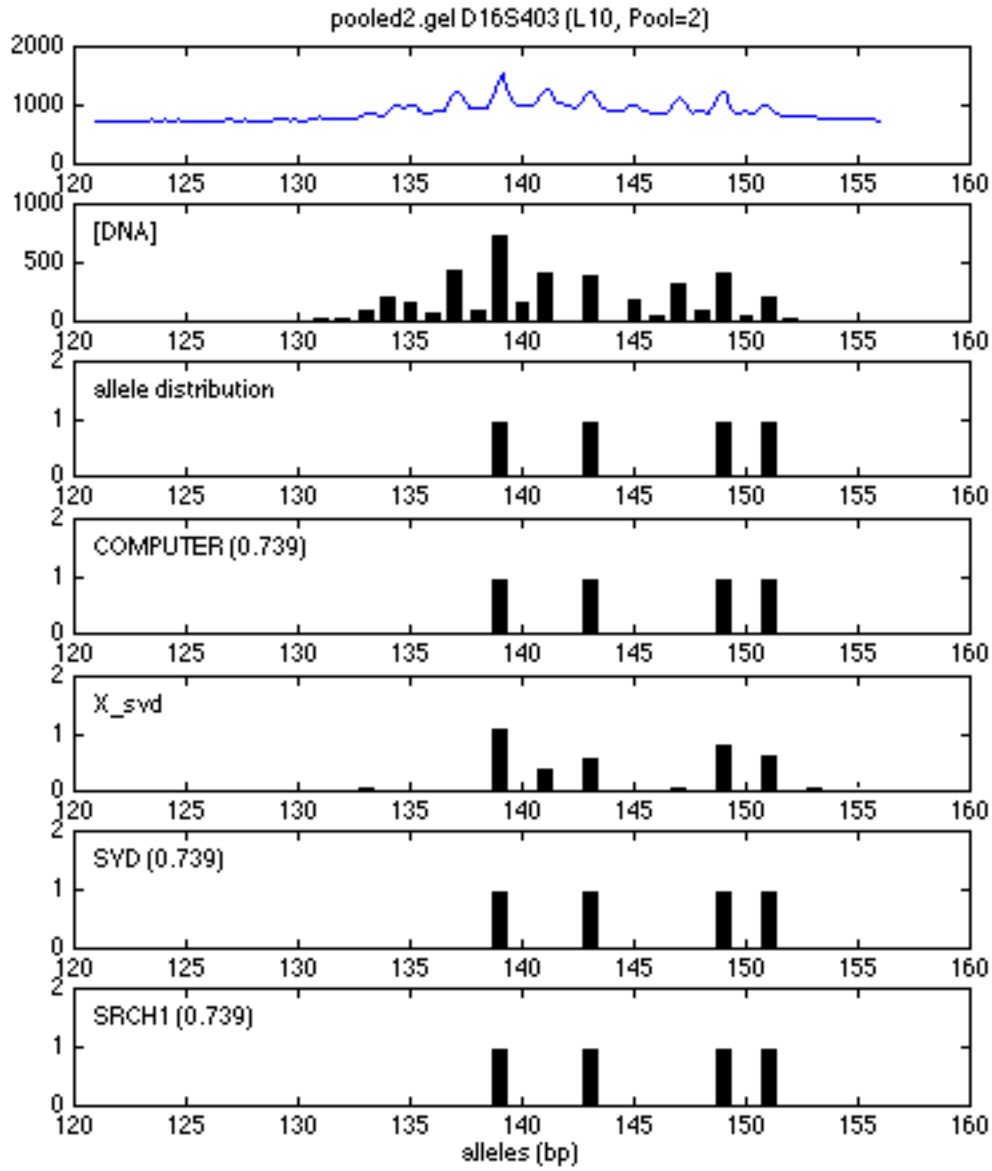


Figure 7.1. Pooled-sample genotyping. Two DNA samples (both heterozygotic) from dinucleotide D16S403 were pooled together and then ran out on an ABI gel. The top pane shows the electropherogram trace for the experiment. The second pane shows the quantitated results, while the third pane shows the *actual* allele distribution for this experiment. The fourth pane (labeled "COMPUTER") shows the computed allele distribution. The fifth pane, labeled "X_svd") shows the original SVD vector from *A/y*, and the pane below it shows the final call by the POOLED_SVD algorithm. The last pane shows the top call by the POOLED_SEARCH algorithm. The numbers in the panes for COMPUTER, SVD, and SRCH1 indicates the computer's confidence measure about the particular result.

We also show the results of POOLED_SVD and POOLED_SEARCH in Figure 7.1. In the pane labeled "X_svd" is the original SVD vector from deconvolving the data vector y with D16S403's stutter matrix A . By iteratively rounding the initial values in X_svd, the resulting integral allele distribution vector computed by POOLED_SVD accurately calls the four alleles, as shown in the pane labeled "SVD". Our POOLED_SEARCH algorithm also ends up with the same allele distribution, as shown in the pane marked "SRCH1".

As a comparison, let us look at the results of one of the 8-sample pools for D16S403 in Figure 7.2. Again, despite the high degree of stuttering, peak overlap, plus-A, and relative amplification artifacts in the data, our pooled GMBD algorithm was able to compute an allele distribution vector (shown in the pane labeled "COMPUTER") quite close to the actual allele distribution (shown in the third pane). In this example, POOLED_SEARCH actually found different solutions (the top three choices were shown in the panes labeled "SRCH1", "SRCH2", and "SRCH3" respectively) from POOLED_SVD. For a final consensus, the computer heuristically selects one of the solutions based on their fit with the data y and the predicted allele distribution A/y (as described in Step 2 of our POOLED_GENOTYPE algorithm). In this experiment, the computer selected the allele distribution computed by POOLED_SVD as the final call.

We have used our pooled GMBD algorithm to determine the allele distributions of pooled samples of up to 96 samples. We report our results for these larger sample pools in the "Results" chapter. Our preliminary results showed that pooled genotyping is indeed possible with GMBD methods. The potential reduction in the required number of laboratory experiments is spectacular: for example, with pools of 100 individuals, there is a 100-fold reduction. With further research on the experimental techniques for generating pooled DNA data in the laboratory, and additional algorithmic refinements to POOLED_GENOTYPE, we may be able to usefully apply this powerful pooled genotyping functionality to allele frequency studies.

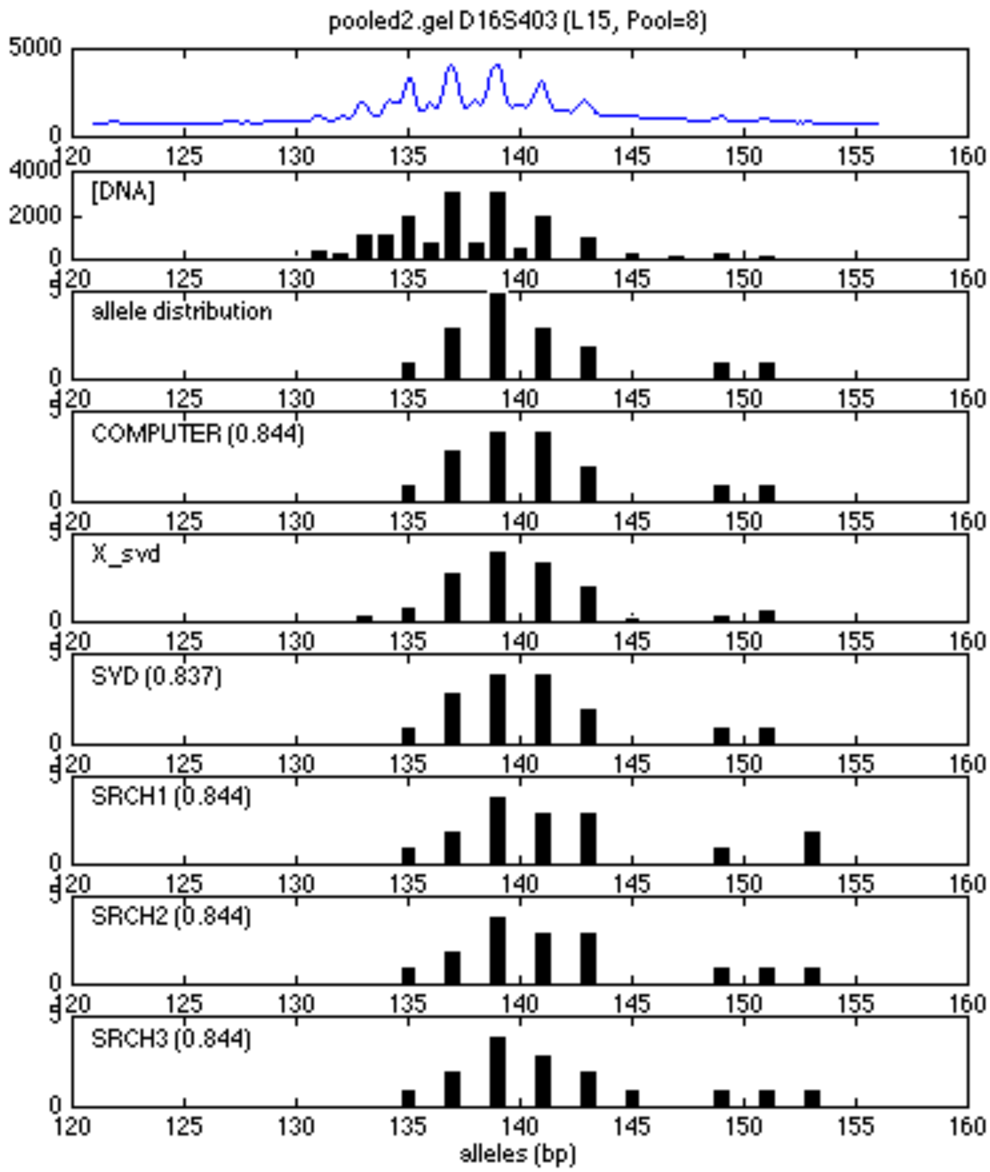


Figure 7.2. Pooling of 8 DNA samples. The top pane shows the electropherogram from an 8-sample pool for marker D16S403. Below it, we show the quantitated results, the actual allele distribution, the computed allele distribution (labeled "COMPUTER"), the original SVD vector (labeled "X_svd"), the allele distribution computed by POOLED_SVD (labeled "SVD"), and the top three allele distributions computed by POOLED_SEARCH (labeled "SRCH1" through "SRCH3").

8. FAST-MAP

We implemented our solution to the microsatellite genotyping problem in a platform-independent automated genotyping system called FAST-MAP[28] (Fluorescent Allele-calling Software Toolkit – Microsatellite Automation Package). Since its public release in 1996, FAST-MAP has been used by geneticists in academic laboratories, government laboratories, biotechnology companies and pharmaceutical companies[29].

In this chapter, we describe our implementation of a practical system that processes real data for real users (who are typically more acquainted with laboratory equipment than computers). We give here an overview of our design and implementation of the FAST-MAP system. Additional details about FAST-MAP are given in the appendices: Appendix B presents FAST-MAP's user environment, Appendix C describes FAST-MAP's libraries, and Appendix D details FAST-MAP's program modules. To help FAST-MAP users in widespread (international) locations, we distributed and maintained the software via the World-Wide-Web at:

`http://www.cs.cmu.edu/~genome/FAST-MAP.html`

8.1. Design

The FAST-MAP system is comprised of three primary components — the programs, the knowledge base, and the user interface:

- The programs constitute the computational engine which automates the requisite tasks in the genotyping problem;
- The knowledge base enables the system to intelligently focus its computation for improved accuracy and efficiency;
- The user interface permits the user to intervene in the fully automated FAST-MAP process, and assist, verify results, or make repairs when necessary.

---

[28]The commercial successor to FAST-MAP is TrueAllele™. For more information on TrueAllele™, see `http://www.cybergenetics-inc.com`.

[29]Groups which have used FAST-MAP include: (1) academic laboratories: Human Genetics Department (Dr. Mike Gorin), University of Pittsburgh; (2) government laboratories: National Center for Human Genome Research (the FUSION group), National Institutes of Health; (3) biotechnology companies: Mercator Genetics, gene/Networks, and deCODE Genetics (Iceland); and (4) pharmaceutical companies: Smithkline Beecham (UK).

Underlying all three components are unifying *semantic objects* which correspond to experimental and genetic entities such as gels, lanes, panels, and markers. These semantic objects are designed into FAST-MAP as the basic units for computation (programs), organization (knowledge base), and presentation (user interface).

8.1.1. Semantic objects

The basic computational unit in FAST-MAP is the *genotyping experiment.* A genotyping experiment is defined by the gel and the lane it was loaded in, and the genetic marker used. In general, the semantic objects in FAST-MAP can be grouped into three categories:

- *experimentative entities* such as gels, lanes, and studies;
- *genetic entities* such as markers, panels, dyes, and size standards; and
- *sampling entities* such as individuals and pedigrees[30].

Figure 8.1 shows the relationships between the semantic objects in FAST-MAP. The semantic objects are related here via the "set_of" relationship and the "has_attribute" relationship. They provide a direct and intuitive mapping between the data structures manipulated by the computer, and the actual entities in the user's world. This semantic connection between user and computer is essential for usability in a non-computer domain.

---

[30]An important characteristic of FAST-MAP is that *no* pedigree information is used to call the alleles — bottom-up computational strategies alone suffice for accurate allele calling. Pedigree information is used solely in FAST-MAP's user interface for presenting genotyping results for user verification.

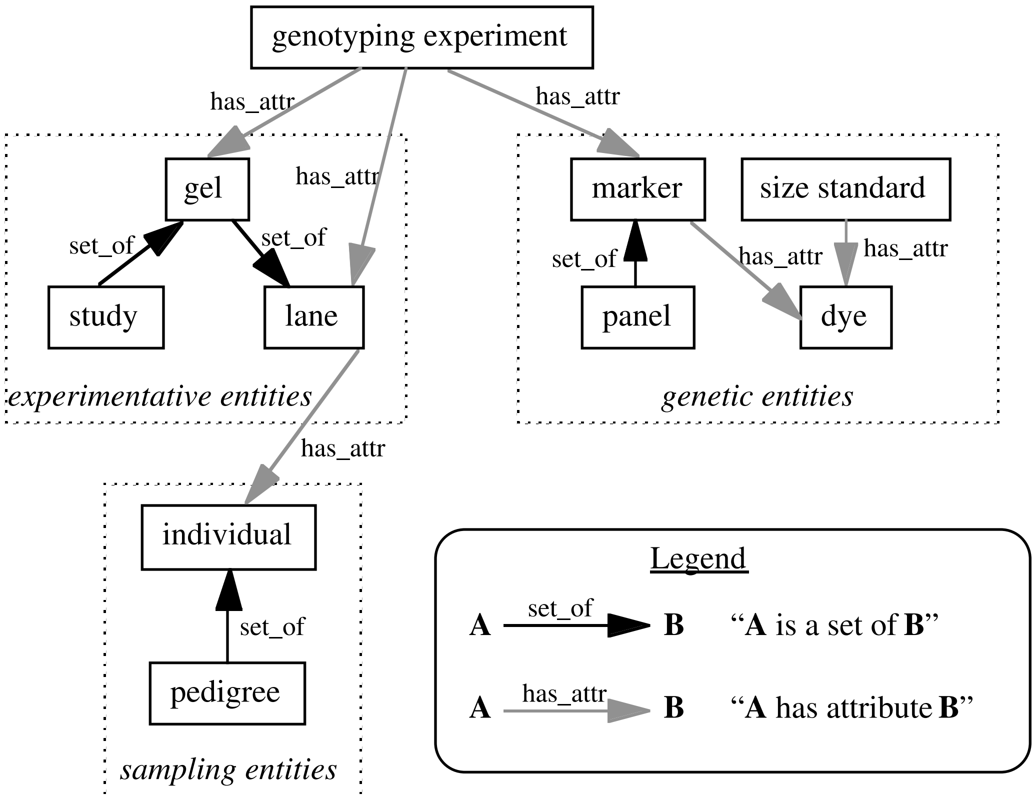


Figure 8.1. Semantic objects in FAST-MAP. The basic computational unit is the "genotyping experiment" semantic object. The other semantic objects are categorized under three groups: experimentative, genetic, and sampling entities. The semantic objects are related via the "set_of" relation and the "has_attribute" relation.

8.1.2. Core programs

For practical genotyping, FAST-MAP includes a comprehensive set of programs for data analysis, visual display, and result outputs. This section describes the computational engine of FAST-MAP — the *core programs,* which implement the various algorithms discussed in the previous chapters. The other FAST-MAP programs are discussed in Appendix D, as well as in the FAST-MAP user manual.

Tasks

Let us recapitulate the three requisite tasks for scoring microsatellite genotyping data:

1. *Data retrieval:* This undoes the multidimensional multiplexing by extracting the gel images and applying a coordinate transformation function:

   $\alpha:$ <x, y> pixels $\leftrightarrow$ <lane, size> points which maps image pixels into molecular units. α*'s* data structure is a sizing grid constructed from MW data;

2. *Data quantitation:* This discretizes the continuous intensity signals into individual stutter bands. These bands are binned with integer allele labels, and quantitated with their relative DNA concentrations:

   β*:* image data + sizing grid α $\rightarrow$ {<bp, concentration>}

3. *Data deconvolution:* This eliminates stutter bands from the quantitated data by deconvolution:

   γ*:* {<bp, concentration>} + genotyping library $\rightarrow$ genotypes

   using automatically constructed genotyping libraries.

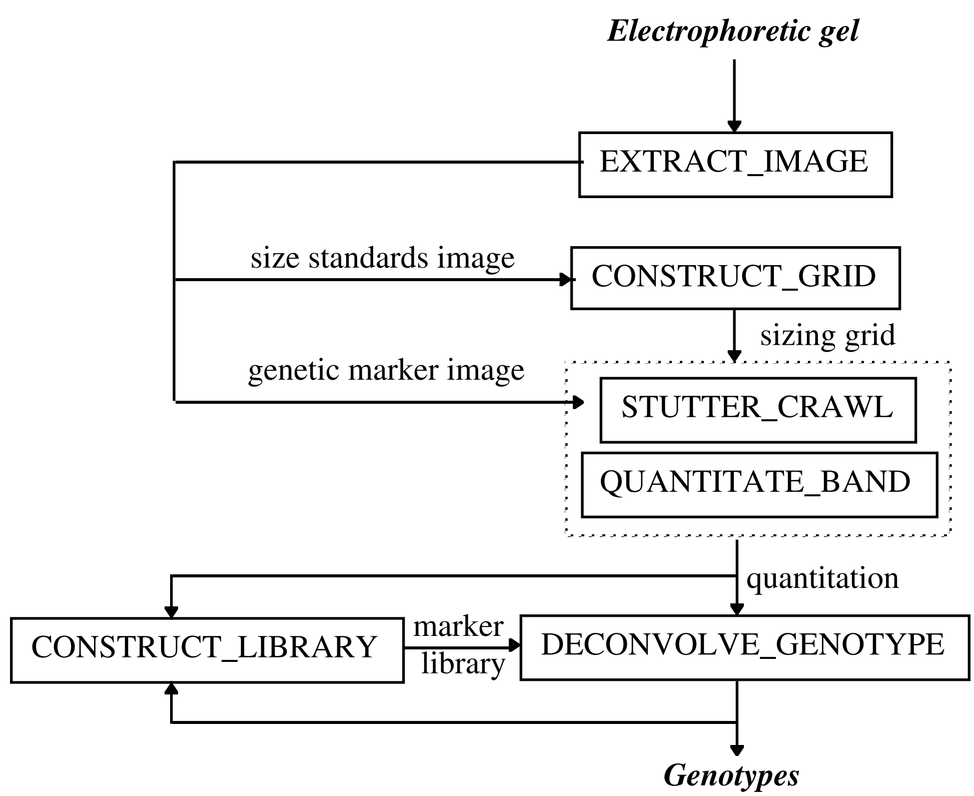


Figure 8.2. The data flow of the automated genotyping algorithms. First, EXTRACT_IMAGE extracts gel images from a raw electrophoretic gel. EXTRACT_IMAGE is specific to the DNA sequencer. Next, CONSTRUCT_GRID builds a sizing grid from the size standards data, which can be used to retrieve marker data for quantitation with STUTTER_CRAWL and QUANTITATE_BAND. Using the marker libraries constructed by CONSTRUCT_LIBRARY, these quantitated data are then genotyped using DECONVOLVE_GENOTYPE. All these algorithms (except EXTRACT_IMAGE) have been discussed in Chapters 4, 5, and 6.

We discussed algorithms for handling the three genotyping tasks in Chapters 4, 5, and 6. For data retrieval, we use CONSTRUCT_GRID (Chapter 4); for data quantitation, we use STUTTER_CRAWL and QUANTITATE_BAND (Chapter 5); for data deconvolution, we use CONSTRUCT_LIBRARY and DECONVOLVE_GENOTYPE (Chapter 6). Figure 8.2 shows the relationship between these algorithms, together with EXTRACT_IMAGE, a dye separation step that is usually provided by the DNA sequencers for data retrieval.

Programs

Corresponding to the data flow presented in Figure 8.2 are three core programs in FAST-MAP (See Figure 8.3):

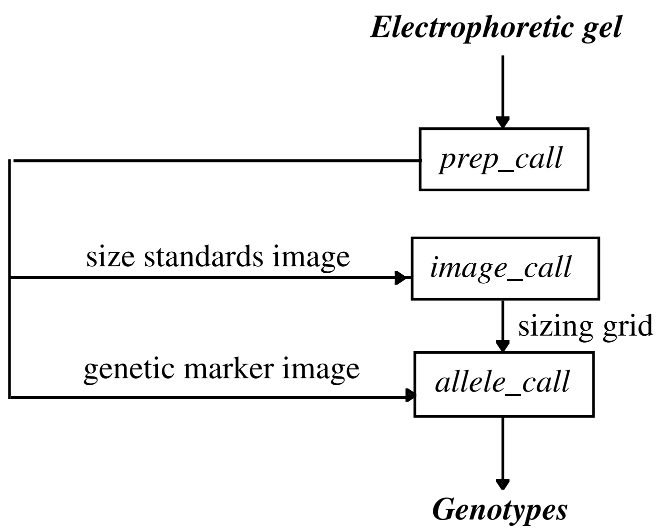


Figure 8.3. Data flow between FAST-MAP's core programs. The core programs consist of: *prep_call,* which prepares sequencer-specific gel data for FAST-MAP by extracting gel images from the raw data, *image_call,* which constructs the sizing grid, and *allele_call*, which calls the alleles for every genotyping experiment on the gel.

- *prep_call:* The program *prep_call* implements the sequencer-specific image extraction (EXTRACT_IMAGE) step. It extracts the image data from the raw gel files generated by different DNA sequencers, and converts the extracted images into a standard format. Since *prep_call* can easily be extended to handle data from new sequencers, FAST-MAP is not restricted to any particular DNA sequencer or data format.

- *image_call:* The program *image_call* implements the construction of the sizing grid (CONSTRUCT_GRID), which performs lane tracking and MW calibration (Chapter 4). The MW sizing grid provides a two-way mapping α between the *observed* image pixel (x, y)-coordinates and the *expected* (lane, base pair)-coordinates. With the sizing grid, the computer automatically extracts the electrophoretic intensity profiles for every lane and fluorescent dye in a gel, reducing the problem from parsing a complex two-dimensional image into analyzing one-dimensional profiles.

- *allele_call:* The program *allele_call* is FAST-MAP's workhorse module. It performs data quantitation and genotyping (both β and γ). For consistent sizing, the program bins all the data bands across multiple gels and assigns internal size labels using STUTTER_CRAWL. The program then quantitates the data bands by finding the best fit between the observed data and the predicted modeling of the bands in the electropherogram. When the data bands from a genotyping experiment are quantitated, *allele_call* applies the marker's stutter and relative amplification information to call the alleles using DECONVOLVE_GENOTYPE. The marker libraries are dynamically learned (CONSTRUCT_LIBRARY) by the computer over multiple gels sharing the same marker panel.

Computation in FAST-MAP is *demand-driven*; a program automatically calls the appropriate precursor program if any intermediate results required by its computation are not available. That is, in terms of the tasks handled by the programs, each of the core programs "subsumes" the program before it:

$$\textit{prep\_call} \subset \textit{image\_call} \subset \textit{allele\_call}$$

where "$\subset$" denotes proper subset inclusion.

For example, if we apply the program *image_call* on a gel that has not been dye-separated, *image_call* will automatically invoke *prep_call* to extract the gel images before constructing the sizing grid. Thus, for *fully* automated processing of high quality data, one can simply invoke the *allele_call* program.

8.1.3. Knowledge base

While the core programs provide computational engines for FAST-MAP, the actual computation is directed by information in FAST-MAP's knowledge base. There are three types of information used:

- *global domain knowledge:* this is general information about genetic entities, such as the possible size range of a marker's alleles, or the molecular sizes (in bp) of the DNA fragments in a MW size standard;
- *specific data knowledge:* this is specific information about genetic experiments, such as the loading pattern of a gel, or the list of component gels in a study.
- *pattern matching knowledge:* this is pattern-matching knowledge that FAST-MAP has learned from the data it has processed. This learned information includes (a) marker- and allele-specific stutter patterns, (b) relative amplifications, and (c) binning information.

Figure 8.4 depicts the information sources forming FAST-MAP's knowledge.

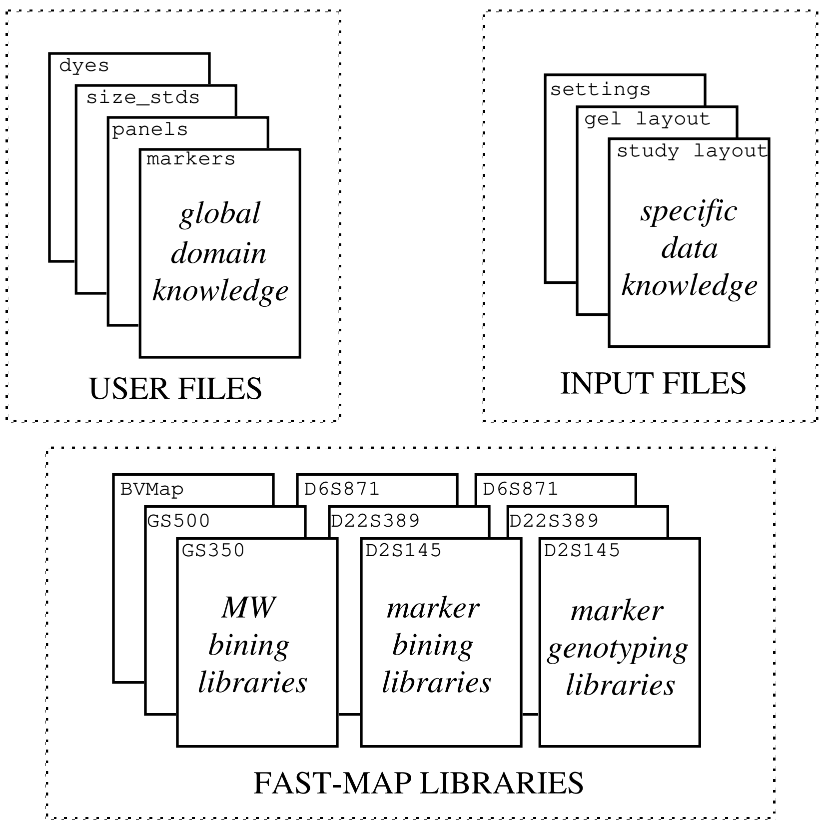


Figure 8.4. FAST-MAP's knowledge base. There are three different sources of information: the "user files" provide global general knowledge about the genetic domain, the "input files" provide specific information about the genetic experiments, and the "library files" provide pattern matching information learned by FAST-MAP from processing previous data.

Global domain knowledge

To search the MW bands in size standard data, FAST-MAP minimally needs to know the DNA fragment sizes used in the size standard. To genotype microsatellite markers, FAST-MAP needs to know general characteristics of the markers, such as which fluorescent dye is tagged on the marker, the size range of possible alleles, and the number of base pairs in a repeat unit. This general knowledge of genetic entities is supplied by the user to FAST-MAP by annotating a shared database of genetic knowledge called the *user files*:

- *dyes*: contains aliases for the various fluorescent dye names used in the sequencers, for example:

```
dye name        aliases

blue            fam
green           tet
yellow          hex
red             tam tamra rox
```

- *size_stds:* contains information about the size standards, such as the dye tagging the DNA fragments, and the sizes of these fragments. For example:

```
name   dye   sizes

% Genescan 500
GS500  TAM   35 50 75 100 139 ... 400 450 490 500

% Bioventure's 20-bp ladder
BVMap  TAM   70 80 90 100 120 140 ... 360 380 400
```

- *markers:* contains genetic microsatellite marker information, such as allele size range, fluorescent dye, repeat type, and whether there has been any special experimental handling. For example:

```
name       min    max    dye     repeat   plusA      ladder

% Markers for "panel1"
%
p1m1       103    161    FAM     2        enhance    yes
p1m2       316    366    FAM     2        enhance    yes
p1m3       176    224    HEX     2        enhance    yes
p1m4       290    326    HEX     2        enhance    yes
p1m5       178    238    TET     2        enhance    yes
```

- *panels:* contains definitions of panel names. Each marker panel is defined by a list of genetic markers used in that panel. For example:

```
name      markers

panel1    p1m1 p1m2 p1m3 p1m4 p1m5
panel1a   p1m1 p1m2
```

There are other user files that contain global (but non-genetic) information. For a complete list of the user files and their detailed descriptions, please see Appendix B.1.

Specific data knowledge

When human genotypers call alleles on gel data, in addition to using general genetic knowledge, they also apply considerable contextual information, such as the design of the study and the layout of the marker multiplexing. Similarly, the *allele_call* computer program needs such information for automated genotyping.

In FAST-MAP, the user supplies specific data knowledge by providing *input files* that annotate each gel or study. For a gel, there are two user-annotated input files:

• *settings:* contains gel-specific annotations and execution preferences. For example:

```
attribute                    value

% Section 1. Gel-specific settings.
%
Gel_file_name                R211a.gel
Matrix_file_name             MATR211
Sample_file_name             R211a.samples
Sequencer_type               ABI
Number_of_lanes              34
Size_standard                BVMap
Min_size_standard            70
Max_size_standard            400
Panel_name                   panel1
Experiment_condition
Noise_threshold              50

% Section 2. Program animation settings
%
Verbose_mode_on              yes
Show_plots                   no

% Section 3. Allele_call settings
%
Redo_import_planes           no
Redo_manifold                no
Redo_quantitation            no
Redo_allele_calling          no
Analyze_noisy_data           yes

% Section 4. Allele_results settings
%
Output_noise_genotypes       no
Output_sort_by               markers
Latest_sample_only           no
Round_evenly_spaced          no

% Section 5. Allele_view settings
%
Prioritize_results           worst_first

% Section 6. Marker_view settings
%
Lanes_per_view               5

% Section 7. Allele_printout settings
%
Send_to_printer              yes
Show_figure                  no
Rows_per_page                4
Columns_per_page             2
Include_electro_plots        no
```

- *layout:* tells FAST-MAP the design of the gel readout experiment — where to find the individual genotype experiments on the gel. This information tells FAST-MAP whether a lane is loaded, and (if so) with which panel and size standard;

```
lane      sample      panel       size std

1         ped01P1     panel1      BVMap
2         ped01P2     panel1      BVMap
3         ped01C1     panel1      BVMap
4         ped01C2     panel1      BVMap
5         ped01C3     panel1      BVMap
6         ped01C4     panel1      BVMap
7         ped01C5     panel1      BVMap
8         ped02P1     panel1      BVMap
9         ped02P2     panel1      BVMap
10        ped02C1     panel1      BVMap
...
32        ped05C4     panel1      BVMap
33        blank       blank       blank
34        blank       blank       blank
```

For each study, there are two user-annotated input files:

- *settings:* contains study-specific execution preferences similar to those in a gel's "settings" file;
- *layout:* contains a list of nicknames of the study's component gels, such as:

```
gel

% Gels for panel1 study
%
GEL1      % 4/15/97, Ped01-Ped06
GEL2      % 4/16/97, Ped06-Ped11
GEL3      % 4/18/97, Ped12-Ped17
```

For a detailed description of the input files of gels and studies, please see Appendix B.2.

Pattern matching knowledge

When microsatellite markers are PCR amplified and size separated under the same experimental conditions, the band patterns and artifacts are reproducible. Thus, the reproducible patterns of prior experiments can be used to computationally remove such artifacts from subsequent experiments. In FAST-MAP, this pattern-matching knowledge is automatically learned and compiled into internal files called *libraries.* As FAST-MAP processes more data, it augments the appropriate library files with additional pattern matching information to improve the accuracy of its predicted patterns.

As shown in Figure 8.4, there are three types of libraries maintained by FAST-MAP:

- *MW binning libraries:* these contain the relative pixel locations of the MW bands found on previous gels. FAST-MAP uses this information to construct expectations of where to find MW bands on a new gel;
- *marker binning libraries:* these contain the relative pixel information on markers that FAST-MAP has learned from analyzing previous gels. FAST-MAP uses this binning information to guide its search for marker bands on the current gel, and to assign consistent integer allele labels to detected marker bands;
- *marker genotyping libraries:* these contain information on stutter patterns and relative amplification ratios observed from previously analyzed marker data, which FAST-MAP uses to deconvolve genotype data into alleles.

Appendix C describes the various library files in detail.

8.1.4. User interface

While it is FAST-MAP's primary goal to *fully automate* microsatellite genotyping, it is equally important to allow for user interaction, verification, and assistance when the data are less than perfect. Instead of a system which processes data fully autonomously (and perhaps autocratically) from beginning to end, we provide optional "checkpoints" in FAST-MAP between major computation steps. At these checkpoints, the user can verify and repair intermediate results, if necessary:

$$prep\_call \xrightarrow{prep\_view} image\_call \xrightarrow{\substack{image\_view \\ marker\_view}} allele\_call \xrightarrow{\substack{allele\_view \\ family\_view}} results$$

Because of the visual nature of the genotyping task, we provide graphical user interfaces[31] at each checkpoint to facilitate data review and editing:

- *prep_view:* this program allows the user to preview the extracted gel image and verify that the layout information supplied to FAST-MAP is accurate. The user can also assist FAST-MAP by pruning confusing regions (e.g., primer regions) away from the gel image;

- *image_view:* this program allows the user to quickly verify the accuracy of the sizing grid constructed by *image_call* by viewing the sizing grid overlaid on top of the size standards image. If the computer made some mistakes, the user can repair the grid;

- *marker_view:* this program allows the user to view the allele size windows of the markers on a gel at a single glance. The user can quickly verify that the marker information supplied to FAST-MAP is correct. If not, the user can repair by graphically expanding or contracting the allele windows;

- *allele_view:* this program graphically visualizes *allele_call*'s genotyping results. The user can repair the computer's mistakes by editing erroneous genotypes;

- *family_view:* when supplemental pedigree information is provided, *allele_view* can provide *family_view* functionality. The user can view the genotypes of all individuals in a family displayed in one window, even when the corresponding genotyping

[31]FAST-MAP's data viewing and editing graphical interfaces were implemented by Nandita Mukhopadhyay.

experiments are on different gels. This turns out to be a very useful tool for quickly (visually) detecting non-Mendelian inheritance.

We detail these viewing programs later on with an example in Section 8.3.

8.2. Implementation

FAST-MAP requires (1) high performance numeric computation for its core programs, (2) fast data retrieval for its knowledge base, (3) efficient graphic visualization for its user interface, and (4) computer platform independence for its system portability. To satisfy these requirements, we implemented FAST-MAP in the MATLAB computing environment (MathWorks, 1993)[32]. MATLAB is a high performance computing environment that integrates numerical analysis, matrix computation, signal processing, and graphics rendering. Moreover, MATLAB is platform-independent, and runs on UNIX, Macintosh, and PC systems.

Fast data retrieval

To attain fast data retrieval for FAST-MAP's large (and increasing) knowledge base, we adopted a two-prong approach:

1. Frequently used data (such as the global domain knowledge stored in the "user files") are loaded into memory as part of FAST-MAP's operating environment. This allows rapid access directly from the system's RAM, instead of from the user files. However, this implementation also requires extra effort to maintain consistency between the system's environment and the actual contents in the user files, since the user can dynamically update the contents of the user files. To detect user updates, we provided the user with an internal "edit" function.

2. Less frequently used data (such as the amount of display data for the genotyping experiments) are not loaded into the system because of memory limitations. To facilitate rapid access from data files, FAST-MAP's processing decomposes the display data into the smallest possible units, and stores the units as separate small files indexed

[32]As of this writing, FAST-MAP is implemented in MATLAB v4.2c.

by descriptive file names. Indexing by these file names emulates pseudo-random accessing with direct file retrievals.

Demand-driven computation

To implement demand-driven computation, FAST-MAP continually stores the partial results in small files (uniquely named for rapid retrieval) as the computation proceeds. By incrementally saving the partial state of the computation, FAST-MAP allows the user to halt a long-running analysis at any point, and resume from where it left off without losing previously computed partial results.

Programming language-specific optimizations

FAST-MAP is computationally intensive, and requires efficient algorithm implementations. Many of the algorithms (e.g. SVD deconvolution) are naturally specified as matrix computations, and MATLAB is highly optimized for such matrix processing. For other algorithms, simple algorithmic optimizations that cater to the matrix-based computation in MATLAB bring about significant improvements in performance. For example, using radix search (which takes advantage of the fixed-width characteristic of items stored in a matrix) and vectorizing the code has yielded significant system speed-ups.

8.3. Execution

After setting up a study (say, STUDY1) for analysis in FAST-MAP, we can process the data in a single step, by typing:

```
>> allele_call STUDY1
```

FAST-MAP would then perform the necessary automatic lane/size calibration and genotyping without interruption. This fully-automated "single-step" mode is suitable for high-throughput centers that routinely generate high quality gel data.

With less-than-perfect data, it is worthwhile to verify partial results at various FAST-MAP checkpoints. By repairing any errors in the intermediate results using FAST-MAP's graphical viewing programs, the user can avoid unnecessary re-computations and improve the quality of the final genotyping results.

The best sequence in which to run the FAST-MAP programs depend on the availability of certain data and knowledge[33]. Figure 8.5 depicts the dependency chart for a typical scenario of processing gels with "standard" markers (i.e., markers we have analyzed before). The dependency chart is divided into two sections, an *obligatory* "calling" section and an *optional* "viewing" section. The obligatory section is shown in the upper half of the figure, with the suggested calling sequence of:

$$prep\_call \rightarrow image\_call \rightarrow allele\_call$$

[33]In TrueAllele™, the commercial successor to FAST-MAP, the dependency between the programs is automatically generated and customized based on the particular scenario. This dependency relationship is presented as a flow chart consisting of clickable buttons to direct the user to invoke programs in the best sequence.

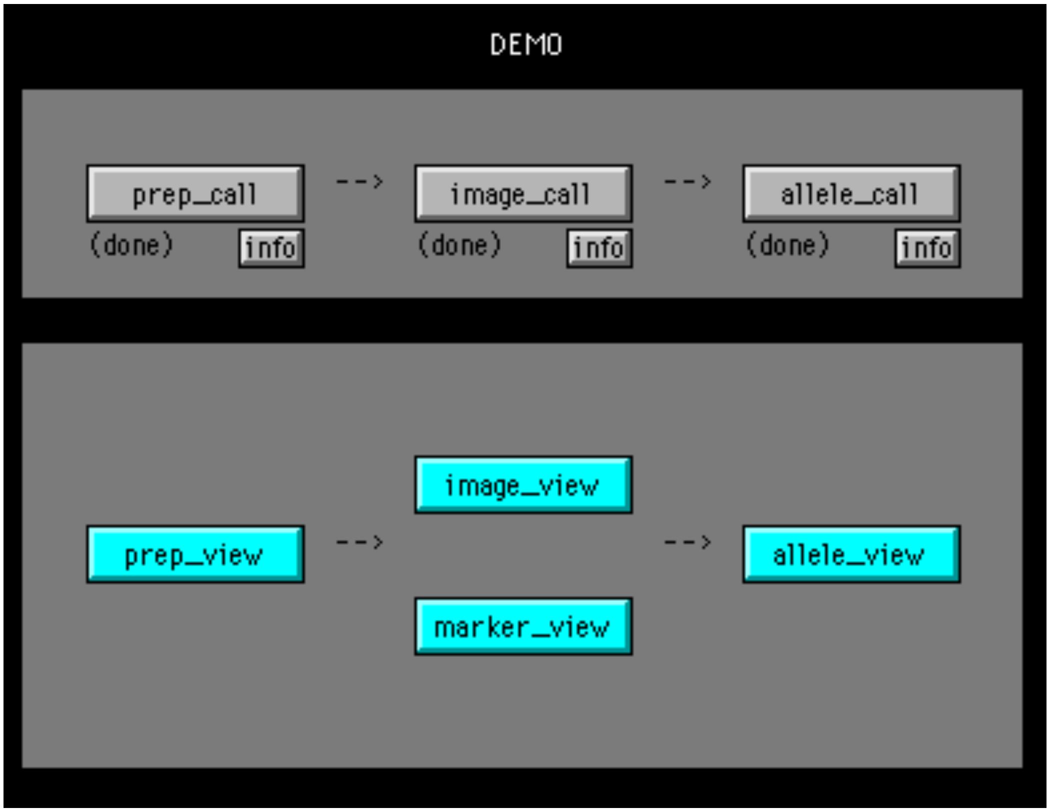


Figure 8.5. Dependency chart of the programs to call in FAST-MAP. If the data is relatively perfect and there is no need for the user to verify intermediate results, then one needs only run *allele_call.* Otherwise, the user may elect to call intermediate programs (in the top pane), and optionally use the viewing programs (in the bottom pane) to check the intermediate results before proceeding further.

The bottom half of the figure shows optional viewing programs that one can interactively verify and repair intermediate results. For example, one can call the prep_view program to view the images extracted by *prep_call*, or invoke the *image_view* program to review the sizing grid constructed by *image_call,* or use the *marker_view* program to verify that the allele window information supplied to the system is correct. After *allele_call*, one invokes the *allele_view* program to review the computed genotypes, and edit any miscalls.

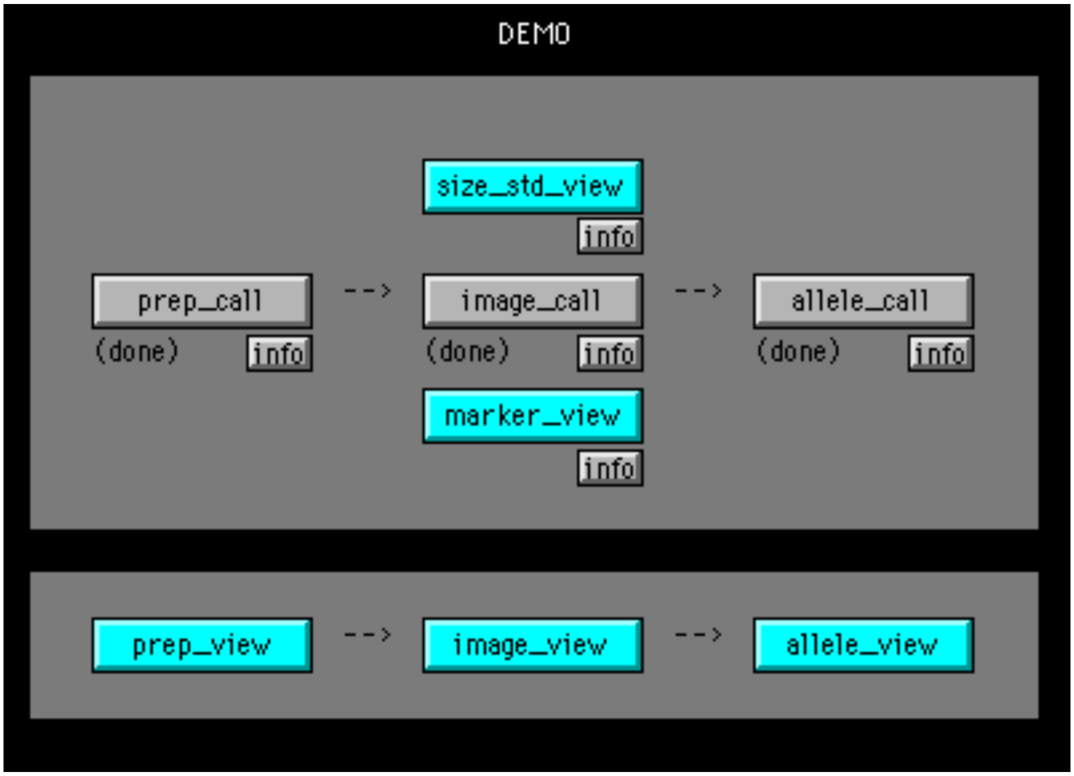


Figure 8.6. Dependency chart of FAST-MAP programs for another scenario. The top pane lays out the *obligatory* programs to call, with the program dependencies displayed from left to right, and top to bottom. The bottom pane shows the optional viewing programs at the various programmatic checkpoints for the user to verify and correct intermediate results.

By creating the necessary initial libraries and verifying that the information supplied to the computer is correct, the user can greatly assist the computer in analyzing new data. In Figure 8.6, we show the program dependencies when a new MW size standard and a new marker panel have been used. Since there are no pre-existing MW binning libraries for FAST-MAP to apply in its sizing grid construction, the user assists by invoking the *size_std_view* program (after extracting the gel image with *prep_call*) to create a MW binning library for the new MW size standard. With this user defined size standard library, FAST-MAP can proceed to calibrate the gel data in molecular sizes with *image_call*. Before calling the *allele_call* program, the user should also use the *marker_view* program to verify that the correct allele windows have been supplied to the computer, since the allele window information for a new marker panel from the public databases is often incorrect.

Example

For discussion, let us go through the multiple steps in analyzing the demonstration gel (that we include with the FAST-MAP package) in the cautious mode. First, we invoke the program *prep_call* to extract the gel images from the input gel file (`demo.gel`) into a sequencer-independent format. Here is a trace of the program on `demo.gel`:

```
demo.gel: prep_call [4-Dec-97 12:16:36]
--------

Image file demo.gel is an ABI/377 collection file (version 2.00).
demo.gel is a 5720 x 194 gel with 4 dye planes.

Reading gel data from demo.gel.........................:.:.:.:.Done.
Adjusting image contrast........Done.
```

This image conversion step typically takes about several minutes to complete, and it prepares the image for viewing in FAST-MAP using the *prep_view* interface (Figure 8.7). There are three things that the user can do in *prep_view* that can help FAST-MAP in its subsequent processing:
(1) Cropping the extraneous regions of the gel image (e.g. the primer region) by setting the minimum and/or maximum scan numbers;
(2) Verifying the actual range of molecular weight standards captured on the gel; and
(3) Confirming that the loaded lanes specified in the gel's layout file is correct.

For example, to crop away the primer region (the bright bands at the bottom of the gel shown in Figure 8.7), the user can click on the "Min Scan" button in the *prep_view* window. In a zoomed-in image (shown in Figure 8.8), the user can then define the minimum scan line for the gel by simple mouse clicks. This lets FAST-MAP ignore the gel region below the specified minimum scan line, hence avoiding the distractingly bright primer bands when calibrating the MW bands.

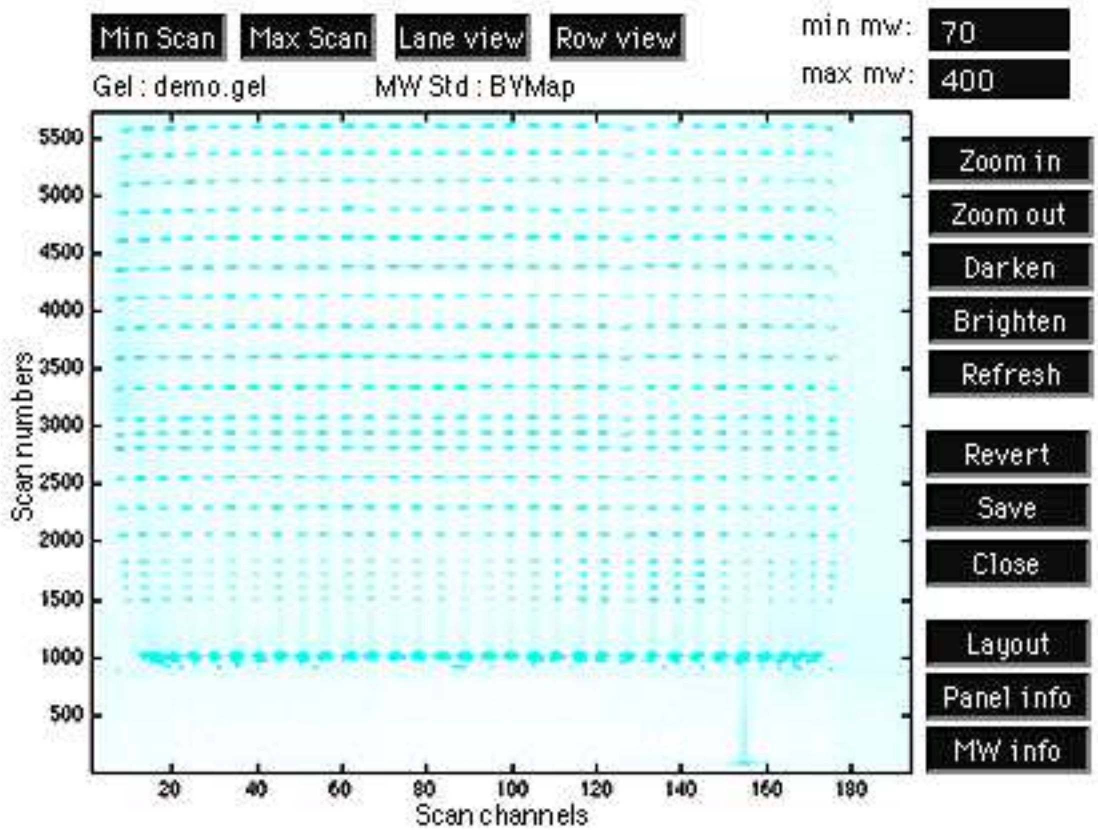


Figure 8.7. Viewing a gel image in *prep_view*. After *prep_call* extracts the gel image data from raw gel files, the user reviews the gel image in *prep_view*.

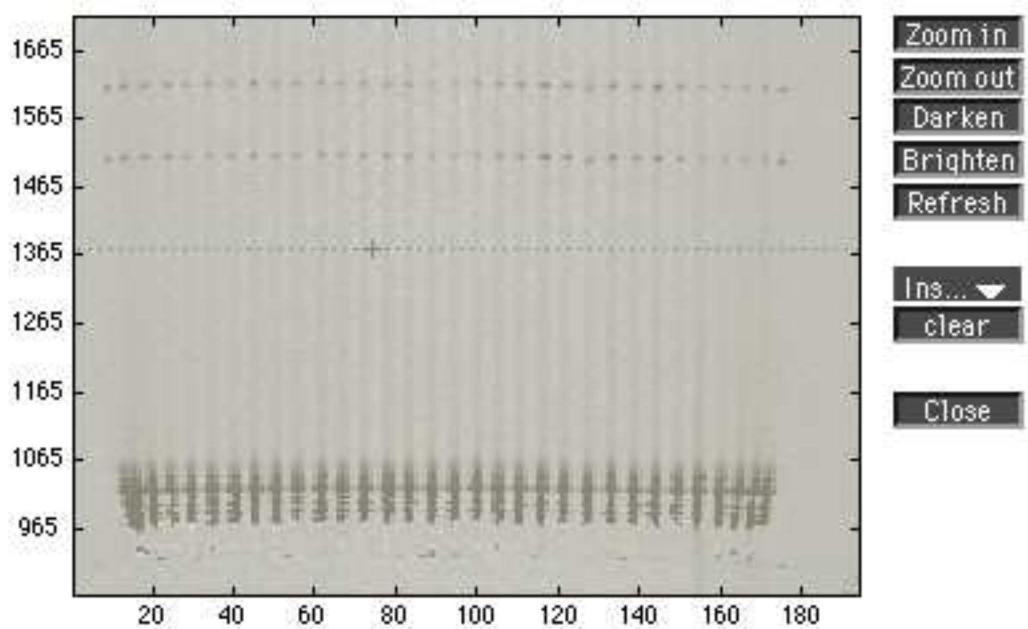


Figure 8.8. Cropping away primer region in *prep_view*. To crop the primer region away from the gel, the user click at one or more minimum scan locations (shown here as a "+"). The computer automatically extends the clicks into a minimum scan line (shown as a horizontal dotted line) separating the primer region from the data.

Once we have verified that the gel is of good quality, and have confirmed that the sizing information supplied to FAST-MAP is correct, FAST-MAP can proceed with its automatic sizing grid construction in the program *image_call:*

```
demo.gel: Image_call [4-Dec-97 12:23:46]
--------
Reading image data for dye 4...Done.
Scanning for gel lanes.......___+++++.::::Done.
Scanning for size std bands....______+++++++..::::::::::::::::Done.
Refining MW peaks .................................Done.
Computing pixel-to-bp calibration ...........................Done.
Saving size calibration...................................Done.
Reading image data for dye 1...Done.
Converting image to profiles..............................Done.
Reading image data for dye 2...Done.
Converting image to profiles..............................Done.
Reading image data for dye 3...Done.
Converting image to profiles..............................Done.
Reading image data for dye 4...Done.
Converting image to profiles..............................Done.
```

As shown in the program trace above, FAST-MAP first tracks the lanes, and then performs MW size calibration on each lane. When the MW sizing grid is constructed, the user can use the interface *image_view* to verify that the sizing grid is correct, and repair any mistakes that may have been made by FAST-MAP. Figure 8.9 shows the sizing grid constructed by FAST-MAP on `demo.gel` as viewed in *image_view.*

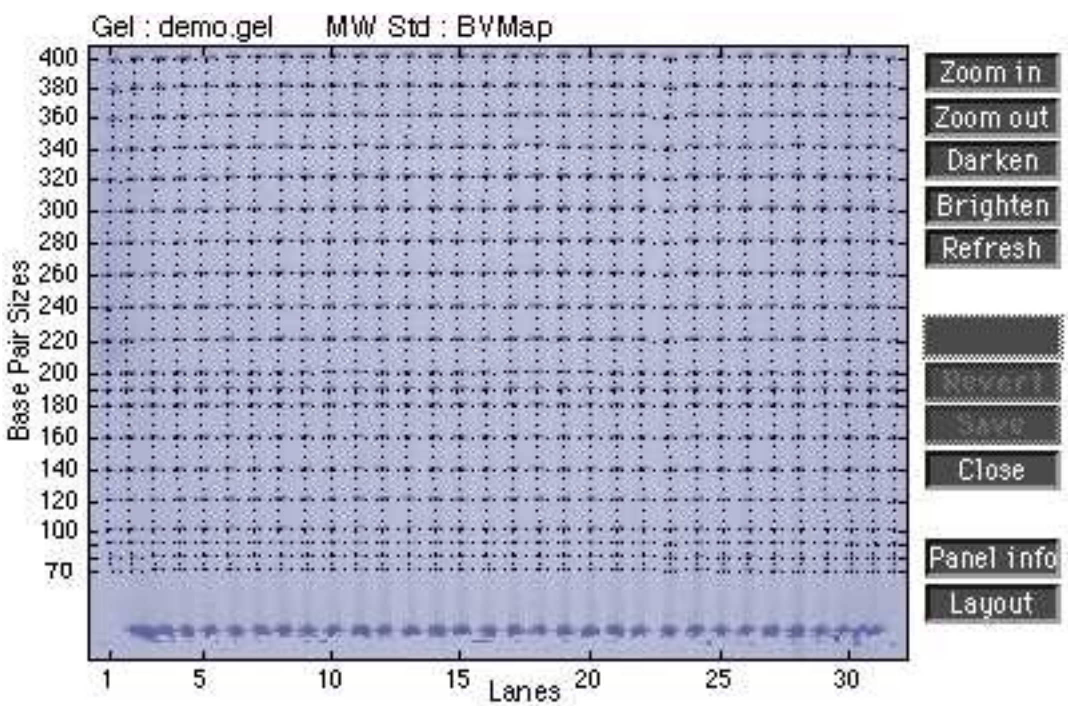


Figure 8.9. Viewing computed sizing grid. In *image_view,* the sizing grid is overlaid on the gel image, allowing the user to verify the correctness of the grid at one glance.

Typically, if we are running a marker panel for the first time, we would obtain the initial allele windows from public databases (e.g. the CEPH database), which may not include all the alleles in the samples for our study. Since FAST-MAP relies on the allele window information for extracting data from electropherograms for analysis, it is important to make sure that the marker information supplied to FAST-MAP is accurate. The user can use the *marker_view* interface to quickly scan through the markers on the gels in a study, and define correct allele windows. Figure 8.10 shows the *marker_view* interface for one of the markers on `demo.gel`.

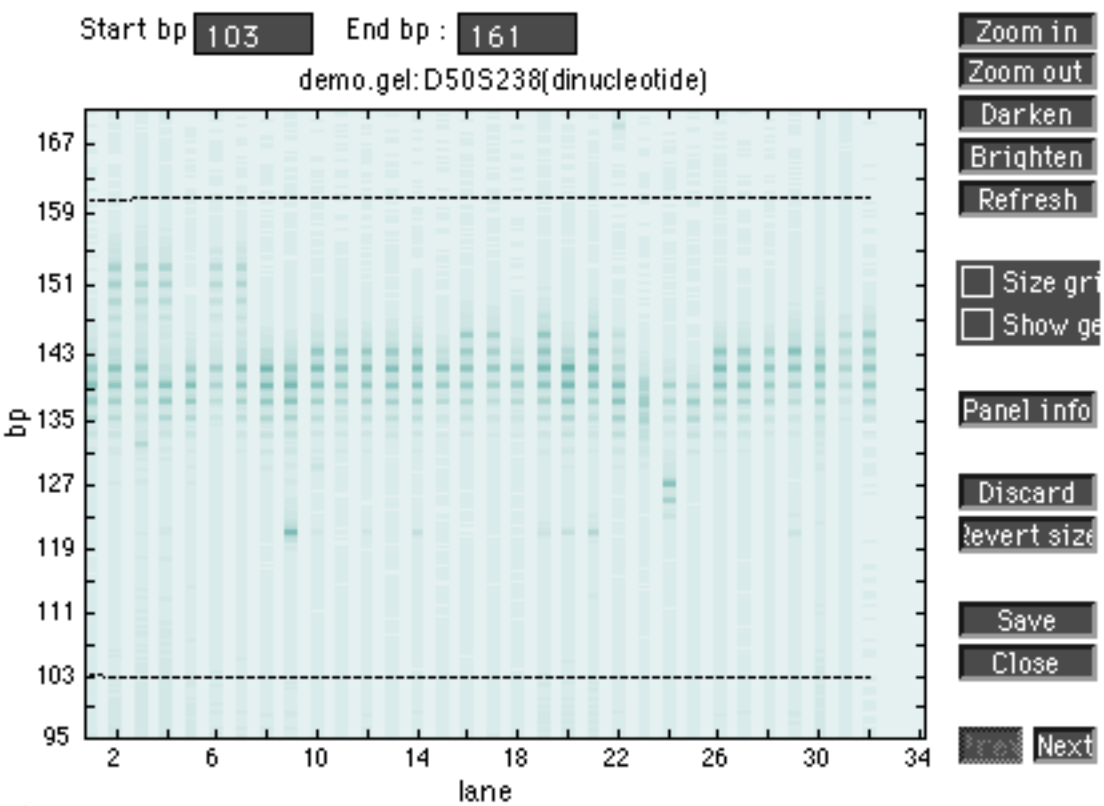


Figure 8.10. Determining allele windows for markers. In *marker_view,* marker bands from different gel lanes are aligned in a single window, allowing the user to quickly verify that the specified allele size window includes all the marker alleles locally.

Once we have verified that the sizing grid and the markers' allele size window information are correct, FAST-MAP can proceed with automated allele calling. It is worthwhile to confirm that the intermediate results are accurate (as we have been doing thus far) before proceeding with this computation-intensive process, as there may be thousands of genotyping experiments to be scored. From this point on, however, the allele calling process is fully automated and requires no further user assistance, even on difficult data. As an example, we show the complete program trace for the first marker in `demo.gel:`

```
Marker D50S238  (1 of 5 markers in CHROM50): [4-Dec-97 12:40:07]
------

Scanning demo.gel for marker windows...........................Done.
Scanning demo.gel for marker bands.............................Done.
Binning marker bands...................................Done.
Quantitating D50S238 demo.gel lane 1.....a.b..Done.
Quantitating D50S238 demo.gel lane 2.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 3.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 4.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 5.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 6.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 7.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 8.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 9.....a.b..Done.
Quantitating D50S238 demo.gel lane 10.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 11.....a.b..Done.
Quantitating D50S238 demo.gel lane 12.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 13.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 14.....a.b..Done.
Quantitating D50S238 demo.gel lane 15.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 16.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 17.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 18.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 19.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 20.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 21.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 22.....a.b..Done.
Quantitating D50S238 demo.gel lane 23.....a.b..Done.
Quantitating D50S238 demo.gel lane 24.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 25.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 26.....a.b..Done.
Quantitating D50S238 demo.gel lane 27.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 28.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 29.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 30.....a.b.a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 31.....a.b.a.b..Done.
Quantitating D50S238 demo.gel lane 32.....a.b.a.b.a.b..Done.
Estimating genotypes.........................................
.........................................Done.
Using the following 23 estimated genotypes to construct initial
stutter library for D50S238:
   demo.gel Lane  1 : <139, 139> (qual = 0.94)
   demo.gel Lane  5 : <139, 141> (qual = 0.95)
   demo.gel Lane  6 : <141, 153> (qual = 0.93)
   demo.gel Lane  8 : <141, 141> (qual = 0.94)
   demo.gel Lane  9 : <121, 139> (qual = 0.92)
   demo.gel Lane 11 : <141, 143> (qual = 0.94)
   demo.gel Lane 12 : <141, 143> (qual = 0.94)
   demo.gel Lane 13 : <141, 141> (qual = 0.93)
   demo.gel Lane 14 : <141, 141> (qual = 0.92)
   demo.gel Lane 15 : <141, 141> (qual = 0.95)
   demo.gel Lane 17 : <141, 141> (qual = 0.93)
   demo.gel Lane 18 : <141, 141> (qual = 0.95)
   demo.gel Lane 19 : <141, 141> (qual = 0.93)
   demo.gel Lane 20 : <141, 141> (qual = 0.95)
   demo.gel Lane 22 : <139, 139> (qual = 0.94)
```

```
   demo.gel Lane 24 : <127, 139> (qual = 0.93)
   demo.gel Lane 25 : <137, 139> (qual = 0.96)
   demo.gel Lane 26 : <141, 143> (qual = 0.95)
   demo.gel Lane 27 : <141, 143> (qual = 0.94)
   demo.gel Lane 28 : <141, 143> (qual = 0.94)
   demo.gel Lane 29 : <143, 143> (qual = 0.94)
   demo.gel Lane 30 : <141, 141> (qual = 0.93)
   demo.gel Lane 32 : <141, 145> (qual = 0.95)
Bootstrapping initial library A using estimated genotypes...........
.........:....................:........................Done.
Genotyping D50S238 demo.gel lane 1....Done.
Genotyping D50S238 demo.gel lane 2....Done.
Genotyping D50S238 demo.gel lane 3....Done.
Genotyping D50S238 demo.gel lane 4....Done.
Genotyping D50S238 demo.gel lane 5....Done.
Genotyping D50S238 demo.gel lane 6....Done.
Genotyping D50S238 demo.gel lane 7....Done.
Genotyping D50S238 demo.gel lane 8....Done.
Genotyping D50S238 demo.gel lane 9....Done.
Genotyping D50S238 demo.gel lane 10....Done.
Genotyping D50S238 demo.gel lane 11....Done.
Genotyping D50S238 demo.gel lane 12....Done.
Genotyping D50S238 demo.gel lane 13....Done.
Genotyping D50S238 demo.gel lane 14....Done.
Genotyping D50S238 demo.gel lane 15....Done.
Genotyping D50S238 demo.gel lane 16....Done.
Genotyping D50S238 demo.gel lane 17....Done.
Genotyping D50S238 demo.gel lane 18....Done.
Genotyping D50S238 demo.gel lane 19....Done.
Genotyping D50S238 demo.gel lane 20....Done.
Genotyping D50S238 demo.gel lane 21....Done.
Genotyping D50S238 demo.gel lane 22....Done.
Genotyping D50S238 demo.gel lane 23....Done.
Genotyping D50S238 demo.gel lane 24....Done.
Genotyping D50S238 demo.gel lane 25....Done.
Genotyping D50S238 demo.gel lane 26....Done.
Genotyping D50S238 demo.gel lane 27....Done.
Genotyping D50S238 demo.gel lane 28....Done.
Genotyping D50S238 demo.gel lane 29....Done.
Genotyping D50S238 demo.gel lane 30....Done.
Genotyping D50S238 demo.gel lane 31....Done.
Genotyping D50S238 demo.gel lane 32....Done.
Updating stutter library........................................Done.
Done processing marker D50S238 at 4-Dec-97 13:01:34.

(Note: D50S238 is marker 1 of 5 markers in panel CHROM50 for DEMO.)


Marker D50S215  (2 of 5 markers in CHROM50): [4-Dec-97 13:01:34]
------

Scanning demo.gel for marker windows............................Done.
Scanning demo.gel for marker bands..............................Done.
Binning marker bands............................................Done.
Quantitating D50S215 demo.gel lane 1.....a.b..Done.
Quantitating D50S215 demo.gel lane 2.....a.b..Done.
:
:
```

Since the computer requires no further user attention in *allele_call*, the user can leave it to analyze the data (say, overnight) unattended if there are many gels to be processed. On an Apple Power Macintosh with a 266 MHz PowerPC™ G3 processor, the total time taken to process a 32-lane ABI gel with 5 markers (such as `demo.gel`) is about 45 minutes. We show, in Table 8.1, an approximate breakdown of the total time spent in processing the various computational tasks. As shown in the table, much of FAST-MAP's computation (44%) was spent in quantitating the data bands, followed by determining the alleles (23%) and constructing the sizing grid (22%). Because of FAST-MAP's highly quantitative expectation-based approach, the actual total running time is proportional to the quality of the gel data, and may vary from gel to gel.

Task	Program	Time	Percentage
Gel image extraction	*prep_call*	5 min/gel	11 %
Sizing grid construction	*image_call*	10 min/gel	22 %
Data band quantitation	*allele_call*	5-10 sec/experiment	44 %
Allele determination	*allele_call*	3-5 sec/experiment	23 %
Total		45 min/gel	100 %

Table 8.1. Approximate FAST-MAP running times for a 32-lane 5-marker ABI gel (e.g. `demo.gel`) on an Apple Power G3/266 MHz Macintosh.

When FAST-MAP has finished analyzing the data, the user can return to review the results using the *allele_view* program. Because FAST-MAP uses expectation-based analysis, by comparing the data with the expectations, the computer can reliably evaluate its own allele calls. Therefore, in addition to calling the alleles, FAST-MAP assigns a confidence measure to each of its allele calls, and sorts the genotypes in terms of their confidence values. This intelligent sorting functionality allows the user to review the results selectively, rather than exhaustively. There are numerous sorting orders available in *allele_view* for the user to selectively review genotyping results (Figure 8.11). For example, the user may view the results in the "worst-first" order, reviewing just the data with the most suspect results, and then stopping when the remaining data and results improve.

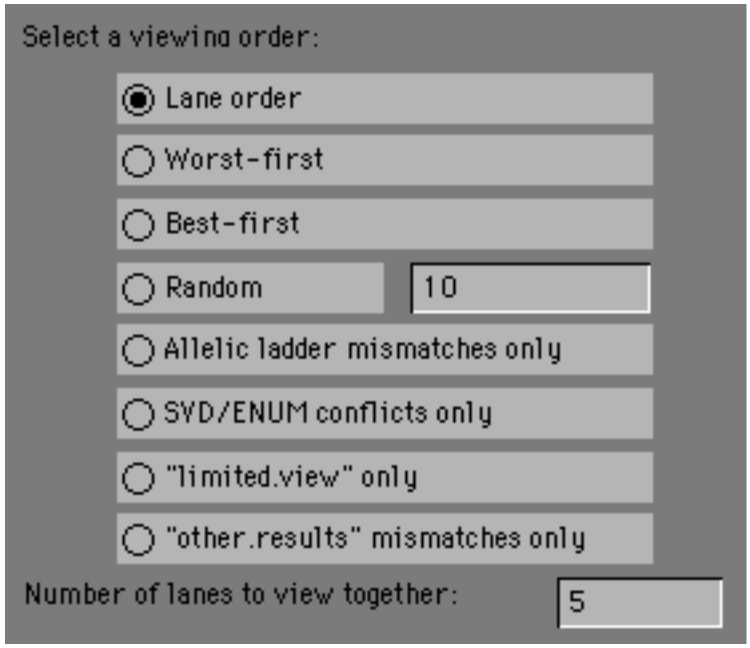


Figure 8.11. Various sorting orders available in *allele_view* for reviewing genotyping results. The user can view the results in the default lane order ("Lane order"), or in the worst-first order ("Worst-first") to correct suspect data. The user can also review the results in best-first order ("Best-first"), or review only a randomly selected subset (say 10) ("Random"). For checking, it can be useful to review just those genotypes not on the expected allelic ladder ("Allelic ladder mismatches"), or those for which the SVD and ENUM algorithms disagree ("SVD/ENUM conflicts"). Alternatively, the user can input a list of experiments to review ("limited.view"), or supply a file of independently called allele results to review the mismatches ("other.results mismatches").

The *allele_view* interface organizes and presents the results with the original data, allowing the user to verify the results at a single glance, or to correct the calls with simple mouse clicks. Figure 8.12 shows the main *allele_view* window with three display panes showing the (a) electropherogram, (b quantitation, and (c) genotype for one genotyping experiment. Each of the display panes are an active graphic object: if the user needs more detailed information, the display panes can be expanded into a more detailed view by a simple mouse click in the display pane, as shown in Figures 8.13 to 8.15.

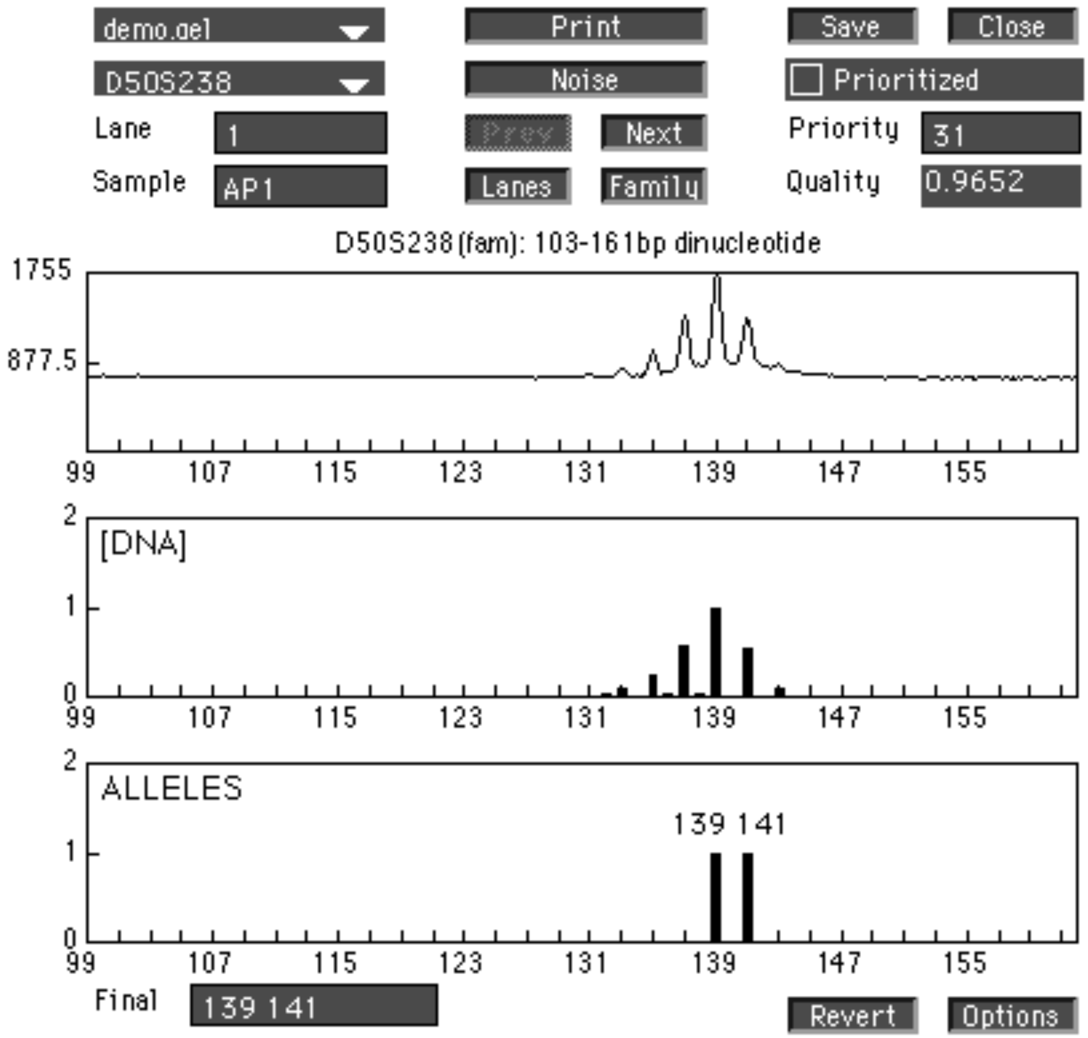


Figure 8.12. The main *allele_view* window. This window shows the electropherogram, the quantitation, and the genotype in three separate display panes. This compact presentation allows the user to verify the correctness of the calls at a single glance. If necessary, the user can expand the view in each of the three display panes by clicking inside the display pane. The user can also correct the genotype by typing in the box labeled "Final".

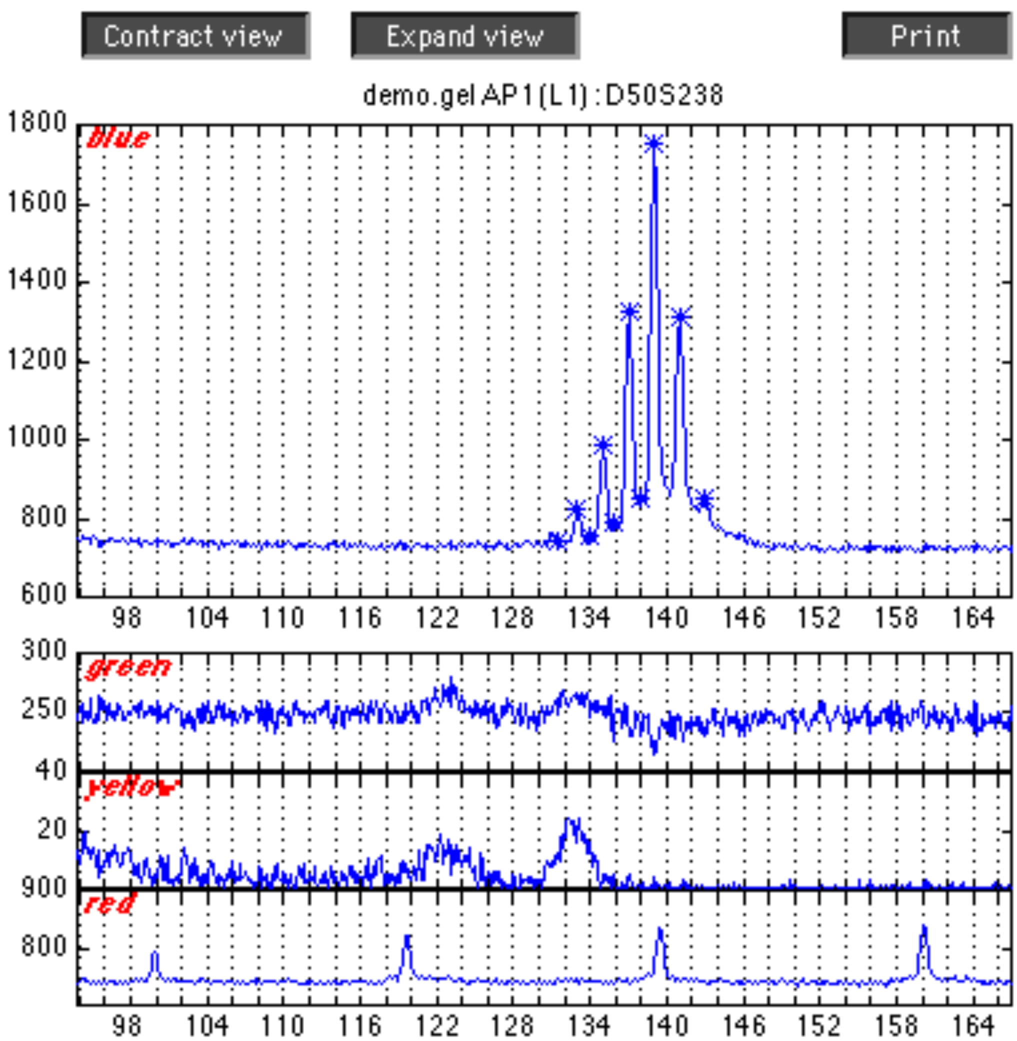


Figure 8.13. The zoomed-in view from the electropherogram display pane of the main *allele_view* window. This window shows the original electropherogram ("blue") for the experiment with the detected data bands (marked "*"). Electropherograms from the other dyes ("green", "yellow", and "red") are also shown. This allows the user to verify whether a spurious band was a dye bleedthrough artifact. To determine if there are data bands lying outside the window, the user can expand or contract the size window by clicking on the appropriate button ("Expand view" or "Contract view") in the window.

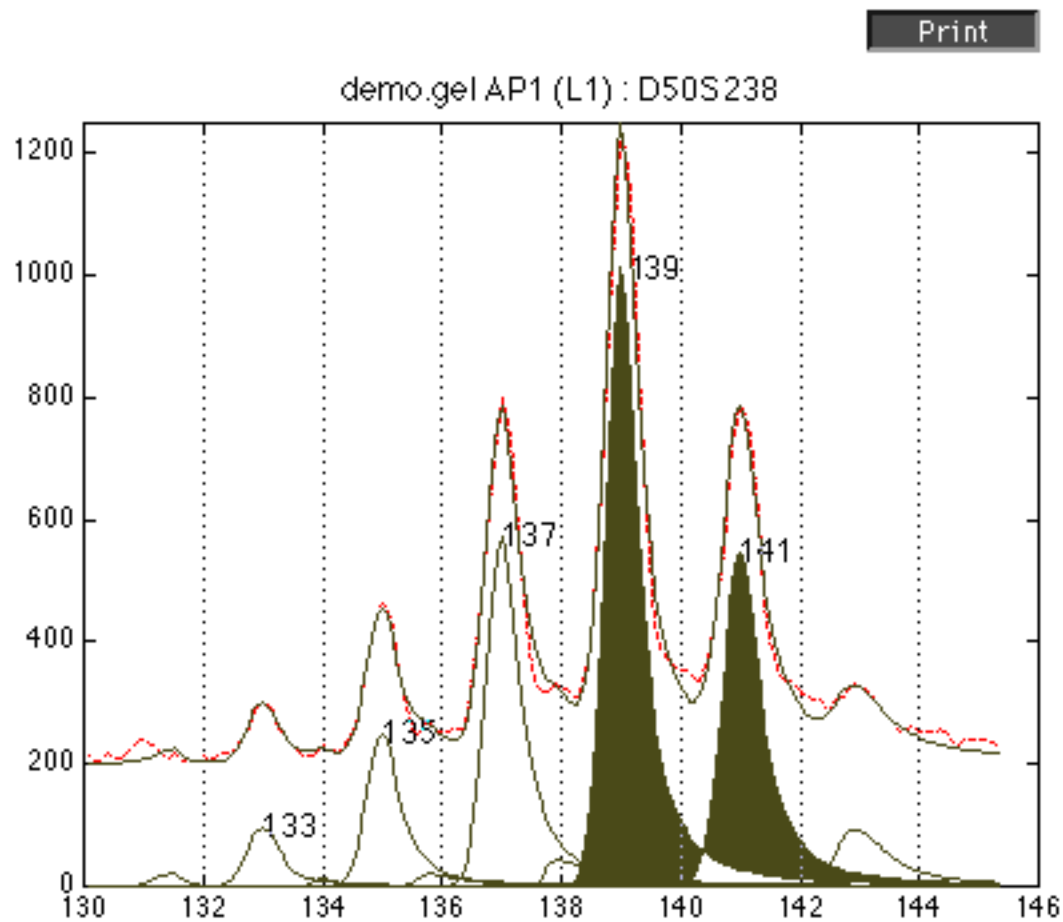


Figure 8.14. The expanded view from the quantitation display pane of the main *allele_view* window. This window shows original electropherogram (dashed line) overlaid with the computed fit (solid line) for comparison. The individual fitted peaks are shown below the electropherogram, and they are labeled with the respective binned alleles. The two bands that correspond to the called alleles are shaded.

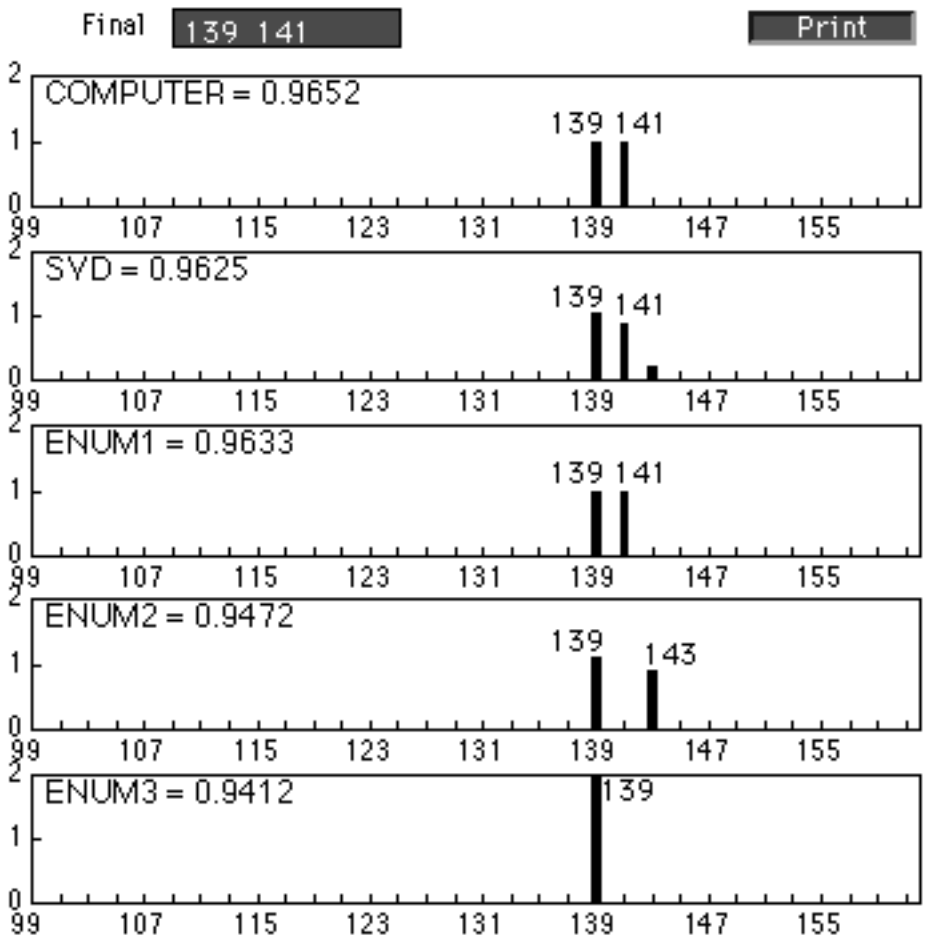


Figure 8.15. The expanded view from the genotype display pane of the main *allele_view* window. This window shows the computer's consensus call ("COMPUTER"), the SVD call ("SVD"), and the top three ENUM calls ("ENUM1", "ENUM2", and "ENUM3"). The computer's confidence measures for the computed genotypes are also displayed. The user can select any of the computed calls as the final genotype by clicking in the corresponding display pane, or edit the genotype in the box labeled "Final".

FAST-MAP does not use any pedigree information to call alleles. We leave family information as an independent source for the user to verify and correct genotypes, if necessary. However, if the user supplies appropriate pedigree information to FAST-MAP, the computer can organize and display the results by family. To bring up a family window showing the electropherograms, quantitations, or genotypes for all individuals in a family (see Figures 8.16 to 8.18), the user clicks on the "Family" button in the *allele_view* window. The family window shows the data and results of family members in a single window, allowing the user to visually detect any non-Mendelian inheritance in the genotypes, and correct them accordingly. The user can also click on the "Lanes" button (instead of the "Family" button) in the *allele_view* window to bring up a window showing the data and results from multiple adjacent lanes (that are not sorted by family), as shown in Figure 8.19.

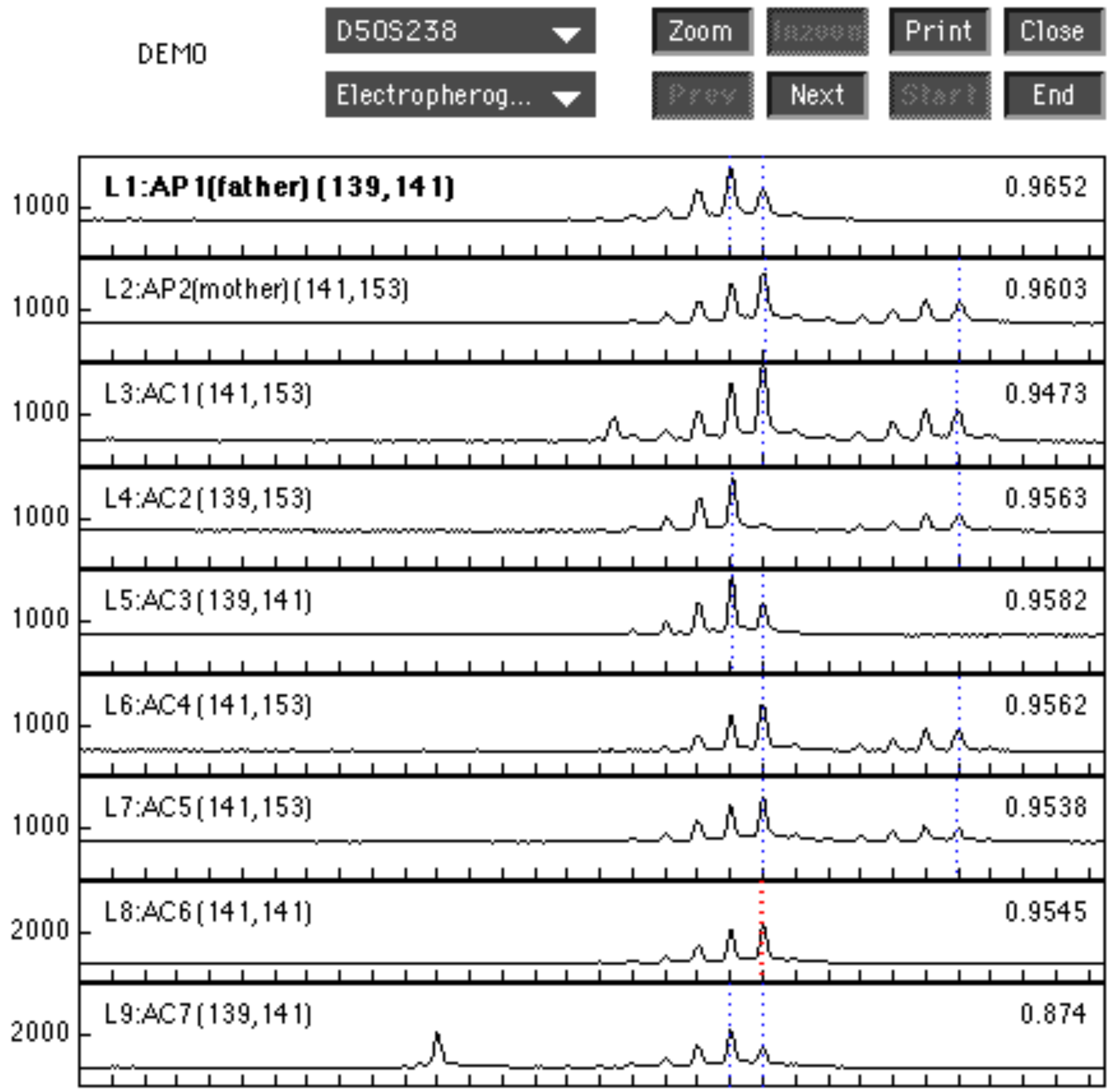


Figure 8.16. Viewing the electropherograms from a family in one window. Clicking on the "Family" button in the main *allele_view* window brings up the family window. The electropherograms of the individuals in the same family are shown in one window, together with their genotypes (indicated by the vertical dotted lines). The computer's confidence values on each of the genotypes are shown on the right.

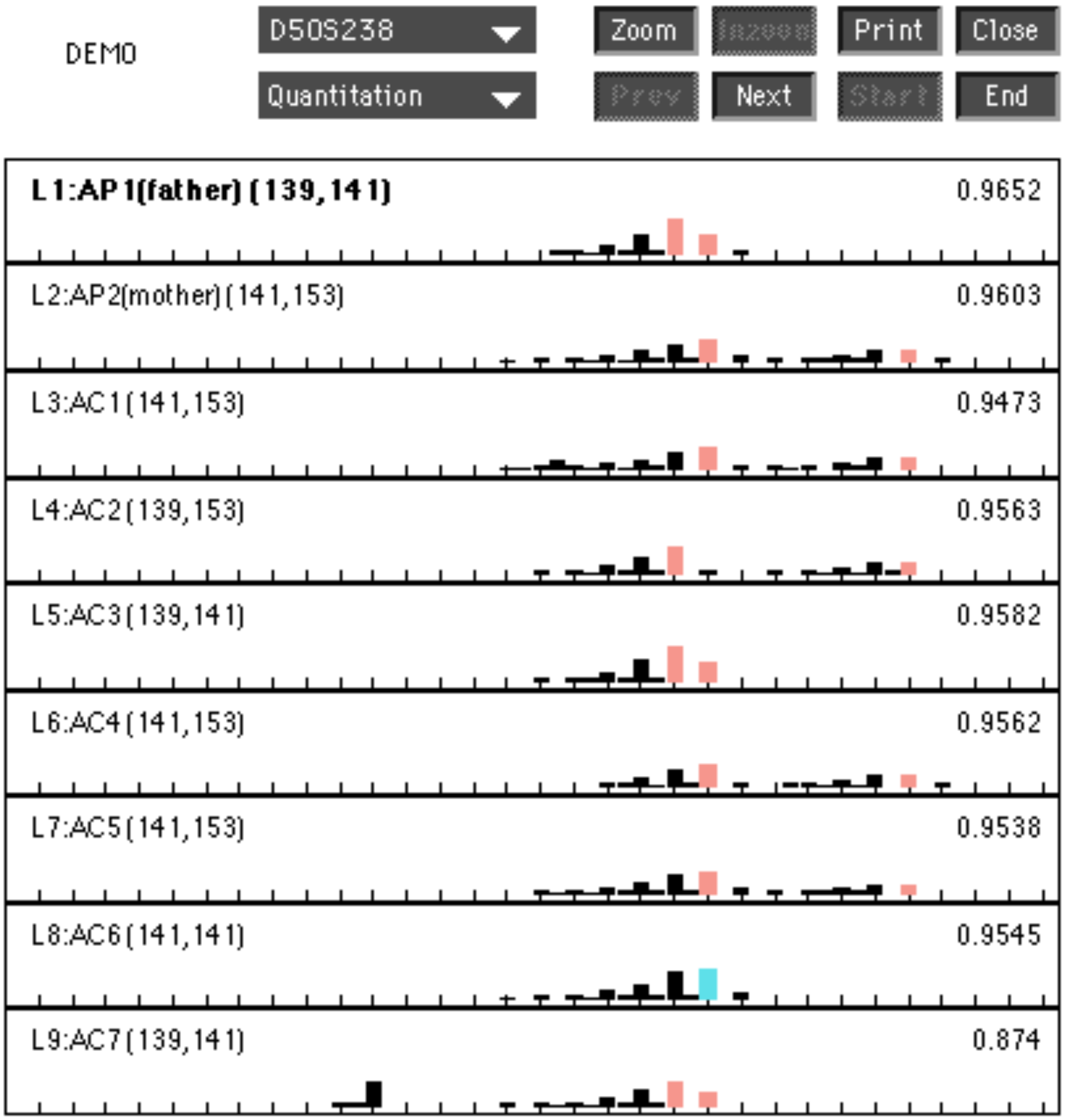


Figure 8.17. Viewing the quantitations from a family in one window. The user can view the genotype quantitations of the individuals in a family aligned in one window. The data bands corresponding to the allele calls are shaded.

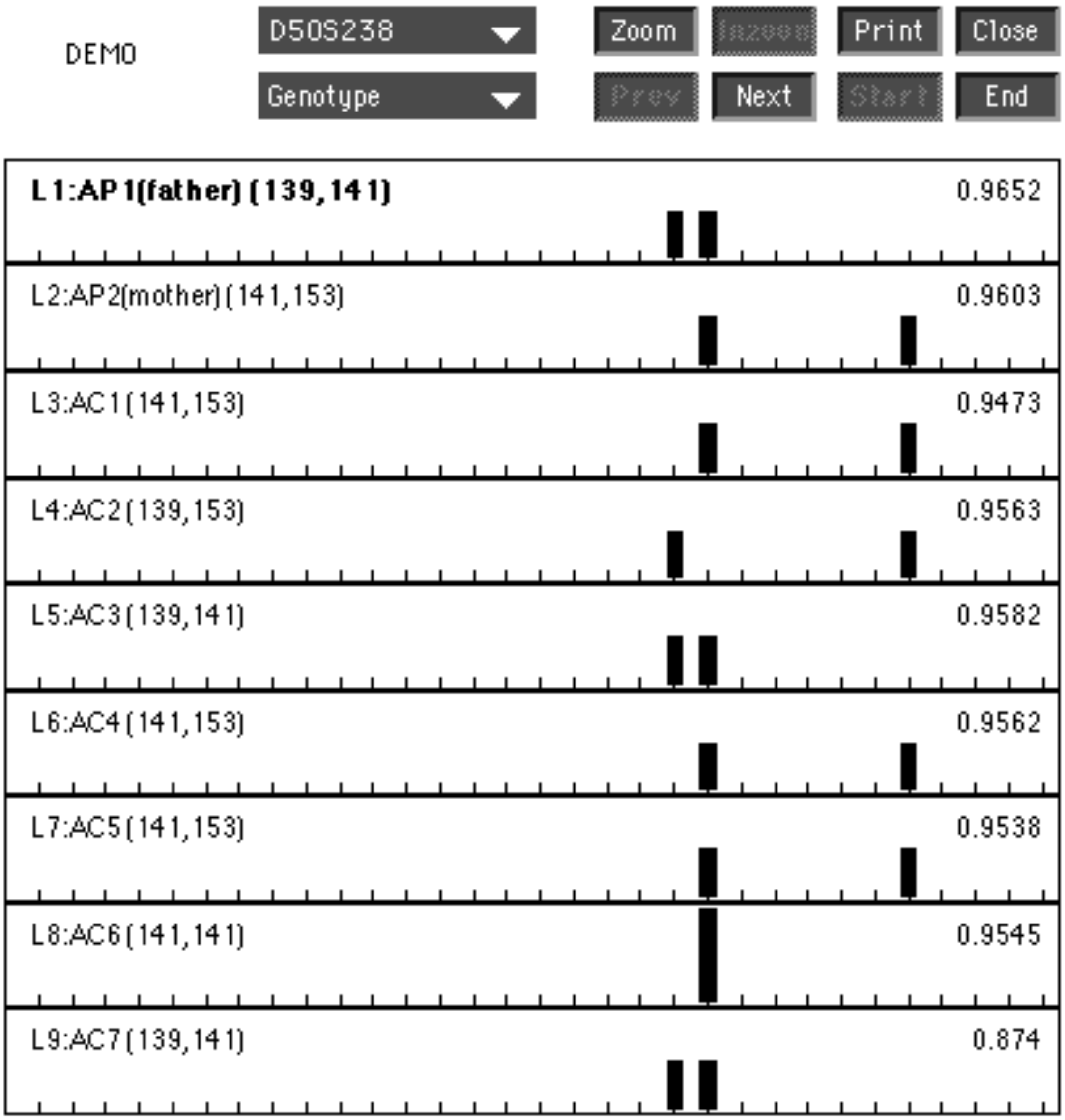


Figure 8.18. Viewing the genotypes from a family in one window. At a single glance, the user can detect non-Mendelian inheritance and correct it in the context of the other family members' genotypes, if necessary.

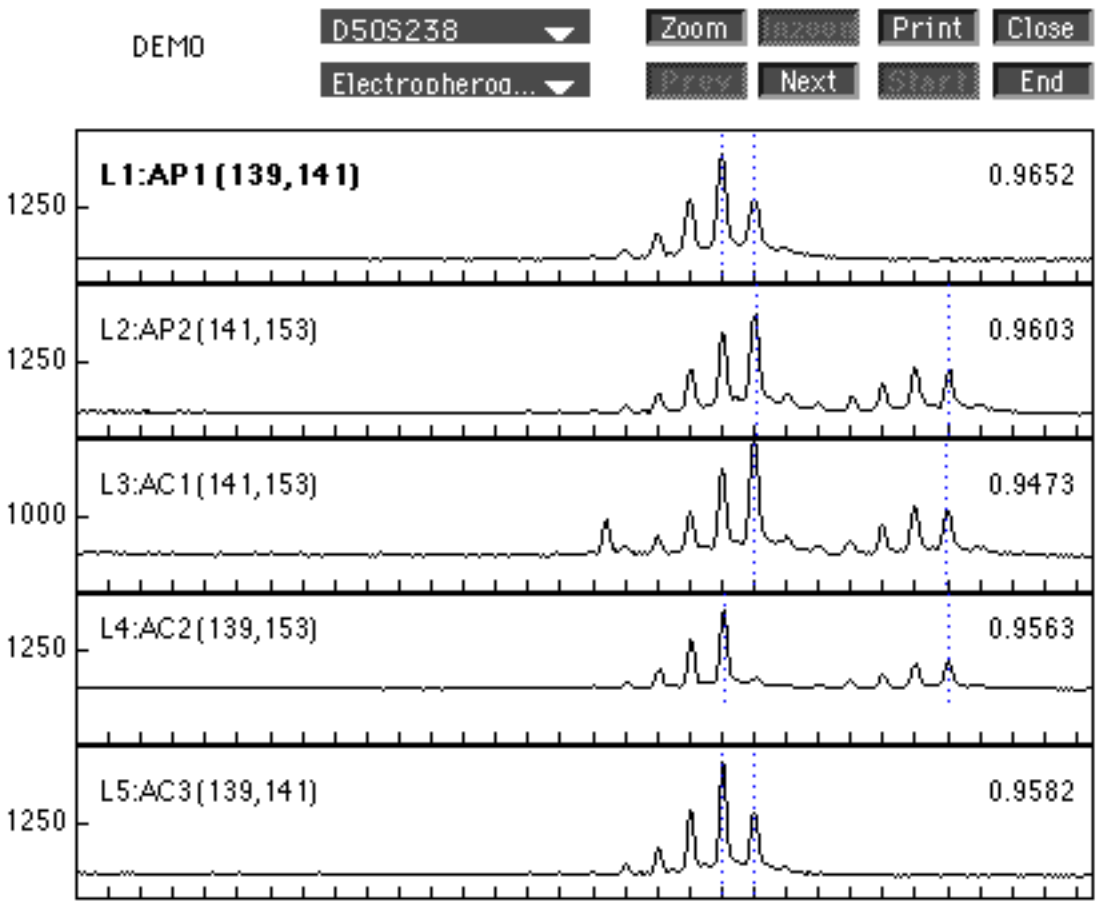


Figure 8.19. Viewing electropherograms from adjacent lanes together. In the absence of family information, the user can view multiple lanes together by clicking on the "Lanes" button in the main *allele_view* window. As in the family view window, the user can view the quantitations or genotypes of neighboring lanes by selecting the appropriate mode from the pop-up button on the top of the window (labeled here as "Electropherog...").

9. Results

Since its public release in July 1996, FAST-MAP has been used in academic research laboratories, national research institutions, and commercial genotyping centers worldwide. Many of our users downloaded the software from the FAST-MAP web site, and have been independently using FAST-MAP to score their data generated in their laboratories. Locally, our group alone has analyzed approximately 500 gels (contributed by our users) in the twelve-month period from July 1996 to July 1997. By working closely with our diverse users (including internal users within our group), and extensively testing the system using real data, we were able to continually improve on FAST-MAP's computational engines and user interface, making it a truly practical and robust system for microsatellite genotyping. We report here the results of FAST-MAP on representative sets of various types of data that we have analyzed with FAST-MAP.

9.1. ABI data

The ABI/373 and ABI/377 automated DNA sequencers manufactured by Perkin-Elmer generate the vast majority of fluorescent-based microsatellite genotyping data. For the purpose of system testing and developing, we have analyzed a selected set of about 50 ABI gels provided to us by our collaborators, predominantly by Dr. Soumitra Ghosh of the NIH FUSION[34] project, and Gordon Bentley of gene/Networks, Inc. In this section, we report on a comparison of our automated genotyping software results, versus our collaborators' labor-intensive use of ABI's Genotyper. Here, our comparisons were made on a FUSION data subset. We will report the results from a data set provided by Gordon Bentley at gene/Networks (a mouse phenomics company) in a later section on mouse data.

For assessment, the FUSION group selected six typical ABI gels, three generated on an ABI/373 and three from an ABI/377. Each gel had 50 lanes, and was run using the same panel, *panel 5,* which consists of 13 dinucleotide repeat markers (see Table 9.1), for a total of 3,900 experiments (102 uncallable alleles were not included in this analysis). The FUSION group manually scored their data using ABI's Genotyper as follows: first, two human experts *independently* reviewed and edited the alleles, using additional inheritance

[34]FUSION stands for Finland-U.S. Investigation of NIDDM Genetics.

processing to correct Mendelian inconsistencies when necessary; then, the experts met with each other to review their individual allele scores and to reach a consensus on the genotypes on which they disagreed.

marker	dye	allele window (bp)	repeat (bp)
D4S392	FAM	90 –123	2
D3S1311	FAM	129 – 158	2
D3S1565	FAM	168 – 197	2
D4S406	FAM	230 – 266	2
D3S1271	HEX	80 – 104	2
D4S423	HEX	120 – 152	2
D4S428	HEX	183 – 209	2
D4S405	HEX	274 – 304	2
D3S1285	TET	93 – 113	2
D4S1534	TET	128 – 164	2
D3S1263	TET	167 – 214	2
D3S1614	TET	218 – 258	2
D4S1597	TET	268 – 298	2

Table 9.1. The markers in *panel 5*. A total of 13 dinucleotide markers are designed for multiplexed run-out on a single gel: 4 of the markers in the panel are labeled with the FAM fluorescent dye, 4 with the HEX dye, and 5 with the TET dye.

The FUSION group also ran an old version of FAST-MAP to automatically compute an independent set of scores. They compared the old FAST-MAP scores against their own by comparing the allele size differences between the two methods of scoring. Their results showed the old FAST-MAP having a 4.39% discrepancy rate relative to the ABI/373 data, and a 1.06% discrepancy for the ABI/377.

Using the six FUSION assessment gels, we refined FAST-MAP further, and we greatly reduced its discrepancy rates to a 1.4% for the ABI/373 data, and a very low 0.3% rate for the cleaner ABI/377 data[35]. Table 9.2 shows the actual number of miscalls per marker made by FAST-MAP versus the number of post-Mendelian human edits. In this case, the exquisite quantitative and pattern-matching methods in FAST-MAP enabled the computer to

[35]These results were obtained using TrueAllele™ v1.02. TrueAllele™ is the commercial successor of FAST-MAP.

surpass manual scoring using only electrophoretic data *without* resorting to any pedigree information.

marker	ABI/373			ABI/377		
	no. of	miscalls		no. of	miscalls	
	genotypes	computer	human	genotypes	computer	human
D4S392	146	4	7	147	0	1
D3S1311	146	3	1	146	1	0
D3S1565	146	1	5	147	1	24
D4S406	146	3	9	147	1	4
D3S1271	143	0	0	146	0	1
D4S423	145	1	0	146	1	2
D4S428	145	0	0	146	0	0
D4S405	145	3	0	145	0	2
D3S1285	146	3	3	146	0	1
D4S1534	147	1	0	147	0	0
D3S1263	146	1	12	147	0	4
D3S1614	147	6	2	147	0	0
D4S1597	146	1	0	147	2	0
total	1894	27 (1.4%)	39 (2.1%)	1904	6 (0.3%)	39 (2.1%)

Table 9.2. The number of miscalls made by the computer versus the number of post-Mendelian edits made by the human experts on the six FUSION assessment gels.

In addition, FAST-MAP generates *binned* alleles in the genotypes which can be used *directly* as inputs to linkage analysis programs, whereas conventional genotyping software such as the ABI's Genotyper outputs sizes interpolated from the MW sizing calibration curve as the alleles. These *real-numbered* sizes have to post-processed by rounding into integral values for linkage analyses. We show in Table 9.3 the errors associated with direct rounding of the MW-interpolated sizes, compared with the errors resulted from rounding off our stutter-crawling adjusted sizes (which is how FAST-MAP generates the binned integral alleles in the genotypes). In the comparison, we assume that the alleles for each of the markers in *panel 5* are evenly-spaced, so any alleles that do not fall on the markers' allelic ladders (because of rounding errors) are considered as miscalls. Our results show that FAST-MAP's stutter-crawling method drastically reduces the associated rounding errors, generating superior genotyping results with unambiguous *integral* allele assignments.

marker	ABI/373			ABI/377		
	no. of genotypes compared	off allelic ladder		no. of genotypes compared	off allelic ladder	
		MW sizing	stutter crawling		MW sizing	stutter crawling
D4S392	146	78	3	147	6	0
D3S1311	146	61	0	146	83	0
D3S1565	146	1	0	147	1	0
D4S406	146	69	2	147	103	0
D3S1271	143	47	1	146	46	0
D4S423	145	0	0	146	6	0
D4S428	145	8	0	146	30	0
D4S405	145	92	0	145	42	0
D3S1285	146	85	4	146	0	0
D4S1534	147	1	0	147	6	0
D3S1263	146	84	0	147	1	0
D3S1614	147	23	1	147	81	0
D4S1597	146	60	0	147	36	0
total	1894	609 (32.2%)	11 (0.58%)	1904	441 (23.2%)	0 (0.00%)

Table 9.3. The number of genotypes containing alleles that do not fall on an allelic ladder (e.g. an odd-even allele pair). The miscalls associated with rounding directly from the MW-interpolated sizes are compared with the binning errors using FAST-MAP's stutter-crawling method.

Precise sizing of the allele bands is critical in size-polymorphic microsatellite markers — the subsequent linkage analyses depend directly on accurate and consistent allele assignments in the genotypes. As a way to evaluate the reliability of FAST-MAP in its allele assignments, we measure the "bin widths" of each allele category (or "bin"). The *width* of an allele bin is defined as the expected absolute difference between the bin label (an integer) and the unrounded sizes of the allele bands assigned to the allele bin (i.e. the expected rounding error). Since the final allele assignment depends on rounding the real-valued molecular sizes, a small bin width implies that the alleles can be assigned to the allele bin with little ambiguity. In Table 9.4, we compare the average bin widths of the markers in *panel 5* using the interpolated sizes from the MW calibration curves, versus the adjusted sizes using the stutter-crawling method. Our results show that for the FUSION group's *panel 5* data, stutter crawling reduces the average bin-widths from 0.25 bp to 0.19 bp for ABI/373 data and from 0.24 bp to 0.12 bp for ABI/377 data.

marker	ABI/373 bin width (bp)		ABI/377 bin width (bp)	
	MW sizing	stutter crawling	MW sizing	stutter crawling
D4S392	0.26	0.16	0.20	0.18
D3S1311	0.23	0.14	0.20	0.06
D3S1565	0.25	0.20	0.29	0.11
D4S406	0.26	0.25	0.30	0.15
D3S1271	0.28	0.22	0.26	0.16
D4S423	0.26	0.19	0.13	0.08
D4S428	0.27	0.19	0.27	0.10
D4S405	0.25	0.18	0.23	0.08
D3S1285	0.27	0.26	0.22	0.12
D4S1534	0.23	0.17	0.26	0.12
D3S1263	0.23	0.19	0.22	0.15
D3S1614	0.21	0.19	0.30	0.13
D4S1597	0.24	0.14	0.25	0.18
average	0.25	0.19	0.24	0.12

Table 9.4. Average allele bin widths using sizes interpolated from MW calibration curves versus using sizes that were adjusted with stutter-crawling.

As we have pointed out in Chapter 5, using unadjusted sizes interpolated directly from the MW calibration curve can cause errors because of the potential chemical differences between the marker DNA and the MW DNA used. We show in Table 9.5 the average size per repeat computed from the gradient of plotting the actual interpolated MW sizes against the assigned alleles (integers) for each marker in *panel5*. The results show that the MW sizes directly interpolated from the MW sizing grid range from 1.87 bp to 2.07 bp per repeat, instead of the expected 2 bp per repeat (the markers are all dinucleotides). This discrepancy can cause miscalls such as the off-ladder errors in Table 9.3, as well as miscalls that are on the allelic ladder, but are off by multiples of the marker's repeat size (this usually happens for genotypes with alleles that are far apart). Let us take the marker

D4S1597 as an example. According to Table 9.5, a genotype for D4S1597 having a pair of alleles that are 12 repeats apart (e.g., at 270 bp and 294 bp) will have a size difference of only 22.44 bp (12×1.87) between the alleles on an ABI/373 sequencer instead of the expected 24 bp (12×2), if we used sizes interpolated from the MW grid directly. This means that a genotype of (270 bp, 294 bp) for D4S1597 may have interpolated sizes of, say, (270 bp, 292.44 bp), which, when rounded, gives an erroneous (270 bp, 292 bp) as the genotype. In FAST-MAP, this problem is solved by locally adjusting the sizing grid using the actual marker data on the gels. The stutter crawling approach adapts the MW sizing grid to conform to the actual 2 bp-per-repeat ladder. Indeed, as shown in Table 9.5, the expected size per repeat after adjustment by stutter crawling is 2 bp per repeat for every marker in *panel 5*.

marker	ABI/373 size per repeat (bp)		ABI/377 size per repeat (bp)	
	MW sizing	stutter crawling	MW sizing	stutter crawling
D4S392	1.91	1.99	1.97	2.00
D3S1311	2.06	2.00	2.06	2.00
D3S1565	1.91	1.99	1.96	2.00
D4S406	1.93	1.99	1.95	1.99
D3S1271	2.07	2.00	1.94	2.00
D4S423	2.02	2.00	2.03	2.00
D4S428	1.95	2.00	1.94	1.99
D4S405	1.91	2.00	1.90	2.00
D3S1285	1.95	1.99	1.98	2.00
D4S1534	2.06	1.99	2.01	2.00
D3S1263	1.94	2.00	1.95	2.00
D3S1614	1.96	2.00	1.96	2.00
D4S1597	1.87	2.00	1.91	2.00

Table 9.5. Sizes per repeat using interpolated sizes from the original MW sizing grids versus using interpolated sizes from sizing grids adjusted by stutter crawling. As the markers in *panel 5* are all dinucleotides, the expected size per repeat for each marker is 2 bp.

9.2. Pharmacia data

The Pharmacia ALF system is another widely used automated DNA sequencer. Because it is a single dye system, high density internal size standards such as those in the ABI machines are not supported. The ALF also may have more limited base pair resolution than competing DNA sequencers. However, the gels do run very straight, which permits automatic lane detection into 1D electropherogram signals during signal acquisition, without any intermediate 2D imaging step. In collaboration with Drs. Michael Gorin, Robert Ferrell, and Daniel Weeks at the University of Pittsburgh, we reviewed over 400 Pharmacia genotyping gels for their retinal genetics project. We report here a comparison of our automated genotyping software results, versus our collaborator's labor-intensive and error-prone results using the ALF software.

Over 6-9 months, our collaborators generated Pharmacia ALF gels containing 40 lanes per gel using panels containing 2 to 5 tetra- (and other) nucleotide repeat markers. They scored these data, and performed initial statistical analyses on their allele calls, including:

- a comparison of allele frequencies for each marker in their ALF data, versus the reported Cooperative Human Linkage Center (CHLC) frequencies; and
- a chi-square analysis for expected degree of allele homozygosity.

These initial analyses revealed a very large allele scoring error (for some markers, in excess of 90%), indicating that the gels had to be rescored. As a three-week project, we rapidly adapted FAST-MAP to handle ALF data, and helped to rescore over 150 key ALF gels within the time allotted.

In the first week, we developed custom user interfaces for sizing in ALF data. Most of our collaborators' data differed from the ABI data (that FAST-MAP was designed for) in two ways:

(a) only two widely separated size standards (e.g. at 100 bp and 500 bp) were used, giving a low local sizing resolution that was inadequate for the 4 bp separation for their tetranucleotide markers; and

(b) the size standards were size separated together with the marker DNA in every lane (in a *single* dye), with the result that one size standard was typically buried in the genetic marker data.

To solve the problem of low sizing resolution, we extended our expectation-based approach by pre-constructing a refined *MW binning library* using gels loaded with a 50-bp sizing ladder[36]. This pre-calibrated MW binning library was then used to *predict* the *non-loaded* sizing bands on the gels that only ran two size standards. With these pre-calibrated expectations stored in the MW binning library, we were able to achieve an effective 50-bp sizing resolution, even on gels with only two size standards that were 500 bp apart.

The second problem of distinguishing the size standard data that were buried in the genetic marker data was much more difficult to solve computationally without user assistance. To achieve our primary goal of attaining the best results possible in a short period of time, we decided to incorporate user assistance in helping the computer decipher the data. We added a powerful "snap-to" user interface to FAST-MAP that allowed the operator to click on only a few of the (visible) size standards to give the computer some information about where the size standards are. By exploiting the horizontal continuity of size standards between lanes, the computer *horizontally* interpolated <u>between</u> the lanes (by intelligent peak finding), and automatically filled in the size standard bands for all the lanes, as shown in Figure 9.1. The computer then used the pre-calibrated MW binning libraries to *vertically* interpolate <u>within</u> the lanes, rapidly establishing highly accurate internal lane sizing from the available data.

---

[36]This was the highest sizing resolution available in their data.

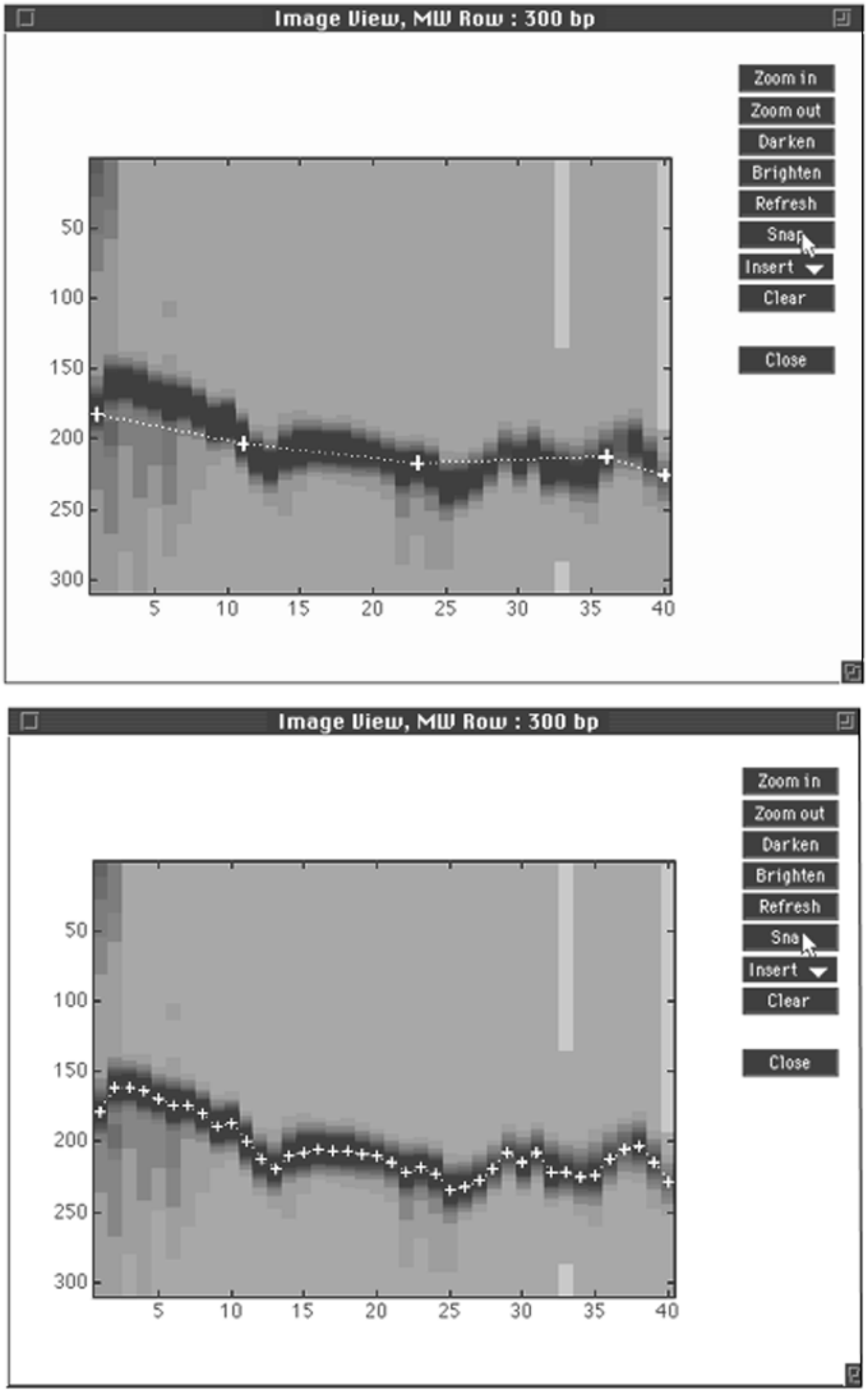


Figure 9.1. Custom "snap-to" ImageView ALF interface. Top: The user clicks on several size standards to suggest where the size standards are. Bottom: The computer automatically fills in the rest of the size row, and then calibrates the DNA sizes in every lane.

We were able to rapidly adapt FAST-MAP to process qualitatively different genotyping gel data in less than a week. We attribute this success to FAST-MAP's highly modular user interface and computational engine. In the second and third weeks of our project, we worked closely with our collaborators and succeeded in scoring the 170 ALF gels that they needed for their deadline. We continued working with them to reach a total of over 400 scored ALF gels over the following month, after which our collaborators continued to use FAST-MAP to score their gels on their own.

To evaluate the quality of FAST-MAP's allele scoring on the ALF data, we compared the allele frequencies for the expected CHLC versus the observed ALF and FAST-MAP calls for many of the markers analyzed. In Figure 9.2, we plot the comparison for one of the markers, showing that the ALF calls did not fit the expected allele distribution, whereas the FAST-MAP scorings did. In fact, for the marker shown, the homozygosity chi-square values for the ALF and FAST-MAP scorings were 186 and 3, respectively. Overall, the homozygosity chi-square values for the FAST-MAP scorings were consistent with the expected CHLC allele distributions (Figure 9.3). The ALF vs. CHLC allele frequency comparison data suggested to our collaborators that they had a scoring problem. The FAST-MAP vs. CHLC frequency comparison data reassured them that FAST-MAP's automated scoring was reasonably accurate.

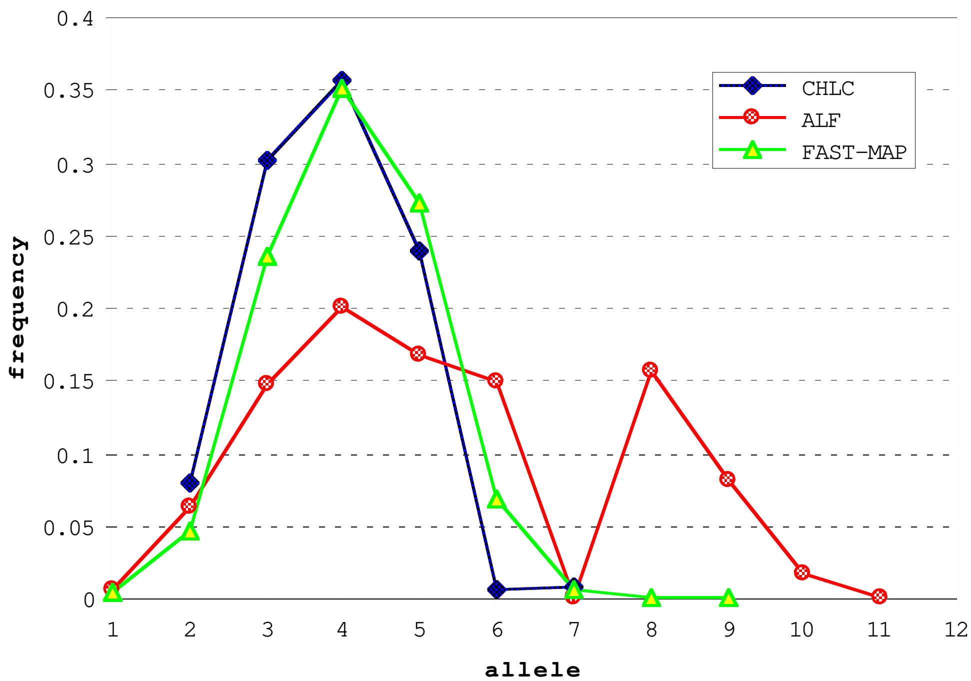


Figure 9.2 A comparison of CHLC, ALF, and FAST-MAP allele distributions for a tetranucleotide repeat marker.

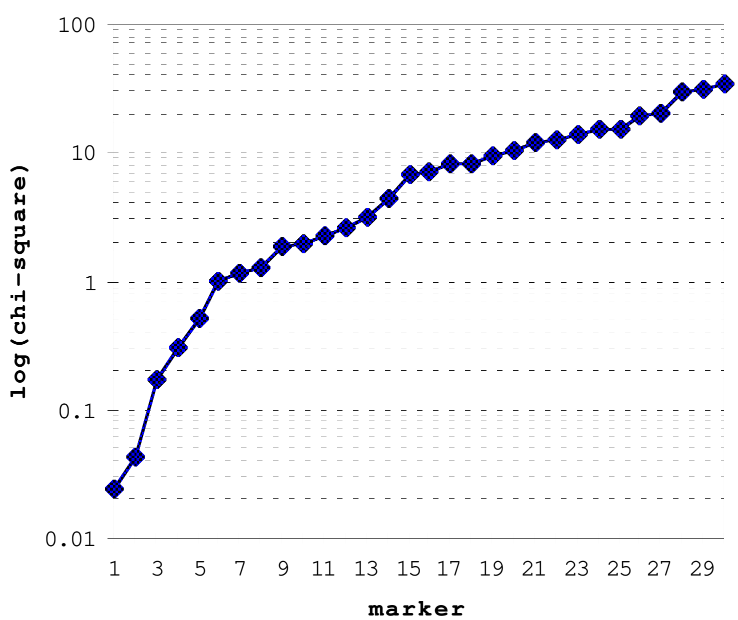


Figure 9.3. The logarithm of the homozygosity chi-square value (FAST-MAP scoring vs. CHLC allele distribution), plotted in rank order for FAST-MAP scoring of 30 CHLC markers.

9.3. Mouse data

The mouse, a genetic relative of humans, is an indispensable model organism for the investigation of the genetic mechanisms in human diseases. The power of mouse genetics is mostly derived from using *inbred* (genetically "pure") strains of mice for cross-breeding[37]. An inbred mouse strain produces a monoallelic system – each marker is homozygotic for precisely one allele. Crosses between inbred strains result in an elegant *biallelic* system which is very useful for the study of inherited complex diseases.

Our collaborator Gordon Bentley at gene/Networks (a mouse phenomics company) generated such backcross genotyping data using microsatellite markers on ABI/377 sequencers. In the study described here, two sets of mouse samples were used, named MOUSE1 and MOUSE2. MOUSE1 and MOUSE2 each have 96 samples. A total of 12 mouse microsatellite marker panels were studied with each sample set. For each marker panel, PCR products of all 96 samples (from either MOUSE1 or MOUSE2) were loaded onto two ABI/XL gels, with 48 samples on each gel. This compact experimental setup produced a total of 48 XL gels (16,128 genotypes) for the two sets of mouse samples and all the twelve marker panels (84 distinct dinucleotide markers[38]).

We show some of the mouse electropherograms in Figure 9.4 to illustrate the wide range of stutter patterns in the data. Although there was great diversity in the stutter patterns observed in the 84 mouse markers, these stutter patterns were not dissimilar from those encountered in human data. As such, we expected FAST-MAP to analyze the mouse data without any system modifications.

---

[37]The short breeding time (21-day gestation) of mice allows geneticists to rapidly generate unlimited numbers of offspring in the laboratories.

[38]There were actually 91 markers, but 7 of them did not amplify and were discarded from the analysis.

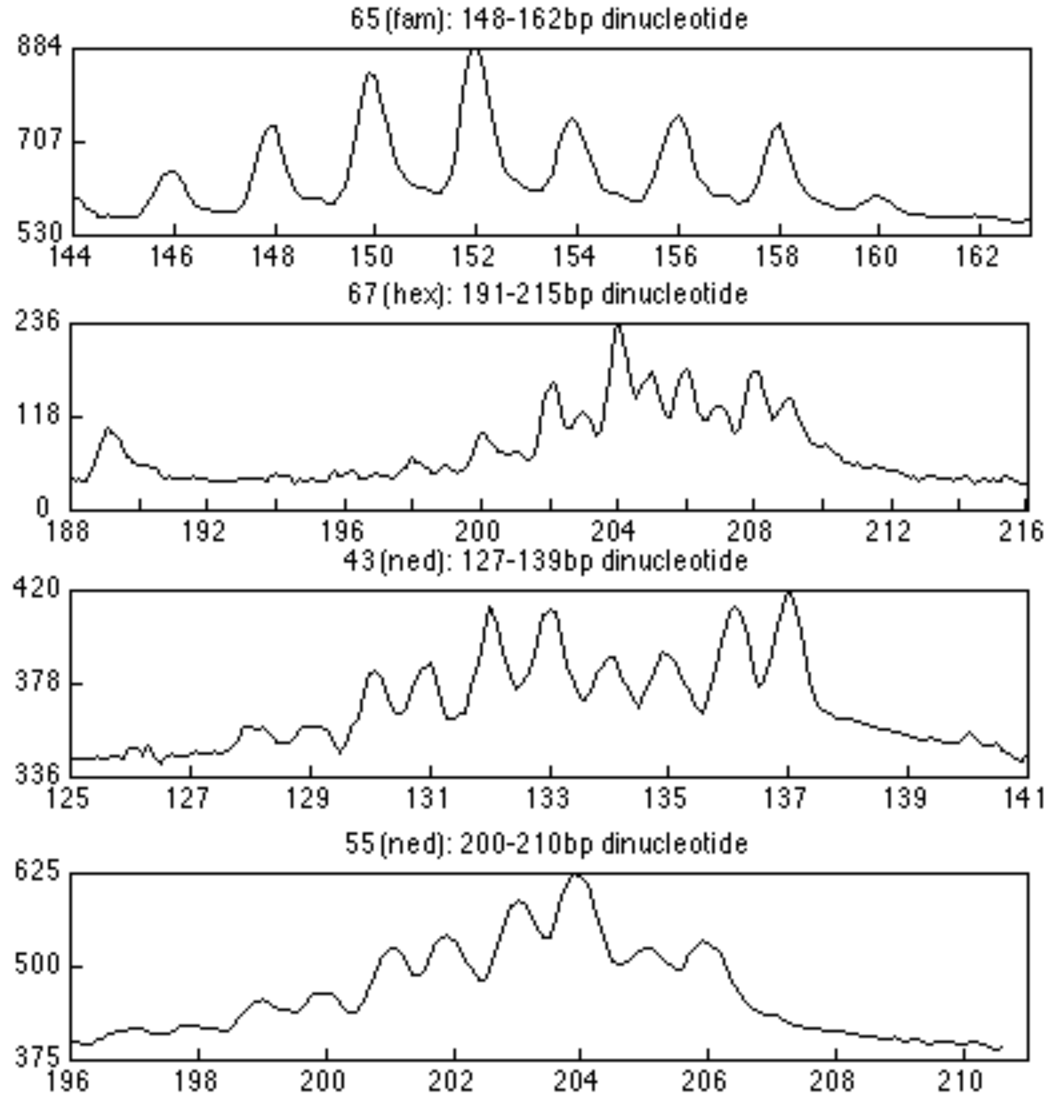


Figure 9.4. Electropherograms from mouse dinucleotide markers. Shown here are examples from four mouse markers to illustrate the diverse stutter patterns in the markers. In particular, the top marker (marker "65"), showed almost no "plus-A" artifacts. The second marker (marker "67"), showed significant "plus-A" artifacts, but the "plus-A" bands are less dominant than the actual allele bands. In the third marker (marker "43"), the "plus-A" bands and the data bands have comparable intensities, whereas in the last marker (marker "55"), the "plus-A" bands actually dominate the actual bands. The fluorescent dyes used were FAM, HEX, and NED from ABI's filter set D.

We ran FAST-MAP unmodified on the 48 gels of mouse data provided to us by our collaborator. For evaluation purposes, we did not allow FAST-MAP to exploit the biallelic characteristic of the mouse data to help in calling the alleles. This artificial limitation allowed us to assess two FAST-MAP features:

(a) its precise sizing to consistently *align* the alleles from lane to lane, as well as gel to gel (since the samples were loaded across two separate gels), and
(b) its stutter removal to accurately call *only* two distinct alleles for each marker in the biallelic mouse system.

Figures 9.5(a,b,c) show how FAST-MAP accurately called the alleles in the mouse data without imposing any biallelic constraints.

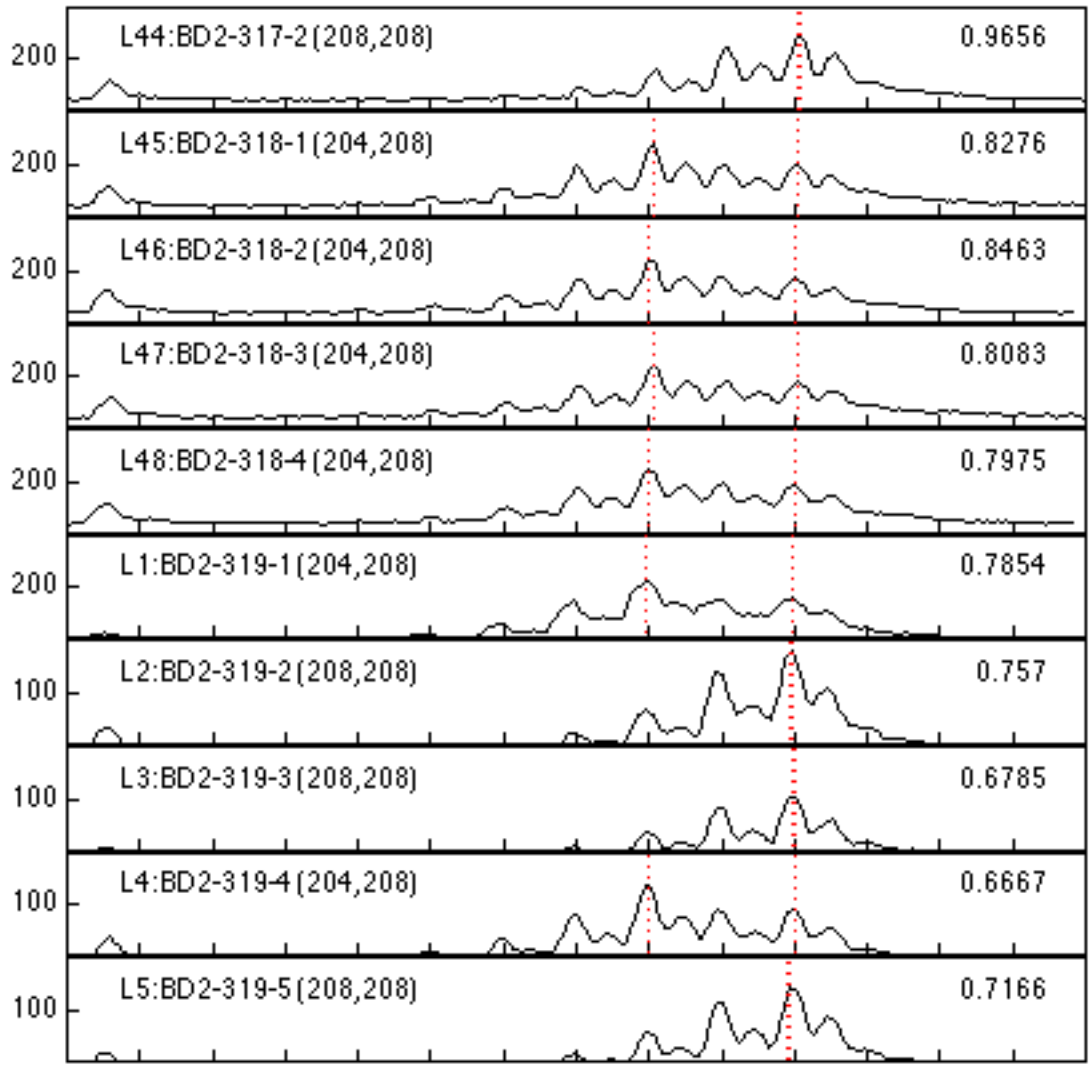


Figure 9.5a. Electropherograms from a mouse marker (marker "67" from Figure 9.4). Shown here are 10 lanes of electropherograms, 5 from each of the two gels that was loaded with the mouse DNA from the corresponding marker panel (we show the last 5 lanes from the first gel, and the first 5 lanes for the second gel here). The vertical dotted lines show the alleles called by FAST-MAP.

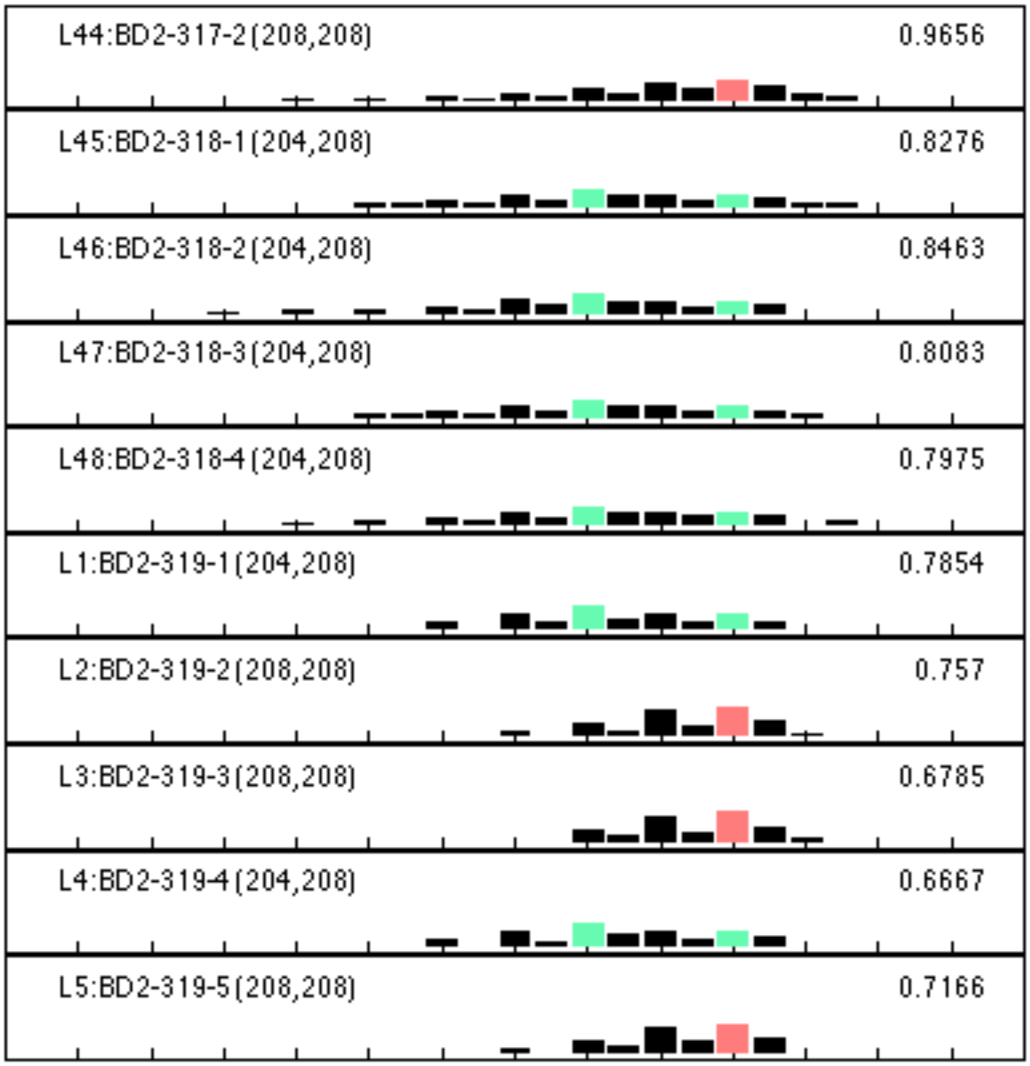


Figure 9.5b. The quantitated data from the electropherograms shown in Figure 9.5a. The darkened bars depict the stutter bands, while the shaded bars show the data bands corresponding to the called alleles. FAST-MAP was able to systematically remove the stuttering and accurately call the two alleles in the biallelic system, despite the high degree of PCR stuttering in this marker. This stutter is shown by the presence of *many* darkened bars in each of the experiments.

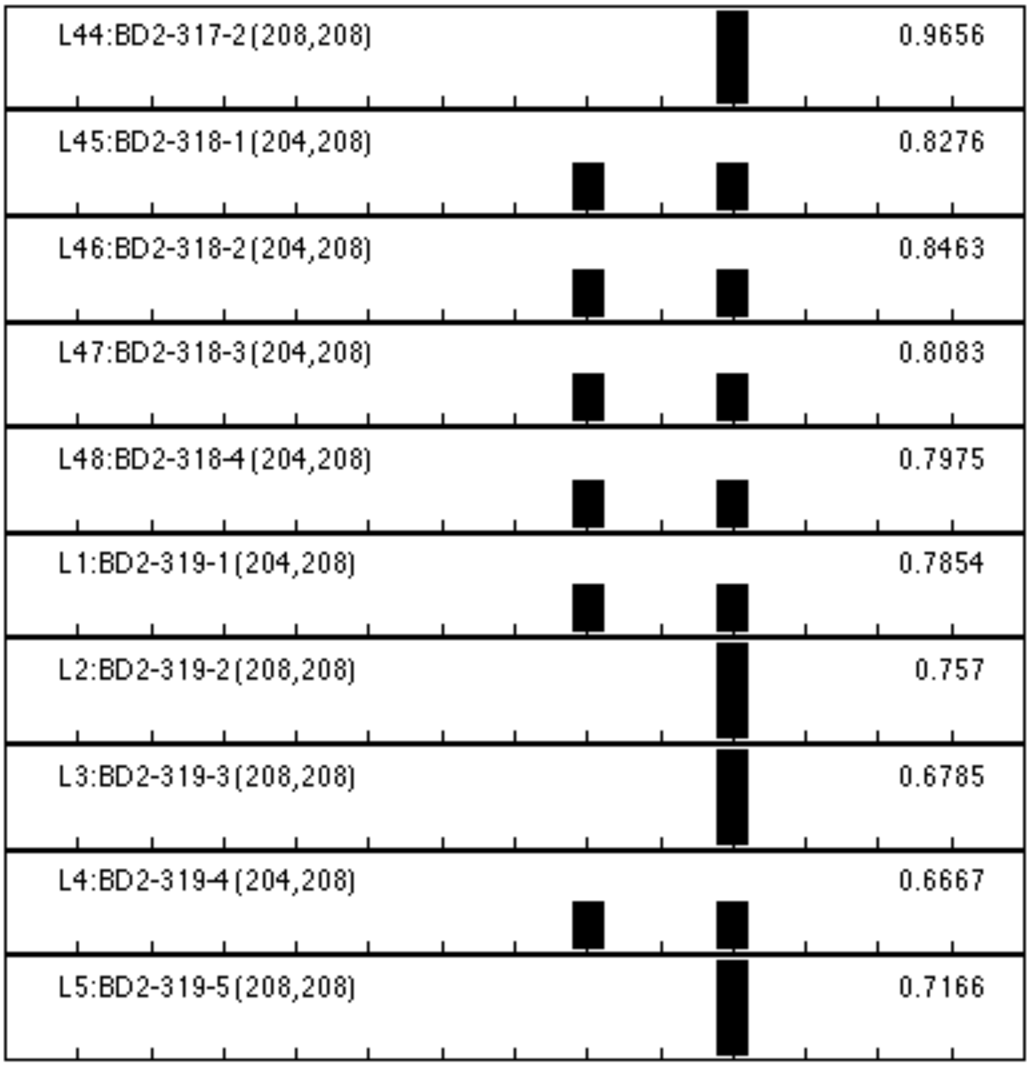


Figure 9.5c. The alleles called by FAST-MAP for the mouse data shown in Figure 9.5a. The results show that FAST-MAP was able to *consistently* call the two alleles across gels and amidst high PCR stuttering, producing genotypes that conformed to the underlying biallelism.

We evaluated FAST-MAP's performance in the mouse data by measuring the number of genotypes called by FAST-MAP that violated the biallelic constraint. A marker genotype called by FAST-MAP that includes a third allele is considered a miscall, since there would only be two possible alleles if all the data (96 genotypes for each sample set) for the marker had been called correctly[39].

Number of genotypes with biallelic conflicts	Number of markers	
	MOUSE1	MOUSE2
0	22	28
1	18	9
2	7	7
3	5	7
4	3	1
5	3	1
6 to 10	8	8
11 to 20	4	9
21 to 30	2	3
31 to 40	2	5
41 to 50	6	3
51 to 60	2	0
61 to 70	1	2
71 to 80	1	0
81 to 90	0	1
91 to 96	0	0
total no. of markers	84	84

Table 9.6. Number of markers in each mouse sample set classified by the number of genotypes called by FAST-MAP with biallelic conflicts.

Table 9.6 shows the number of markers with biallelic conflicts in the genotypes called by FAST-MAP. The results show that, without any modifications to handle biallelic mouse data, FAST-MAP was able to type 22 out of the 84 markers in MOUSE1, and 28 out of

---

[39]Note that the converse is not true: perfect biallelism in the called alleles does not necessarily imply accurate allele calls. Errors such as homozygotes called as heterozygotes (and vice versa) are not accounted for in our biallelism comparisons. Thus, the biallelism results presented here should be considered as an upper bound on FAST-MAP's performance on this set of mouse data.

the 84 markers in MOUSE2, with *perfect* biallelic allele calls. As summarized in the charts in Figure 9.5, at least two-thirds of the 84 markers had close to perfect (0 to 5 errors) biallelic allele calls.

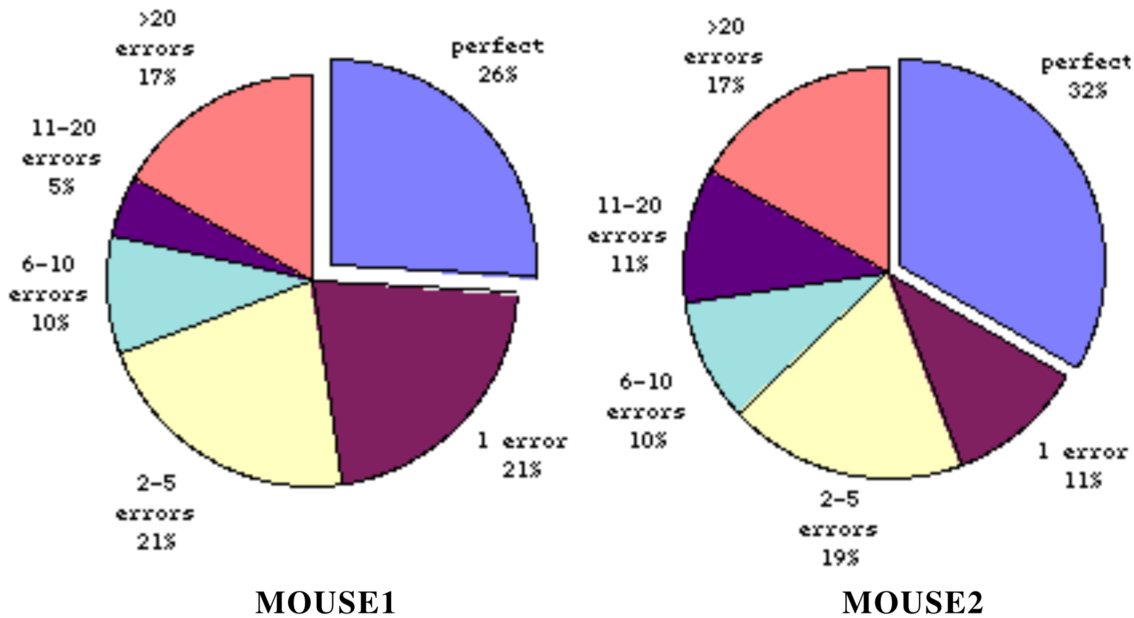


Figure 9.6. Pie-charts showing the percentages of markers in the mouse data with varying degrees of biallelic conflicts in the genotypes called by FAST-MAP.

The results showed that FAST-MAP's automated genotyping technology was not restricted to the human species, and that FAST-MAP could genotype biallelic data with reasonable accuracy without any extensions. Of course, one should take advantage of the biallelic constraint in the data to improve the accuracy of the allele calls. FAST-MAP could use the additional biallelic constraint to focus its pattern matcher on recognizing at most the two possible alleles for each marker. We show in Figure 9.7 an example of how this biallelic *expectation* can considerably improve the accuracy and robustness of FAST-MAP's expectation-based allele calling by avoiding any spurious bands that occur at the non-biallelic locations.

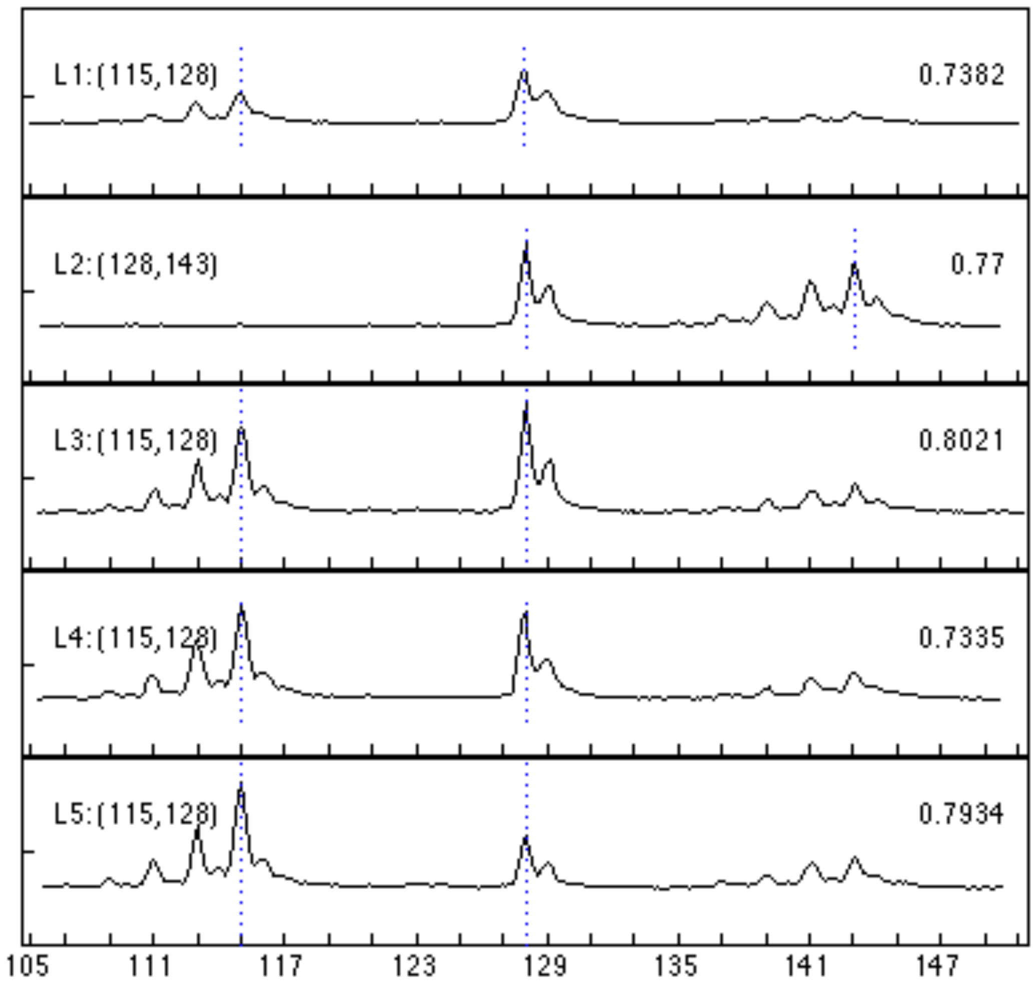


Figure 9.7. Potential for automating error detection using the biallelic nature of the data. We show here an example from marker "3" of MOUSE2 in which there were bleedthrough (or contamination) bands in every lane at 128 bp (and 129 bp) that caused the miscalls. If FAST-MAP could exploit the expected biallelism in the data, it would have ignored the spurious bands at 128 bp using the *expectation* of only two possible alleles at 115 bp and 143 bp to call the alleles for the marker.

9.4. Pooled DNA data

In Chapter 7, we described how our deconvolution methods enabled a useful new functionality called *pooled microsatellite genotyping*.

- By pooling together the DNA from many (e.g., 10-100) individuals in equimolar ratios, and PCR amplifying with a single genetic marker, allele frequency data can be generated;
- Estimates of the allele frequency in the pooled population can then be determined using our deconvolution methods to effectively eliminate underlying PCR stutter in the pooled data.

The pooled DNA approach is attractive because it can potentially reduce the number of required experiments 10-100 fold. For example, a single PCR amplification and sizing of a pool of 100 individuals reduces 100-fold the work to amplify and size each individual separately. Chapter 7 described a *computational* solution for genotyping pooled microsatellite DNA as an extension of our GMBD algorithms for individual DNA genotyping. The success of the pooled DNA approach also relies on *experimentally* generating high quality pooled DNA data that can be automatically analyzed by the computer.

9.4.1. Post-PCR pooling

We conducted laboratory experiments to assess how well our automated genotyping software determined allele frequencies from microsatellite data on pooled DNAs. Lillian Bloch, from Cybergenetics, Inc., generated the post-PCR pooling data described in this section, as well as the pre-PCR pooling data to be described in the next section.

Our initial explorations were done using pools of post-PCR products. Individual DNAs were PCR amplified with microsatellite marker primers, and size separated on an ABI sequencer. FAST-MAP determined the alleles in each DNA sample, and in the process constructed the binning, stutter, and relative amplification calibrations for each marker. Pools of 2, 4, and 8 individual PCR products were then constructed, and size separated on an ABI/377 sequencer. FAST-MAP then performed pooled GMBD to compute the allele distributions for the pooled samples of different sizes.

For comparison, we compute the L1-norm between a computed allele distribution vector and the actual allele distribution vector, divided by two to give the number of alleles "misplaced" by the computer. Each misplaced allele causes a difference of one allele at the source allele bin, and a second allele at the target allele bin. We divide the L1-norm by two so that each misplaced allele is not counted twice.

Let us examine the results of D16S403, one of the dinucleotide repeat markers that we investigated in more details[40]. D16S403 is a FAM-labeled dinucleotide repeat marker with an allele range of 125-155 bp. In our experiment, eight individual DNA samples for D16S403 were PCR amplified, and then pools of 2, 4, and 8 were constructed from the individual PCR products. All the samples (individual and pooled) were then size separated on an ABI/377 sequencer, and the resulting gel data automatically analyzed by FAST-MAP.

FAST-MAP used the individual sample experiments to construct the binning, stutter, and relative amplification calibrations for D16S403. It also determined the alleles in each DNA sample, as shown in Table 9.7. From these genotypes, we compute the expected (actual) allele distributions for the different pooled samples.

sample	allele1	allele2
s1	135	137
s2	137	141
s3	143	151
s4	139	149
s5	139	141
s6	139	141
s7	137	143
s8	139	139

Table 9.7. Genotypes of the eight individual DNA samples for D16S403. The alleles are in bp units.

With the D16S403 marker library that it had constructed from analyzing the individual sample experiments, FAST-MAP then computed the allele distributions for the pooled experiments. We show the results for the eight pooled samples in Table 9.8.

[40]We have shown example pooled data and results for D16S403 in Section 7.4 previously.

samples		allele distribution									total	misplaced
		135	137	139	141	143	145	149	151	153		
s1-2	actual	1	2	0	1	0	0	0	0	0	4	
	computed	1	2	0	1	0	0	0	0	0	4	
	L1-norm	0	0	0	0	0	0	0	0	0	0	0
s3-4	actual	0	0	1	0	1	0	1	1	0	4	
	computed	0	0	1	1	1	0	1	0	0	4	
	L1-norm	0	0	0	1	0	0	0	1	0	2	1
s5-6	actual	0	0	2	2	0	0	0	0	0	4	
	computed	0	0	2	2	0	0	0	0	0	4	
	L1-norm	0	0	0	0	0	0	0	0	0	0	0
s7-8	actual	0	1	2	0	1	0	0	0	0	4	
	computed	0	1	1	0	1	0	0	0	1	4	
	L1-norm	0	0	1	0	0	0	0	0	1	2	1
s1-4	actual	1	2	1	1	1	0	1	1	0	8	
	computed	1	2	2	1	1	1	0	0	0	8	
	L1-norm	0	0	1	0	0	1	1	1	0	4	2
s5-8	actual	0	1	4	2	1	0	0	0	0	8	
	computed	0	1	3	2	1	0	1	0	0	8	
	L1-norm	0	0	1	0	0	0	1	0	0	2	1
s1-8	actual	1	3	5	3	2	0	1	1	0	16	
	computed	1	3	4	4	3	0	0	0	1	16	
	L1-norm	0	0	1	1	1	0	1	1	1	6	3

Table 9.8. Actual versus computed allele distributions for D16S403. Each of the pooled samples were labeled with an *si-j,* indicating that the pooled sample contained individual samples *si* through *sj*. The number of "misplaced" alleles (the rightmost column) in each computed allele distribution was computed by dividing the L1-norm by 2.

Table 9.9 shows the number of misplaced alleles in the allele distributions computed by FAST-MAP for the different dinucleotide repeat markers in our post-PCR pooling experiments. The number of misplaced alleles increases as the size of the pools become larger, since there are more alleles to estimate with larger pools.

microsatellite marker	repeat (bp)	number of misplaced alleles						
		2-sample pool				4-sample pool		8-sample pool
D16S403	2	0	1	0	1	2	1	3
D1S468	2	0	0	1	0	2	1	2
D3S1304	2	1	0	1	0	2	1	2
D5S211	2	1	1	1	0	2	1	4
D5S408	2	0	0	1	0	2	1	3

Table 9.9. Number of misplaced alleles in the computed allele distributions for different sizes of pooled samples. A total of 8 DNA samples were used in generating post-PCR pools of size 2, 4, and 8, giving four 2-sample pools, two 4-sample pools, and one 8-sample pools for each of the markers studied.

To determine whether our pooled GMBD methods scale up with increasing pool sizes, we compare the *total* number of misplaced alleles (out of the 16 alleles from the 8 DNA samples) for different pool sizes. That is, we sum the misplaced alleles from Table 9.9 for samples of the same pool sizes. The results in Table 9.10 show that the performance of pooled GMBD does not noticeably diminish as the pool size increases. For example, the computed allele distributions (for 16 alleles) from a single 8-sample pool were as accurate as those computed from two batches of 4-sample pools.

microsatellite markers	repeat (bp)	total no. of misplaced alleles (out of 16 alleles)		
		2-sample pool	4-sample pool	8-sample pool
D16S403	2	2	3	3
D1S468	2	1	3	2
D3S1304	2	2	3	2
D5S211	2	3	3	4
D5S408	2	1	3	3

Table 9.10. Total number of misplaced alleles in the computed allele distributions for the different sizes of pooled samples. A total of 16 alleles from 8 individual DNA samples were determined using batches of 2-sample pools and 4-sample pools, as well as in a single 8-sample pool.

These preliminary synthetic results with post-PCR pooled samples suggested that FAST-MAP's pooled GMBD approach could potentially work well with real pooled DNA data. Post-PCR pooling does not reduce the work required to amplify the DNA samples, since each sample must be individually PCR amplified before pooling for gel run-out. To fully exploit the efficiency in pooled genotyping, it is important to pool the DNA samples *prior to* PCR amplification, so that there is only one PCR amplification experiment for each pooled sample, regardless of pooling size. Thus, to realistically assess FAST-MAP's pooled GMBD analysis algorithms, we needed to experimentally generate *pre-PCR* data.

9.4.2. Pre-PCR pooling

We conducted a new set of pooled experiments with these useful characteristics:

(a) pre-PCR pooling: DNA samples were pooled and PCR amplified as a single mixture;

(b) large pool size: DNA from up to 96 individuals were pooled together;

(c) different marker repeats: Di-, tri-, and tetranucleotide repeat markers were used to assess the presence of PCR stuttering affects FAST-MAP's automated pooled genotyping.

We genotyped each individual separately (using FAST-MAP) to generate the actual allele distributions for comparison. This step also compiled the marker libraries used in pooled GMBD. For individual DNA genotyping, DNA samples from 96 individuals were prepared in equimolar 3 ng/ul concentrations, and then separately PCR amplified using different microsatellite primers, including di-, tri-, and tetranucleotide repeat markers. The PCR product of each DNA sample was size separated on an ABI/377 sequencer (including internal lane size standards). FAST-MAP was then used to automatically determine the alleles in each sample. These individually scored genotypes collectively formed the actual allele distributions, and were used to compare with the pooled sample analysis.

For pre-PCR pooling, we pooled the DNA samples into 8 pools of 12 samples. We took aliquots from these pools to construct 4 pools of 24 samples, then 2 pools of 48 samples, and finally 1 pool containing 96 samples. (We subsequently learned that the 7th of the 8 12-fold pools contained one extra sample, but this addition did not materially affect our protocols, analysis, or results.) Each pooled template contained 48 ng of DNA in an 18 ul volume, and was part of a standard 50 ul PCR containing 2.5 units of Amplitaq Gold, 1.25ul of each primer, 200uM dNTP's, and 2.5mM $MgCl_2$. We separately PCR amplified the 15 pools using different di-, tri-, and tetranucleotide repeat markers in an MJ Research PTC-100 thermocycler (30 cycles of 94°C for 1.25', 55°C for 1', and 72°C for 1'). We size separated the PCR products on an ABI sequencer, including GS350 50 bp internal lane size standards.

The FAST-MAP analysis of our data included automated lane tracking, automated binning, automated DNA quantitation of each peak, and automated estimation of the pooled allele distribution. Table 9.11 shows the total number of misplaced alleles out of 194 alleles for the different pool sizes for a di-, a tri- and a tetranucleotide marker. We also present the graphical comparisons between the three markers in Figure 9.8.

microsatellite markers	repeat (bp)	total no. of misplaced alleles (out of 194 alleles)			
		12-sample pool	24-sample pool	48-sample pool	96-sample pool
D10S212	2	63	53	46	31
D10S1230	3	45	55	33	28
D2S1394	4	46	53	62	47

Table 9.11. Total number of misplaced alleles for the different pool sizes of pre-PCR DNA pooling. Shown here are the results of one di-, one tri-, and one tetranucleotide marker. A total of 97 DNA samples were pooled, so one of the eight 12-sample pools actually had 13 samples, one of the four 24-sample pool had 25 samples, one of the two 48-sample pools had 49 samples, and there were 97 samples in the 96-sample pool.

As with the post-PCR experiments, the pre-PCR results showed that increased pool size did not diminish the performance of pooled GMBD. In fact, for the three markers shown, the larger sample pools had fewer misplaced alleles than the smaller sample pools collectively. Most striking is the result of the dinucleotide D10S212, as compared to the (stutterless) trinucleotide and tetranucleotide. Not only did the presence of PCR stuttering in D10S212 not hinder FAST-MAP's automated pooled genotyping, but it appeared to improve the robustness of pooled genotyping. As shown in Figure 9.8, pooled genotyping of dinucleotide D10S212 was generally more accurate than the stutterless tetranucleotide D2S1394 (and at least as accurate than the trinucleotide D10S1230, if not more so). In spite of its convoluting stutter bands, D10S212 had the most accurate computed allele distribution for the 96-sample pools. Just as in single-sample genotyping where the presence of stutter signatures had helped to distinguish spurious peaks from actual data, the stutter signatures appeared to have made pooled GMBD computation more robust. In the absence of PCR stuttering, alleles are wholly represented by single data bands. Without the expectation of stutter patterns, an error in a *single* band is difficult to detect or recover, and can drastically affect the overall allele distribution computation. In the presence of PCR stuttering, however, the additional stutter bands might serve as secondary "backups" for data bands that failed to properly amplify, allowing the computer to recover from spurious errors.

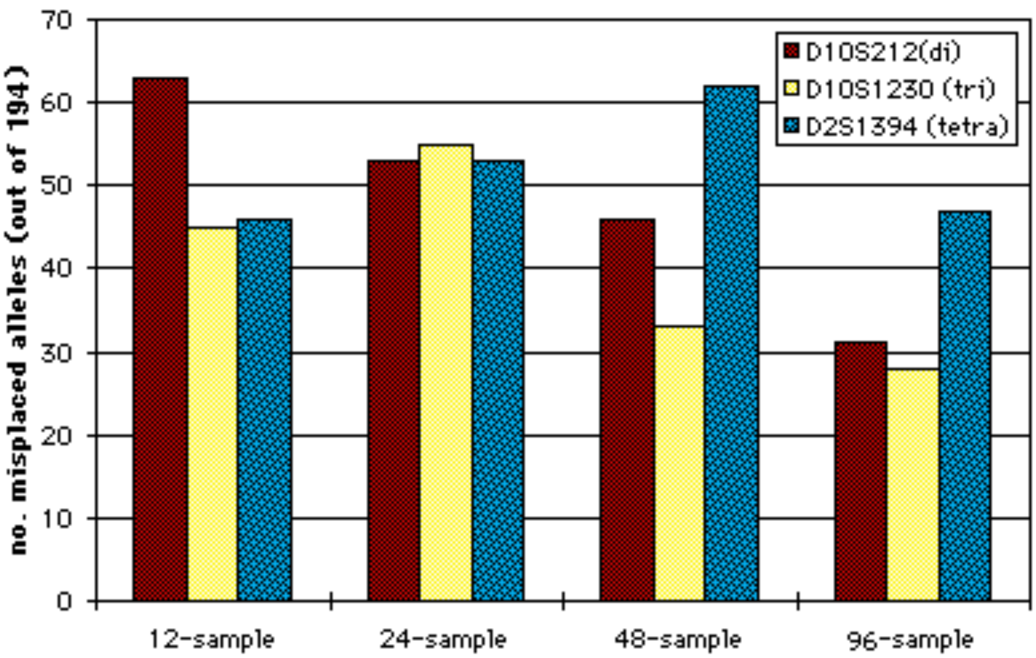


Figure 9.8. Comparisons of the total number of misplaced alleles for the three different markers in our pilot pre-PCR pooling experiments.

9.5. Mononucleotide data

The mononucleotide repeats are the most ubiquitous class of microsatellite markers, and could have been the microsatellite markers of choice for high resolution genetic mapping. Unfortunately, because of their inherent PCR stuttering and the requisite 1 bp separation for allele calling, the mononucleotides avoided for the same reasons that the dinucleotides have been traditionally avoided.

The mononucleotides are computationally similar to the dinucleotides. Just as FAST-MAP's deconvolution methods remove stutter bands from dinucleotides, they can be used to remove stutter bands in mononucleotides. Similarly, FAST-MAP's stutter-crawling method can be used to achieve the 1 bp separation required in mononucleotides on high-resolution sequencing gels. In fact, when binning the alleles by stutter crawling, the inherent 1 bp separation in the mononucleotide data (which intrinsically lacks the "plus-A" artifact) can lead to less ambiguity for FAST-MAP, since no other bands are *expected* to lie between allelic bands, unlike larger repeat sizes. Therefore, the mononucleotides might well be superb microsatellite markers for FAST-MAP's data-intensive and data-disambiguating computational processing.

Since mononucleotide repeats are non-conventional genotyping data (like the mouse and pooled DNA data), we need to experimentally generate high quality genotyping data from mononucleotide markers. One FAST-MAP user, Dr. Charles Mein from Dr. John Todd's group in the Wellcome Trust Centre for Human Genetics at the University of Oxford, generated mononucleotide data and provided us with preliminary data for FAST-MAP analysis. In the following example, a mononucleotide marker, Mono1, in the allele size range 100 to 110 bp, was amplified with TET-labeled PCR primers. 34 different DNA samples for Mono1 were loaded on an ABI/373 gel[41].

FAST-MAP performed automated lane tracking, automated binning, automated DNA quantitation of each data peak, and automated allele calling by deconvolution of each experiment on the gel without any system modifications. In Figure 9.8, we show an sample electropherogram trace for a heterozygote separated by 1 bp. The observed stutter

[41]The gel image for the MW sizing plane of this gel was shown in Figure 4.3a of *Chapter 4: Grid Construction.*

patterns of Mono1 are similar to those from a dinucleotide repeat marker (such as one with dominant "plus-A" artifacts), except that the true alleles can be as close as 1 bp apart.

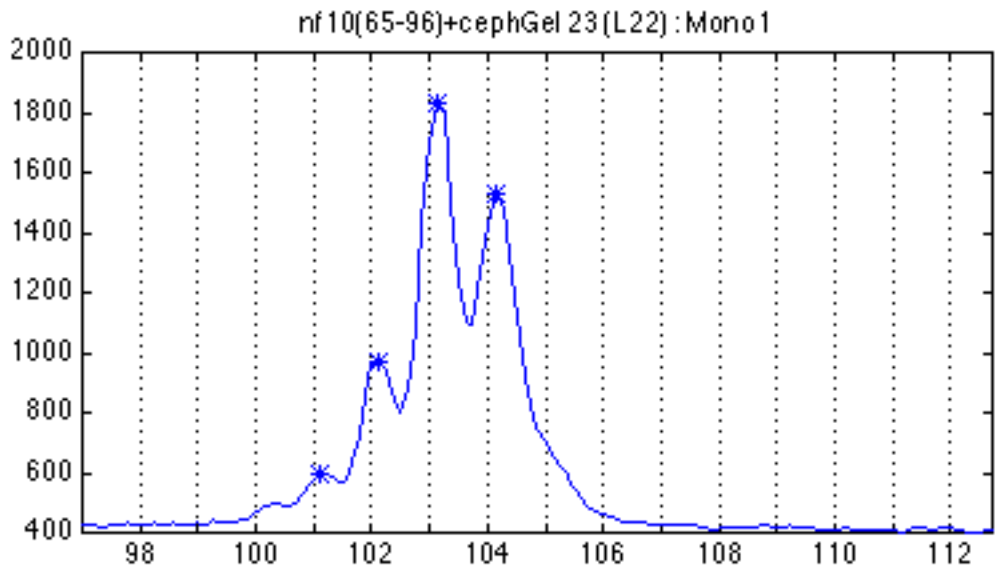


Figure 9.8. An electropherogram trace for Mono1, a TET-labeled mononucleotide marker with the size range of 100 to 110 bp. Shown here is a heterozygote which has alleles that are separated by only 1 bp (103 bp and 104 bp).

In this pilot study, FAST-MAP scored all 34 different DNA samples of Mono1 correctly. We show the electropherogram traces and the allele calls for the first 10 lanes in Figures 9.9 and 9.10. Despite the 1 bp separation between the true alleles of some of the genotypes, FAST-MAP was able to remove the PCR stutter artifacts and accurately call the alleles. Our results showed that mononucleotide data can be generated for *fully automated analysis* by FAST-MAP.

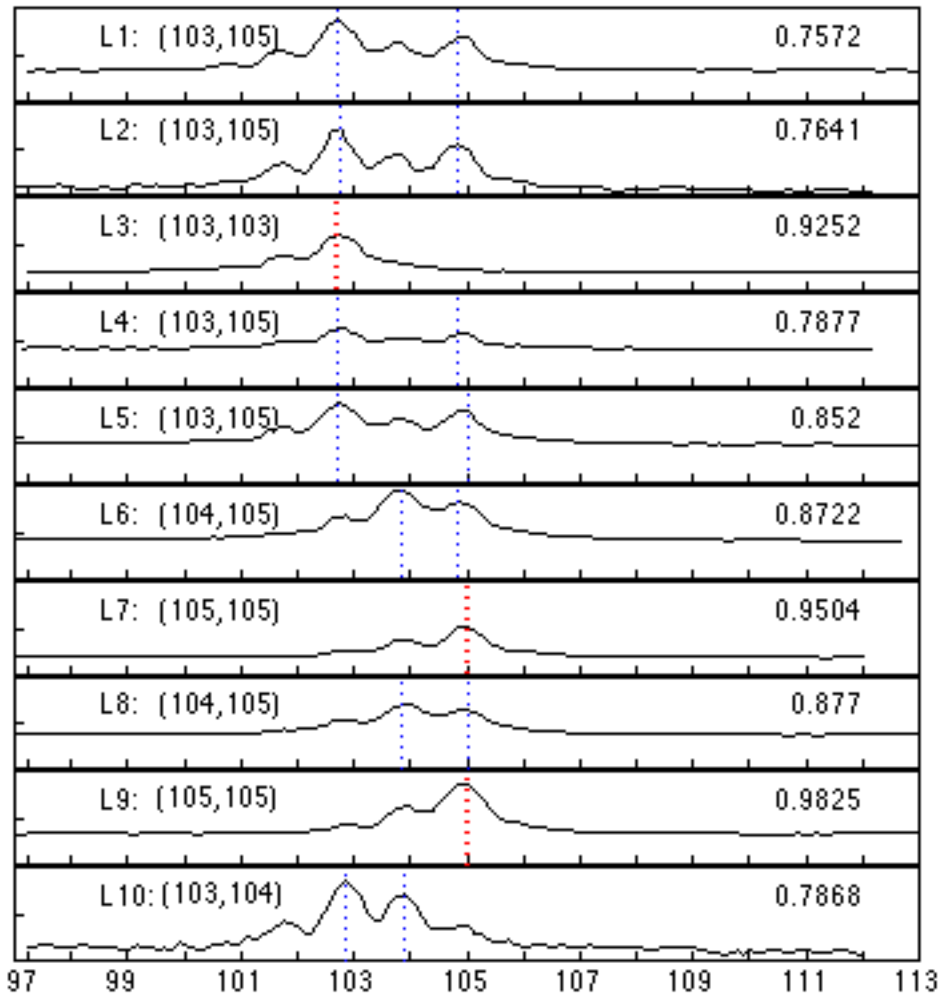


Figure 9.9. Electropherogram traces for the first 10 lanes for Mono1. The vertical dotted lines indicate the alleles called by FAST-MAP.

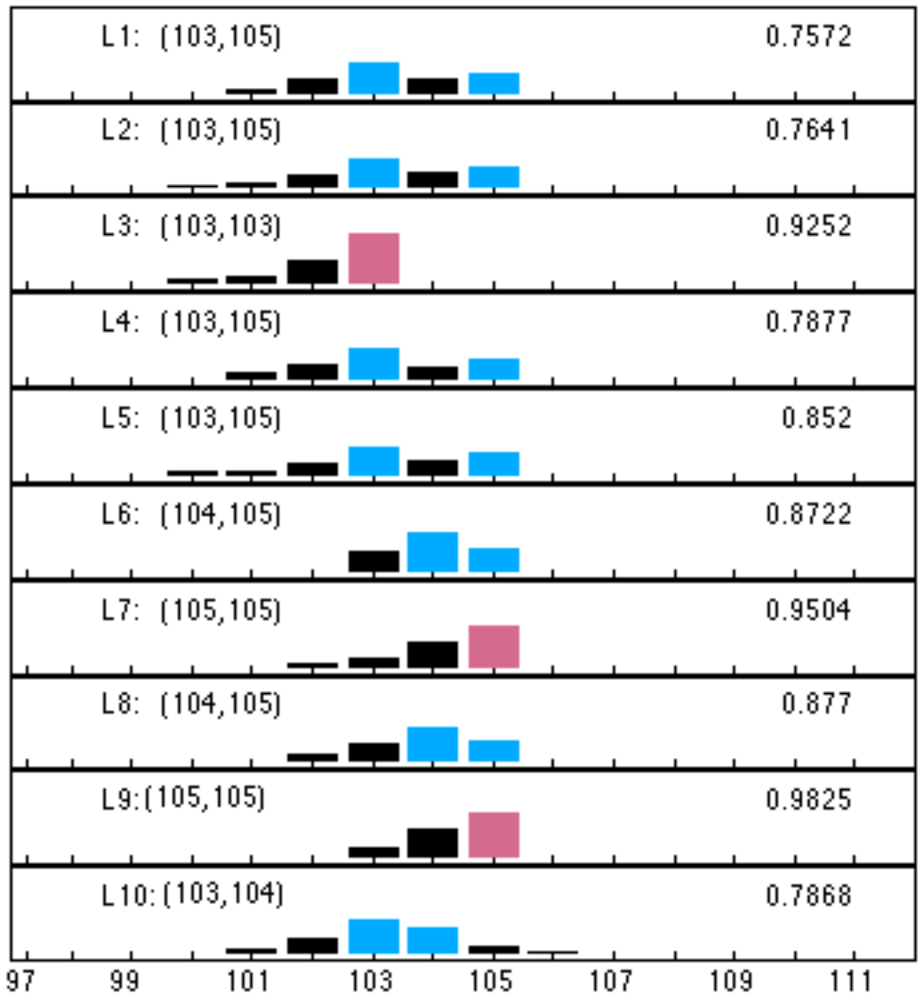


Figure 9.10. Quantitated results for the first 10 lanes for Mono1. The lightly shaded bars indicate the alleles called by FAST-MAP. Note that the alleles could be as close as 1 bp apart, as shown in lanes 6, 8, and 10, where the consecutive bars were called as two distinct alleles.

9.8. Other applications

The primary goal of this thesis is to overcome problems associated with microsatellite genotyping. Thus far, we have described the results of automating the analysis of microsatellite genotyping data from:

(a) different DNA sequencers: ABI vs. Pharmacia;

(b) different microsatellite repeats: mononucleotides vs. dinucleotides, trinucleotides, and tetranucleotides;

(c) different species: human vs. mouse; and

(d) different pooling sizes: single individual vs. pooled samples.

To explore how our computational problem solving might be applicable to other domains in molecular genetics, we went beyond microsatellite analysis and applied our computational approaches to non-genotyping problems. We describe here two non-genotyping problems: differential display analysis, and gridded filter scoring. These two problems have vastly different computational requirements. Differential display analysis involves exquisite data quantitation of lane traces on sequencing gels. Gridded filter scoring, on the other hand, is a completely different problem in molecular genetics having a scoring bottleneck that might be solved computationally. We show in this section how we can transfer our experiences in microsatellite genotyping to these non-genotyping problems in molecular genetics.

9.8.1. Differential display

Differential display (DD) was developed as a rapid and sensitive reverse transcriptase PCR (RT-PCR) technology to amplify and compare gene expression events in eukaryotic cells (Liang and Pardee, 1992). DD RT-PCR has helped identify genes in neoplasia, senescence, and development. Recent innovations have enabled technological improvements in DD RT-PCR, including the use of fluorescence-based automated DNA sequencers (Jones *et al.*, 1997; Luehrsen *et al.*, 1997). Since a key component of DD readout is accurate DNA sizing and peak quantitation, we explored the use of the technology in FAST-MAP by analyzing expressed gene sizing data from DD applications.

Our collaborator Dr. David Whitcomb at the University of Pittsburgh has extensive experience using conventional DD for gene discovery. Following our specifications, his group generated DD data on their ABI automated sequencer, using a within-lane 20 bp sizing ladder. We analyzed the data using FAST-MAP – this automated analysis included lane tracking, binning calibration, DNA peak detection, and peak overlap quantitation. Since all these DD data analysis steps are already part of FAST-MAP data processing, we were able to analyze the pilot DD data generated by Dr. Whitcomb's group with minimal system modifications. Figure 9.11 shows that the peak quantitation process in FAST-MAP can also model the peak shapes and account for peak overlap in DD data analysis.

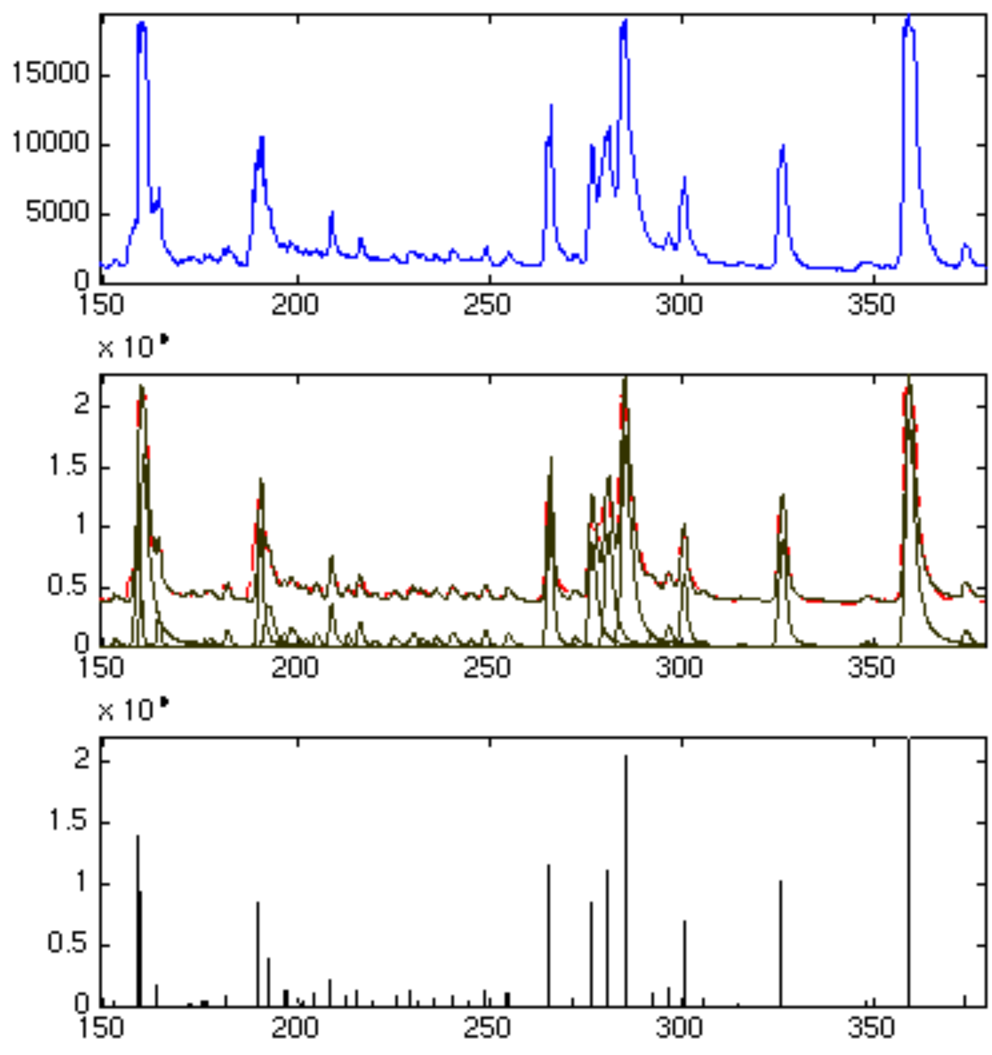


Figure 9.11. A FAST-MAP viewing window adapted for displaying quantitative DD analysis. Pane 1 shows the electropherogram trace, pane 2 the quantitative fit of model peaks to data, and pane 3 the final quantitation results.

9.8.2. Gridded filter scoring

With the advent of recombinant DNA technology (Watson *et al.*, 1992), libraries of human DNA fragments are now available for genomic analysis. These genomic libraries consist of sets of recombinant clones that are made from randomly generated overlapping DNA fragments covering the entire genome. The clones are hosted as DNA inserts in viral, bacterial, yeast, and other cloning vectors.

A major use of the clone libraries is in building physical maps, where DNA probes are localized to specific genomic regions by hybridizing against the clones. The key bottleneck is in the data analysis: determining the degree of hybridization between a probe and a clone. Since the hybridization experiments are typically performed on gridded nylon filters, we call it the *gridded filter scoring* problem.

Problem

Recombinant clones from a genomic library are generally available as sets of two-dimensional microtiter plates. Each clone is identified by its plate number and location (row and column) on the plate. For high throughput screening, an entire library is gridded onto nylon filters by high precision robots. The filters are then hybridized with one or more DNA probes (Lehrach *et al.*, 1990; Monaco *et al.*, 1991), which probes all the clones on the filter in one parallel operation. Such arrayed clone libraries can rapidly screen for a specific gene or genomic region.

The major steps in a gridded filter screening experiment are:

1. *Filter spotting:* Amplified products from individual clones are spotted onto 8×12 cm nylon filters using an N-pin robotic replicating tool (e.g. a 96-pin HDRT replicating tool of a Biomek 1000 workstation). To maximize the usage of the limited "real estate" on the filter, different N-sets of clone products are spotted on the same filter at a horizontal or vertical offset from the previous sets. By exploiting the dexterity and accuracy provided by robotic handling, a total of $N \times m \times n$ spots can be positioned on the filter, where m and n are the number of sets displaced horizontally and vertically respectively. The values of m and n currently range from 2 to 8; these numbers should increase as gridding technology advances.

2. *Probe hybridization:* After the clones have been spotted onto the filter, radioactively labeled DNA probes are then hybridized overnight against the spotted products. The filter is then exposed to an autoradiographic film for a period of time (typically 1 to 8 days).

3. *Filter scoring:* The autoradiograph image of the filter is developed. Since there is currently no effective automated scoring technology, each of the $N \times m \times n$ spots is visually inspected by an experienced human eye to *subjectively* score the degree of probe hybridization.

As with microsatellites genotyping, manual scoring is a key bottleneck precluding dense high throughput gridded filter experiments. The scoring is a labor-intensive, tedious, time-consuming, and error-prone process. Our goal, therefore, is to transfer the techniques from automated microsatellite genotyping to remove this bottleneck, and thereby enable the full automation of gridded filter experiments.

Solution

Figure 9.12 shows an autoradiograph image from a $96 \times 2 \times 2$ gridded filter experiment. Although its appearance differs greatly from an electrophoretic gel, there are many similarities in the underlying structure of the experimental data. The microsatellite genotyping problem has two main phases — sizing grid construction and marker band quantitation. Similarly, we develop an automated gridded filter scoring solution that has two phases — spotting grid construction and spot quantitation. We can adapt the algorithms from microsatellite genotyping to solve the scoring problem:

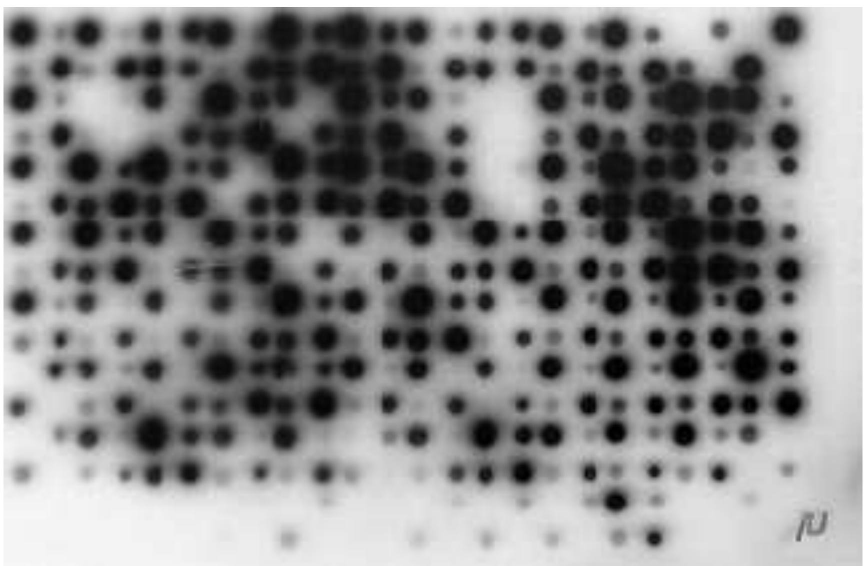


Figure 9.12 Gridded filter experiment. In a high throughput gridded filter experiment, a complex DNA probe is hybridized against clones previously spotted on a nylon filter. The autoradiograph image shown here is the output of an experiment with 384 clones arranged in a $96 \times 2 \times 2$ array. The task is to consistently score the 384 spots based on each spot's radiographic intensity.

- Spotting grid construction. The first task is to locate where the spots reside on the filter image. To do this, we construct a *spotting grid* that is similar to the sizing grid used in microsatellite genotyping. Using our design knowledge about the layout of the robotic gridding, we can construct an *expected* grid. Then, we can rapidly map this expectation onto the data on the filter image to compute an actual spotting grid for the filter. Just as with sizing grid construction, this expectation-based approach is both efficient and robust in building spotting grids for real laboratory data. Figure 9.13 shows the spotting grid constructed for the filter in Figure 9.12.

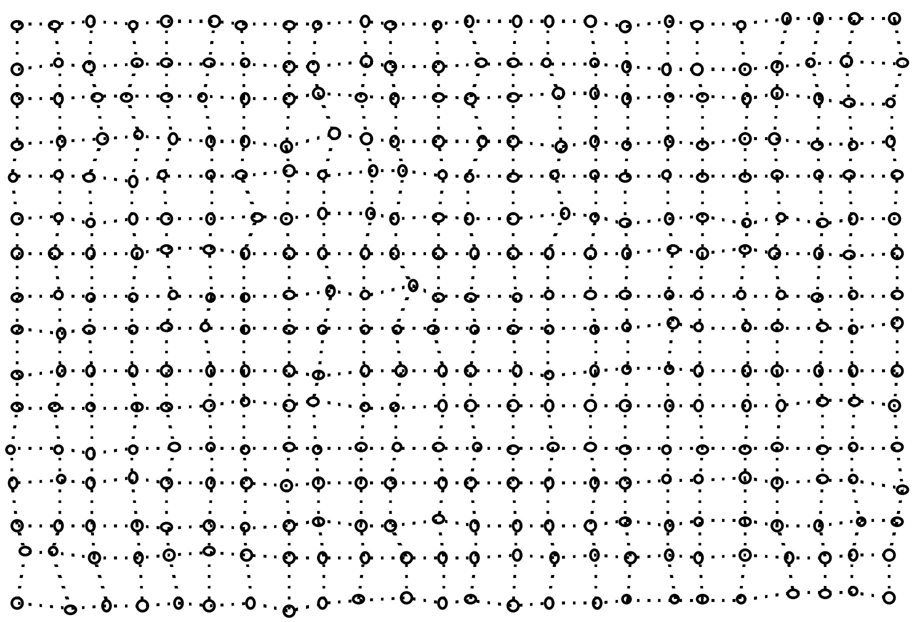


Figure 9.13. A spotting grid for the gridded filter shown in Figure 9.12. The grid was constructed automatically based on robotic gridding information. With this grid, the computer can localize the 384 gridded spots on the image, and focus its computation on the 384 local subproblems.

- Spot quantitation. Unlike DNA migration in gel electrophoresis, the gridded spots diffuse two-dimensionally on the filter. However, the same computational technique for handling marker band diffusion still applies: we use a model function similar to the *Gauss-Runga* model for microsatellite data bands to accurately quantitate the spots, as well as account for diffusion effects from neighboring spots. Here, we use a 3-D Gaussian model for the spots to model the two-dimensional diffusion:

$$\Gamma_{h,\sigma}(c_x, c_y) = h \cdot e^{-\frac{(x-c_x)^2+(y-c_y)^2}{\sigma^2}}$$

where (c_x, c_y) is the center of a spot, h the height, and σ the Gaussian half-width. Figure 9.14 depicts a typical 3-D Gauss curve.

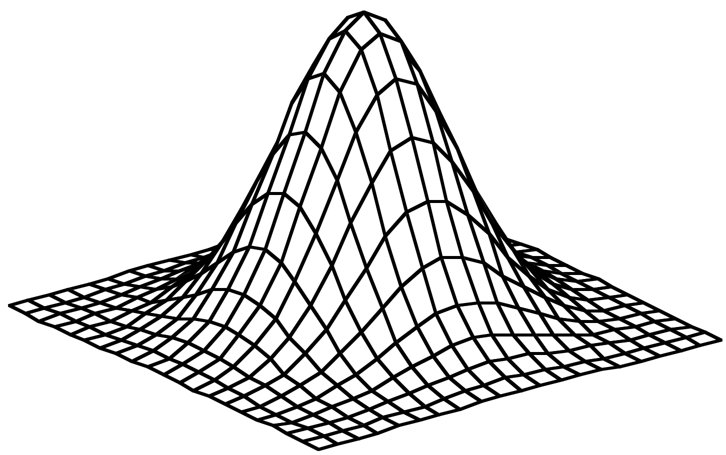


Figure 9.14. A 3-D Gaussian model for gridded filter spotting. Using this mathematical model to fit the filter spots, we can systematically eliminate diffusion effects, and quantitate the spots by computing the volume under the curve.

The model-based approach allows us to mathematically eliminate the overlapping diffusion of closely-spaced spots, and accurately quantitate the radiographic intensity of each spot (i.e. the degree of probe hybridization of each clone). Figure 9.15 shows a close-up of how the original terrain of the top left corner (first three rows and columns) compared with the fitted terrain using the 3-D Gaussian model function. Figure 9.16 shows the fitted image resyntheiszed by the computer. The score at each gridded spot was computed from the individual fitted spot volume. Table 9.12 shows the normalized scores for the top left corner that our software computed.

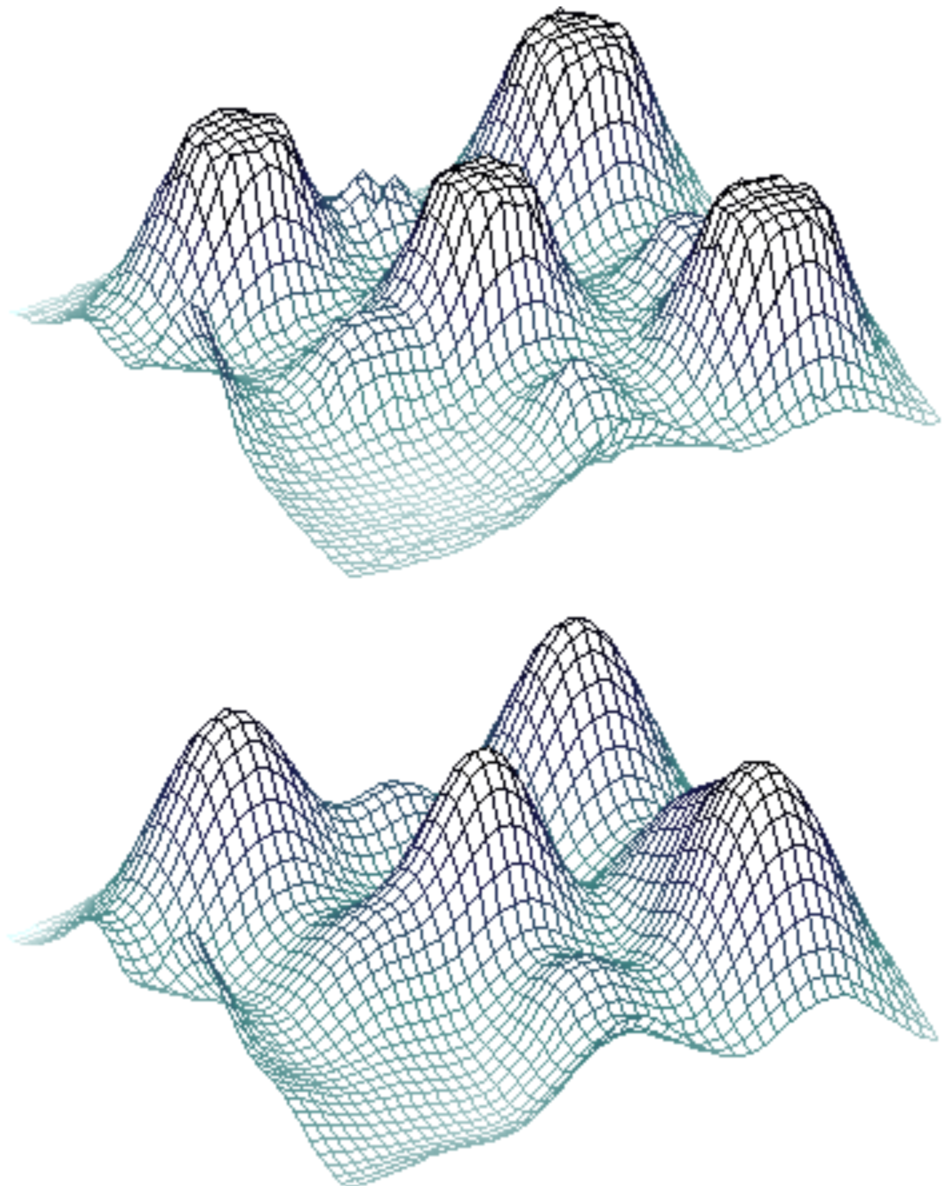


Figure 9.15. Close-up plots of the top left corner of the filter in Figure 9.12. The upper plot depicts the terrain of the first three rows and columns from the original autoradiographic image. The lower plot depicts the fitted terrain using the 3D-Gaussian model function generated by the computer. The flattening of the peaks observed in the original image was due to over-saturation of the x-ray film.

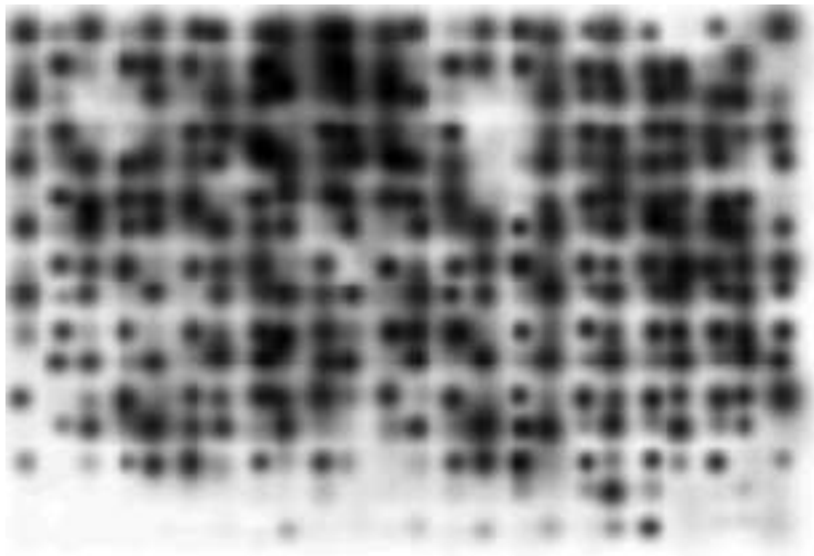


Figure 9.16. The fitted image for the filter shown in Figure 9.12. The fitted image was resynthesized by summing the fitted model of gridded spots using the 3-D Gaussian function (shown in Figure 9.14).

	column 1	column 2	column 3	…
row 1	0.91	0.53	0.90	…
row 2	0.43	0.89	0.33	…
row 3	0.90	0.49	0.12	…
:	:	:	:	

Table 9.12. Normalized scores of the top left corner of the filter shown in Figure 9.12. The scores were automatically determined based on the fitted volumes at each spot.

Discussion

The preceding example (Figure 9.12) shows that it is indeed possible to adapt computational approaches from the genotyping problem to solve a different problem — scoring gridded filter experiments. We could further improve the results by addressing two problems:

- Spot ambiguity. Although we know the spotting pattern of clones onto the filter, the pattern need not exactly correspond to the *post-hybridization* pattern observed on the autoradiograph. In the filter shown in Figure 9.12, 384 clones were spotted onto the filter in a $96 \times 2 \times 2$ array. However, not all 384 spots showed up on the autoradiograph after probe hybridization, since a spot is developed only when the radioactively labeled DNA probe hybridizes with its clone. Thus, the robotic spotting pattern only estimates the post-hybridization pattern. In the case where a DNA probe does not hybridize with most of the clones (as in the filter shown in Figure 9.17), it is impossible to construct the spotting grid using only robotic spotting information.

- Spot over-saturation. A second problem is that the autoradiographic signals from the spots tend to be over-saturated, resulting in "flattened" peaks as shown in Figure 9.15. The 3-D Gaussian model does not account for this over-saturation effect. Using an alternative "flattened" gaussian model function would not solve the problem either, as not all spots are over-saturated.

Problem simplification is a recurring practical strategy in this dissertation. We have seen, in microsatellite genotyping, how we can simplify a problem *algorithmically* by reducing it into simpler subproblems, such as reducing the two-dimensional problem of sizing grid construction into the one-dimensional problems of lane tracking and MW calibration. In a domain like molecular genetics, where data are generated in a laboratory, we can often simplify a problem *non-algorithmically*. This is done by changing the nature of the problem, such as improving the data quality to experimentally eliminate confounding artifacts (e.g., using the "G-clamping" or "PIG-tailing" techniques to remove the "plus-A" artifacts from microsatellite genotyping data), or modifying the format of the data to assist the computer in its analysis. Interdisciplinary problem solving is a collaborative effort and often requires interdisciplinary solutions.

To solve the problem of spotting ambiguity in grid construction, we can improve the certainty of the spotting patterns by introducing "reference spots". For example, we can anchor the spotting grid with positive spots by spotting positive DNA controls onto the grid to guarantee hybridization. The filter in Figure 9.17 has a positive reference spot in the top left corner of each cluster. The computer can construct the spotting grid based on the positive reference spots, and then interpolate for the other cloning spots. Figure 9.18 shows a spotting grid automatically constructed using the positive reference spotting information.

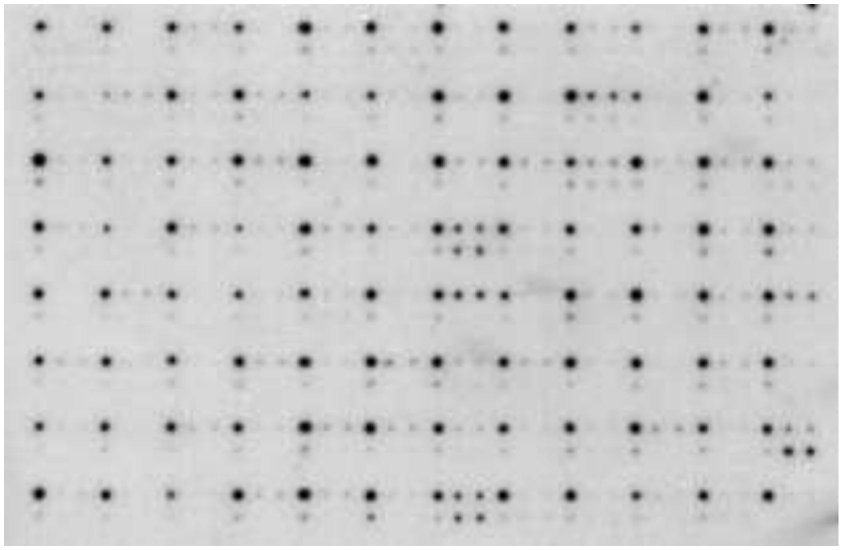


Figure 9.17. A gridded filter with positive reference spots. The robotic spotting here is a $96 \times 3 \times 2$ array, with 384 clones (i.e., 96×4), 96 spots reserved as positive references (top left spot in each of the 96 clusters), and 96 spots reserved as negative references (bottom left spot). Unlike the filter in Figure 9.12, most of the clones on this filter do not hybridize with the DNA probes. It would therefore be problematic to automatically construct a spotting grid for this filter without positive reference spots.

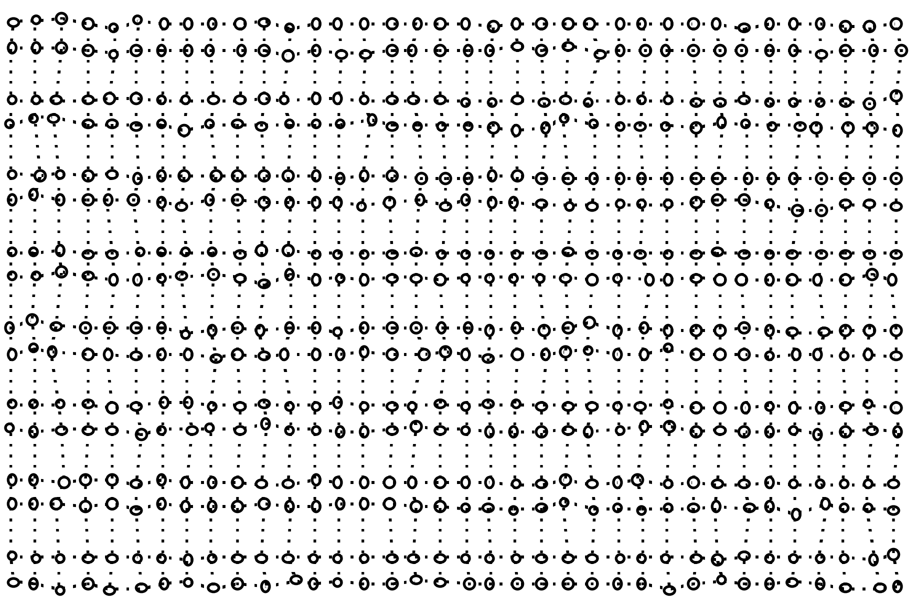


Figure 9.18. The spotting grid for the filter shown in Figure 9.17. This grid was automatically constructed from the positive reference spots, and interpolated to detect the remaining cloning spots.

To solve the spot over-saturation problem, we can shorten the exposure time of the autoradiographic film, or taking a snapshot periodically during the exposure to keep track of the signal level. Figure 9.19 shows a close-up plot of a middle section of the filter in Figure 9.17 (rows 7, 8, 9, and columns 19, 20, 21, 22). For comparison, we show the plots from the original image, as well as from the fitted image. Since the degree of over-saturation has been minimized for this filter, the two plots show a closer match than those depicted in Figure 9.15. However, some of the (reference) peaks from the original image are still flattened[42], indicating that a better solution would be to model the over-saturation effect using a different fit function, and introducing a saturation threshold factor. Nevertheless, the computer's fitted image shown in Figure 9.20 shows a very good match with the original image in the data peak regions.

Table 9.13 gives the normalized scores for the twelve spots shown in Figure 9.19. The scores were computed from the fitted volumes of the spots using the 3-D Gaussian model function.

	…	column 19	column 20	column 21	column 22	…
:	…	:	:	:	:	…
row 7	…	0.94	0.67	0.58	0.89	…
row 8	…	0.25	0.55	0.63	0.20	…
row 9	…	0.92	0.67	0.58	0.67	…
:		:	:	:	:	

Table 9.13. Normalized scores of the 12 spots in the middle cluster shown in the close-ups in Figure 9.19. The scores were assigned based on the estimated volumes of the fitted spots.

[42]By ensuring that the reference spots have the most intense signals, and minimizing over-saturation of these reference spots, we can avoid over-saturation at the clone spots.

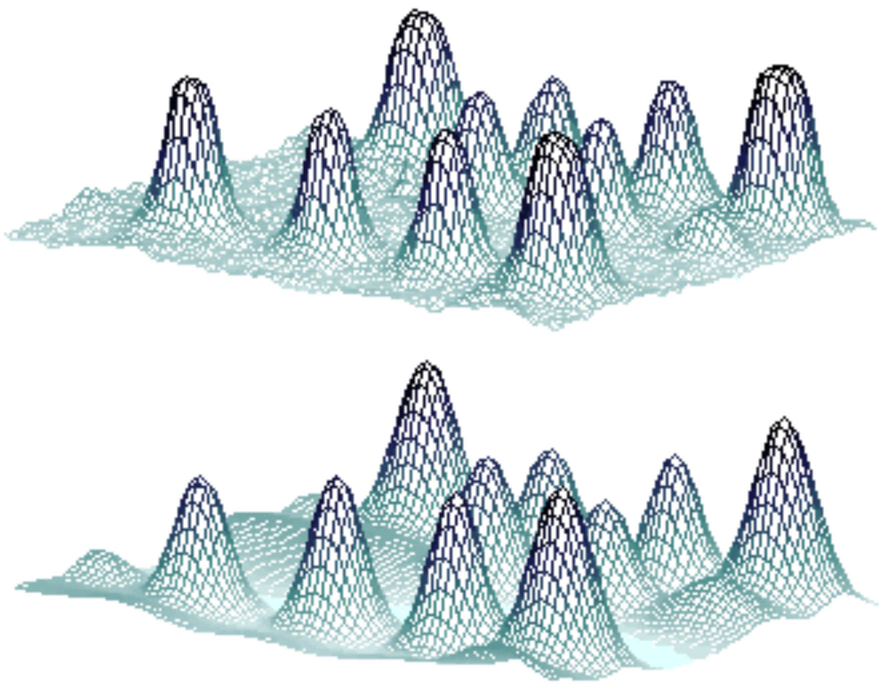


Figure 9.19. Close-up plots of a middle cluster (rows 7, 8, and 9, and columns 19, 20, 21, and 22) in the filter of Figure 9.17. The upper plot depicts the 3D close-up from the original autoradiographic image. The lower plot depicts the fitted terrain using the 3D-Gaussian model function generated by the computer. Although we have tried to minimize spot over-saturation in this filter, some of the reference spots (e.g. the ones at row 7 and columns 19 or 22) still display slightly flattened peaks.

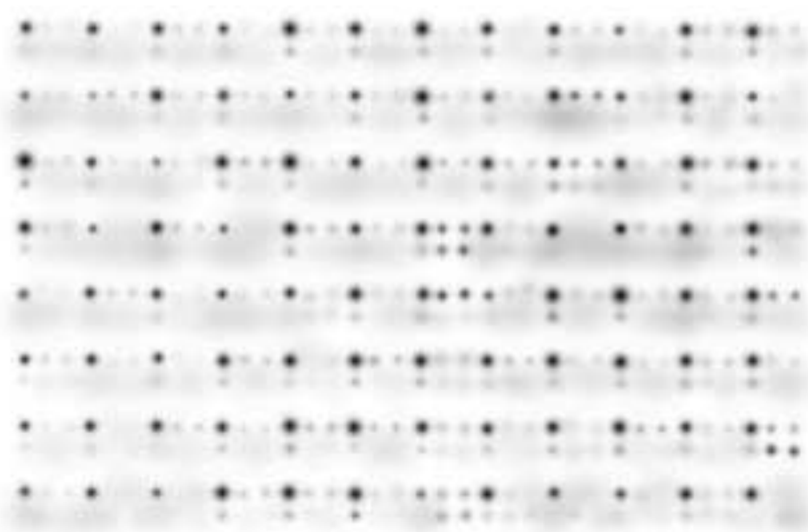


Figure 9.20. The fitted image resynthesized by summing the fitted 3D-Gaussian model functions at each of the gridded spots. Note that the background in the original image in Figure 9.17 has been removed from in the fitted image.

10. Conclusions

This dissertation shows how we eliminated the scoring bottleneck in microsatellite genotyping using computational problem solving methods. By studying the science and technology behind related molecular biology domains, we elucidated the computational structure underlying the problem. Based on this structure, we divided the problem into manageable subproblems. We then systematically solved these subproblems traditionally using fundamental computer science techniques, and creatively using novel approaches — such as exploiting PCR stutter artifact as a useful data component.

Since a key goal is to develop *useful* technology, we have fully implemented our solutions in FAST-MAP, a practical microsatellite genotyping software system. FAST-MAP is used by geneticists in their laboratories for analyzing genotyping data. By actively including real users and their data in our system's development and testing cycles, we were able to design, implement, and (continually) refine a software system that is practical, robust, accurate, and elegant to both developer and user. Such synergy between computer scientists and geneticists is crucial in successfully solving complex cross-disciplinary problems, such as the microsatellite genotyping problem.

10.1. Contributions

The major contribution of this thesis is completely removing the current bottleneck in microsatellite genotyping. We built a genotyping data analysis system that is fully automated, robust, accurate, and efficient. Specifically, we:

- created new computational methods that account for intrinsic data artifacts (such as PCR stuttering) by accurately modeling the underlying processes;
- devised novel computer-based techniques to overcome data limitations (such as inadequate MW sizing resolution) by exploiting all available data for calibration; and
- employed efficient learning strategies that improve the system's performance by automatically incorporating specific knowledge about genotyping data into the system.

Moreover, we explored and enabled new functionalities (such as pooled DNA genotyping for population genetics), and applied our computational strategies to solve other non-genotyping problems in molecular genetics (such as differential display analysis and gridded filter scoring).

Most importantly, this thesis has demonstrated that computational strategies can be a powerful tool in overcoming critical molecular biology bottlenecks. We have shown, with good results in microsatellite genotyping, that techniques in computation and information science can be as useful as laboratory innovations for solving key problems in molecular biology.

The current genetic revolution is expected to bring about an explosion of genetic information. We believe that the future of molecular genetics will be shaped simultaneously in traditional "wet" (molecular biology) labs and in "soft" (computer) labs. Hence, the most valuable contribution of this thesis to both computer science and genetics is the introduction of *computational problem solving* into molecular genetics. We hope that this thesis has helped bridge the experimental world of molecular biology with the computational world of computer science.

10.2. Future Work

In solving the microsatellite genotyping problem, we enabled new functionalities in molecular genetics. We also opened up new areas of research for computational problem solving. We provide a selective preview of our ongoing and future research.

10.2.1. FAST-MAP refinements

We plan to refine FAST-MAP on a diverse range of data (of varying quality) to further improve accuracy and robustness. Example projects include:

(a) Learning to improve accuracy from past experience.

FAST-MAP was designed to be a fully automated genotyping system, requiring little or no user intervention. Thus, we did not provide FAST-MAP with a functionality that allows it to learn directly the user's feedback to its mistakes (since this would require extensive user interaction, which would contradict the fully-automated design goal). However, when FAST-MAP repeatedly makes the same mistake, the user *should* be able to teach the computer by entering correct user edits. These edits could be flagged in marker libraries, so that FAST-MAP could recompute calibrations from user-corrected data, instead of working from incorrect assumptions.

(b) Adding more independent allele determination algorithms.

FAST-MAP would benefit from additional allele determination algorithms, besides SVD and ENUM. As both SVD and ENUM call alleles based on quantitated data, one useful algorithm would call alleles directly from the observed electropherogram signal, without intervening quantitation steps. Such an algorithm might be more sensitive to input data characteristics, such as baseline shifts and incorrect peak shapes in the electropherograms.

(c) Supporting more computer and sequencer hardware platforms.

FAST-MAP currently reads data formats from two of the most popular sequencer machines: the ABI and Pharmacia automated DNA sequencers. Many new instruments (e.g. from Genomyx, GenSys, Hitachi, LI-COR, and Molecular Dynamics) are becoming available. FAST-MAP's underlying modularized design should let it handle image data from these new machines. Moreover, its subsequent image and genotype processing are independent of the sequencer platforms. By adding input modules that translate new image file formats into FAST-MAP's universal image format, FAST-MAP should effectively process data from virtually any DNA sequencing machines.

(d) Linking to other data and software.

Most genetic linkage studies involve access to diverse databases (e.g., ENTREZ, GDB), and application of various software tools (e.g. LINKAGE, SIBPAL, GENEHUNTER). For a networked computer, most of these resources are readily available on the Internet. To greatly improve the geneticist's productivity in conducting genetic linkage studies using microsatellite genotyping, we can extend FAST-MAP into an integrated genotyping system that (1) provides easy on-line access to databases, and (2) allows direct transfer of results to genetic analysis software.

10.2.3. Pooled DNA analyses

Our pooled DNA analysis pilot study demonstrated that one can generate and analyze pooled DNA templates using microsatellite markers. Interestingly, these results included dinucleotide repeat markers with PCR stutter artifact. Our current software solves two critical problems: accurate peak quantitation and PCR stutter removal. However, we only partially solved the third signal analysis problem — adjusting the data for relative allele amplification. To date, we have used the amplification ratio table compiled from individual genotypes to estimate the relative allele amplification in pooled DNA data. However, relative amplification in pooled alleles may differ from relative amplification between a single allele pair. To correct for relative amplification in pooled DNA, the ratio of a marker's pooled distribution to the sum of its individual components might provide a more realistic calibration curve than pair-wise ratios. This new ratio function might depend on various factors, including pool size, PCR primers and conditions, and allele distribution. Precalibrating such curves for each marker (across varying pool complexities) might permit mathematical correction for relative amplification. To fully realize the potential benefits of pooled DNA analyses, we plan to further investigate the pooled relative allele amplification problem.

10.2.4. Genetics in non human species

Mammalian and avian species contain microsatellite markers. In particular, there are many species of biomedical (e.g., mouse, rat, hamster) and agricultural (e.g., cow, pig, sheep) importance that contain useful microsatellite markers. It would therefore be very useful to have an automated genotyping technology that is not restricted to the human species.

Much of power of the animal model derives from inbred strains that can be cross-bred and genotyped to map complex traits, and ultimately elucidate mechanisms of disease. As exemplified in our mouse data, genotyping of inbred strains has an important property: each marker is biallelic. We have shown how this biallelic property introduces a constraint that can be used by FAST-MAP for more rapid and accurate genotyping of microsatellite markers. Moreover, by focusing on recognizing only the two possible alleles for each marker, FAST-MAP might enable sophisticated multiplexing of panels for greater throughput. These new applications will be challenging computational problems that might be important to molecular genetics.

10.2.5. Forensics and personal identification.

Microsatellites are currently used in forensic science and for personal identification. Most current and proposed microsatellite forensic panels are comprised of 12-16 tetranucleotide repeat markers. The larger repeats were introduced because of historical problems with dinucleotide repeat PCR stutter. These problems have been overcome by our pattern recognition methods. Most importantly for forensic applications, PCR stutter provides additional information that we have shown to be useful in distinguishing true alleles from spurious peaks, resulting in more robust personal identification than with tetranucleotides. As forensic and personal identification applications require exquisite certainty and precision measures, we plan to conduct further research in exploring novel experimental designs, statistical analyses, and computational strategies in exploiting the additional data redundancy provided by dinucleotide stutter data. The results might increase accuracy and throughput, at a lowered cost.

10.2.6. Using dense genotyping in preventive medicine

By reducing the cost of fully automated microsatellite genotyping, we are closer to acquiring high-resolution genetic snapshots of individuals and their relatives in the population. From such snapshots, we can determine who shares which inherited chromosomal regions with whom. When this genotype information is coupled with large scale phenotype information, the geneticist may be able to ascertain correlations between common genetic diseases (e.g., cancer, stroke, diabetes) and their genomic locations from the highly informative (and highly complex) chromosomal and phenotypic inheritance patterns. Many new questions and possibilities arise, many of which are computationally intensive because of the vast amount of data involved. This opens up challenging unexplored avenues for both modern genetics research and computational problem solving. Some questions include:

- Are particular patterns of inheritance associated with specific varieties of disease?
- Can the relative risk of a common (complex) disease be determined from such patterns of inheritance, without knowing the actual genetic composition of individuals (see Figure 10.1)?
- What additional genetic and phenotypic information is needed to ascertain disease risk?
- Can the practice of medicine be fully customized to each individual's genetic composition?
- Can medical or environmental interventions reduce the risk of common genetic disease?

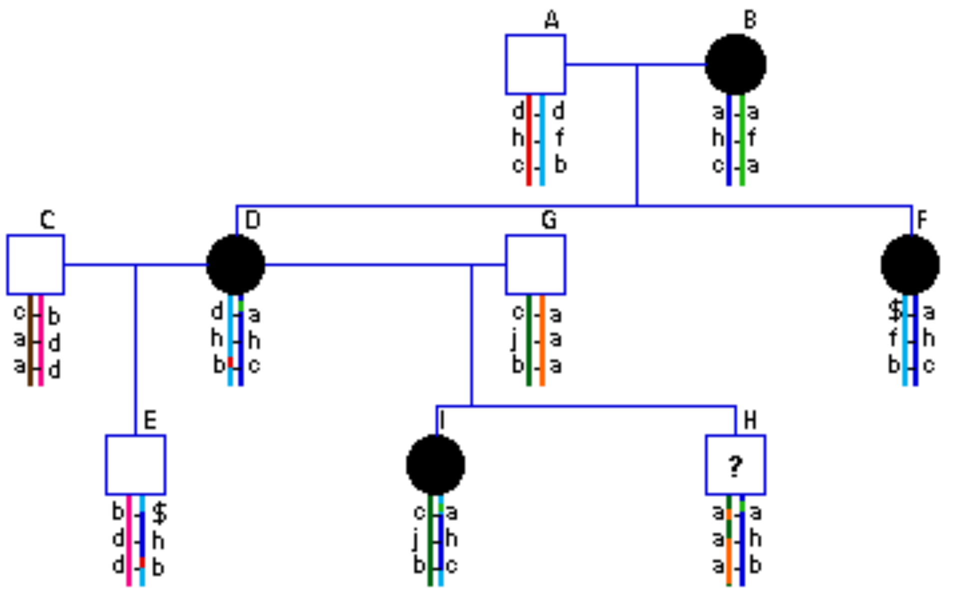


Figure 10.1. Assessing the risk of an individual to a complex genetic disease from dense genotyping data. In the example shown, the disease risk is increased by the occurrence of either an allele *a* at the first locus or an allele *b* at the second locus, with the co-occurrence of an allele *c* at the third locus. The two shaded bars represent the pair of chromosomes inherited by each individual, with the exact locations of the disease gene loci marked, from top to bottom, with horizontal lines between the two chromosomes (here, all three loci reside on the same chromosome). In this display, the alleles are labeled alongside the loci, with *$* indicating a mutated allele. With dense genotyping, the only genetic information available is the inheritance patterns as depicted by the shadings on the chromosomes. In particular, *no* gene allele information is used. To practice preventive medicine using dense genotyping data, the issue is such these shaded patterns are sufficient to infer an individual's (say, H's) risk of disease.

10.3. A Scenario for the Future

With the full automation of large scale, high throughput genotyping, population-wide full-genome scanning might become technically and economically feasible (and perhaps also socially and politically acceptable). Using a standard set of polymorphic markers for dense genotyping, an abundant collection of genotypic data that was previously too expensive to generate could become readily available. This data could be *shared* and *reused* for many projects, such as the localization of disease genes for many of the thousands of human genetic diseases.

In addition to routine disease gene identification, dense genotypic data could also be coupled with large-scale phenotypic data of the population to identify non-genetic factors that contribute to the development of complex disease. This could help isolate all risk factors, both genetic and non-genetic. With a clear understanding of multifactorial genetic disorders, a geneticist might accurately assess, for each individual, their predisposition for common human diseases. Physicians would then prescribe customized treatment *and* prevention plans according to these personalized DNA profiles. One can foresee a paradigm shift in healthcare from the current focus on *treatment,* to a more effective *preventive* approach.

With preventive medicine, curable disorders could be detected and remedied before irreversible damage occurs. The risk of certain multifactorial disorders might be reduced by moderating the environment. For example, a proper genetically prescribed program of diet, exercise, and pharmaceuticals administered from childhood might significantly reduce the risk of disease later in life. In this way, genetic information technologies might significantly improve longevity, the quality of life, and health care costs.

Appendix A: Stutter Data Simulation

To compare the various deconvolution methods that we have designed for genotyping microsatellite markers(Perlin *et al.*, 1995), we implemented a software program in Common LISP for generating simulated microsatellite markers and their genotyping data. As we were not aware of the relative amplification artifact when we conducted the simulation study, it was not included in our simulation models. However, we did apply a realistic noise model to ensure that the simulated data were as close to real data as possible.

A.1. Markers

As we were only interested in the performance of our deconvolution algorithms with microsatellite markers that exhibit pronounced PCR stuttering, we simulated only dinucleotide repeats. For each marker, we simulated 10 to 25 possible alleles, normally distributed over the size range from 100 to 200 bp. Associated with each marker was a stutter matrix A that was simulated as follows: each column of A , which corresponds to an allele's real-valued stutter pattern vector, was generated using an exponential decay rate that was inversely proportional to allele size. The columns in A were then normalized to sum to unity since they each represented a relative DNA mass of a single allele. In our simulation, we randomly varied the expected number of stutter bands for each marker. To simulate minor to extreme PCR stuttering, our markers have trailing stutters that ranged from 3 to 12 bands.

For our comparative study of deconvolution methods, we generated a library of 150 simulated microsatellite markers, as well as their stutter matrices and allele distributions.

A.2. Data

Using the markers' simulated stutter matrices and allele distributions, we generated thousands of genotyping data vectors for each marker. To generate a genotyping data vector y for a marker, we must first generate a random genotype vector x by drawing a pair of alleles from the marker's possible alleles set. Each allele was drawn based on the marker's allele frequencies and distribution. With the genotyping vector, we can then compute the data vector y as the convolution product Ax. Finally, each simulated data vector was subjected to an additional noise component (see A.3).

A.3. Noise

Our model of noise consisted of two components: (1) random background noise N_b, and (2) scaled ($k\%$) normally distributed measurement error $N_{m,k}$. For a data vector y, the total simulated noise reading is the sum $N_b+N_{m,k}(y)$ with a preset noise level of $k\%$.

To model random background noise, we implemented N_b as a uniformly distributed random variable within the interval $[-x_{\min}, +x_{\min}]$, where *xmin was* the minimum measured value in the simulation. To model the scaled measurement noise, we implemented $N_{m,k}(y)$ as a function of the original data vector y, such that $N_{m,k}(y)$ is a normally distributed random variable with zero mean and a variance scaled relative to y such that

$$\text{Prob}(-kx \le N_{m,k}(x) \le +kx) > 0.99$$

In this way, the measurement error was simulated within the preset noise level k with high probability. To study the performance of our deconvolution algorithms under different levels of noise, we used 0 to 15% for k in our study.

Appendix B: FAST-MAP's User-Annotated Knowledge Base

When calling the alleles manually, the human genotyper makes use of considerable contextual information. Information such as the design of the study, the layout of the marker multiplexing, and the stuttering characteristics of the markers are necessary for interpreting the gel image data. Such information is similarly needed by FAST-MAP for its genotyping with *allele_call.*

To provide the necessary contextual information, the user must annotate two classes of files prior to running *allele_call* on a gel or study:

- user files: These files reside in a common directory ("FAST-MAP/user/") that is globally accessible by all gels and studies in the system. They contain annotation information that can be used in the analysis of any gels or studies in the system:
    - preferences: contains user preferences and system-specific settings;
    - nicknames: contains aliases for gel and study directories, as well as any shared matrix files (for dye-separating the gel files) or sample sheets;
    - dyes: contains aliases for the various dye names used in the sequencers;
    - size_stds: contains information about various size standards;
    - markers: contains information on the markers;
    - panels: contains definitions of panel names;
    - pedigrees: contains pedigree definitions (in LINKAGE format).

- input files: These files are associated with a particular gel or study. They reside in the "input/" directory of the corresponding gel or study directory, and contain information pertaining to the gels or studies that they are associated with. There are only two user input files:
    - settings: contains gel or study-specific annotations as well as execution preferences;
    - layout: for a gel, contains the gel layout (which lanes are loaded with what panel or size standards); for a study, contains a list of the nicknames of the component gels that make up the study.

All the user-annotation files are header text files[43] (i.e., table representation), a format that is easy for both people and computers (database and spreadsheet programs) to use. In FAST-MAP, the user can annotate both the user and input files at any time using the "edit" function. FAST-MAP maintains this user-annotated knowledge base dynamically by keeping track of the changes to the user-annotated files. Collectively, these user-annotated files and the FAST-MAP libraries (described in Appendix C) constitute the complete knowledge base for FAST-MAP for its intelligent processing.

B.1. Global information: User files

The global information are stored in the *user files,* so called because they reside in the common "user/" directory that is shared by all the gels and studies. In this section, we describe the information stored in the various user files.

B.1.1. preferences

The "preferences" file specifies the computer environment in which the FAST-MAP programs are run. The user MUST annotate the "preferences" file correctly before attempting to run FAST-MAP.

Here is a typical "preferences" file:

```
attribute                      value

% (A) EDITING preferences:
% Space_char  - special character to use in place of SPACE in pathnames
% Editor_name - editor to use for 'edit'/'ed'
%
% (B) DISPLAY preferences:
% Display_type    - display images in 'color' or 'grayscale'
% Mouse_clicks    - use 'single' or 'double' clicks for mouse
% Watch_cursor    - show watch cursor while waiting
% Gel_orientation - display gel image in 'reverse' orientation (smaller
%                   bps on top) or 'normal' (bigger bps on top)
%
% (C) PRINTING preferences:
% Printer_name       - name of printer to send file/plots to
% Printer_type       - 'ps' for postrcript or 'psc' for color postscript
```

[43]A "header text file" is a text file where each line represents a record of information, and the whitespace (e.g., blanks or tabs) in each line separate the different information fields. The first line of the file specifies the column headers that name the different information fields.

```
% Print_orientation  - orientation of the printouts

Space_char                    ~
Editor_name                   emacs

Display_type                  color
Mouse_clicks                  single
Watch_cursor                  yes
Gel_orientation               reverse

Printer_name                  stone
Printer_type                  ps
Print_orientation             landscape
```

Detailed descriptions

Here are the definitions of the various attributes in "preferences":

```
attribute                     value
```

The header tells FAST-MAP that the first column is comprised of "attribute"s that FAST-MAP can set when the program is run, and that the second column contains "value"s for these "attributes". Each subsequent line therefore contains two entries: the attribute name describing the property to be set, and the value of that property.

```
Space_char                    ~
```

The special character to use in place of SPACE in pathnames. For example, to define a pathname like the following (as typical in Macintosh):

```
My Macintosh:FAST-MAP Data:Gel 12/22:
```

Type:

```
My~Macintosh:FAST-MAP~Data:Gel~12/22:
```

The user may define "`Space_char`" to any character as long as it is never used in the pathnames. The default is the tilde character "~".

`Editor_name`   `emacs`

The `"Editor_name"` attribute tells FAST-MAP the editor that the user prefers to work with. The value of this attribute is the operating system (e.g., UNIX) command that one would type to start the editing program. For example, this editor command can be the name of the editor, or a path name to its binary executable file if it is not on the operating system's search path.

The user must set `"Editor_name"` for FAST-MAP. On UNIX, the recommended editors are "emacs" and "vi", which are usually bundled with most UNIX systems. The user may also set it to a spreadsheet program (such as Excel on Macintosh) as long as it inputs and outputs tabbed-text format. On Macintosh, FAST-MAP will bring up Matlab's own editor if `"Editor_name"` is not set by the user.

`Display_type`   `color`

`(color)`	Display images and figures in color.
`(gray)`	Display images and figures in grayscale.
`(default)`	`color`

This attribute specifies the display capability of the computer monitor.

`Mouse_clicks`   `single`

`(single)`	Detect single mouse-clicks.
`(double)`	Detect double mouse-clicks.
`(default)`	`single`

This attribute specifies whether a single or a double mouse-click is to be detected by FAST-MAP to invoke mouse-click-activated actions.

```
Watch_cursor             yes
```

(yes)	Display a watch cursor while waiting.
(no)	Do not display a watch cursor.
(default)	yes

This attribute specifies whether a watch cursor should be displayed in a busy window. On some system, turning this feature off may speed up display time.

```
Gel_orientation          reverse
```

(normal)	Display gel with bigger bps on top.
(reverse)	Display gel with smaller bps on top.
(default)	reverse

This attribute specifies the preferred orientation of gel images in the displays.

```
Printer_name             slate
```

The attribute tells FAST-MAP the name of the preferred printer. FAST-MAP assumes that the computer can access the printer specified. For example, to print a "file" on a printer named "slate" in a UNIX system, one would type:

```
> lpr -Pslate "file"
```

If this commands works from UNIX, then one can set the `"Printer_name"` attribute to the value "slate". On a Macintosh, the user can use the "Print" command under the "File" menu. In this case, since the "Print" menu command uses the printer specified by the Macintosh's "Chooser", there is no need to define this attribute.

`Printer_type              ps`

`(ps)`	Postscript printer.
`(psc)`	Color postscript printer.
`(default)`	`ps`

This attribute specifies type of the printer as specified in "`Printer_name`". Currently, FAST-MAP only supports Postscript printers on UNIX. On a Macintosh, there is no need to define this attribute since the user can always use the "Print" menu command to print.

`Print_orientation         landscape`

`(landscape)`	Print in landscape orientation.
`(portrait)`	Print in portrait orientation.
`(default)`	`landscape`

This attribute specifies orientation of the printouts. Again, on a Macintosh, there is no need to define this attribute as the user can specify the printout orientation under "Page Setup" with the "Print" menu command.

B.1.2. nicknames

The "nicknames" file is where the user can define easy-to-remember aliases to pathnames of various data objects in FAST-MAP. For rapid data accesses, the user can refer to these nicknames instead of having to remember (and type in) the complete pathnames.

There are four types of objects that can be defined in "nicknames": (1) gel directories; (2) study directories; (3) shared matrix files; and (4) shared sample sheets. Although FAST-MAP imposes no restrictions on how the user organizes the "nicknames", it may be useful to divide the "nicknames" file into four sections, each corresponding to an object type. By organizing the "nicknames" in this way, the user can take advantage of the "type" column's capability of inheriting the value from a previous row if left unspecified. Here is how a typical "nicknames" may look like:

```
nickname       pathname                                          type
%
% Section 1: Gel Directories
%
GEL1     /afs/cs/project/genome/demo/gels/Gel1/                gel
GEL2     /afs/cs/project/genome/demo/gels/Gel2/

%
% Section 2: Study Directories
%
STUDY1   /afs/cs/project/genome/demo/studies/Study1/           study

%
% Section 3: Shared matrix files
%
MATR211  /afs/cs/project/genome/demo/matrices/R211.matrix      matrix

%
% Section 4: Shared sample sheets
%

% None at present
```

The "nicknames" file must be annotated by the user when a new gel or study is added to FAST-MAP. Periodically, the user should also remove obsolete nicknames from the file for faster performance.

Detailed descriptions

Here are the definitions of each of the three columns in "nicknames":

```
nickname                      pathname                      type
```

The header tells FAST-MAP that the "nicknames" user file contains three columns. Each row describes a FAST-MAP object by stating first the object's alias under `"nickname"`, followed by the full `"pathname"` of the object, an then the `"type"`.

`Column 1:       nickname`

The first item in each row contains the alias or nickname of the object that will be used in FAST-MAP to refer to the object defined. The nickname must (1) be unique for every object defined in the "nicknames" file, and (2) contain only non-reserved characters.

`Column 2:       pathname`

The second item contains the complete pathname of the object. If the object is a directory (object type "gel" or "study"), it must be a pathname pointing to a *directory*. in other words, on UNIX, the pathname must end with a "/", whereas on a Macintosh, it must end with a ":". If the objet is a file (object type "matrix" or "sample"), it must be a complete pathname pointing to a *file* (that is, ending with the particular file's name).

If the original pathname contains SPACE characters, replace each SPACE with the special character defined as `"Space_char"` in "preferences".

`Column 3:       type`

The third item specifies the type of the object. There are four types supported in FAST-MAP: "gel", "study", "matrix", and "sample". Enter "gel" in the third column if the object defined is a gel directory, "study" if it is a study directory. Enter "matrix" if the object is an ABI matrix file, and "sample" if the object is an ABI sample file. Note that the pathname of a gel or study nickname points to a *directory*, whereas the pathname of a matrix or a sample file nickname points to a *file*.

The "type" column is *optional* in the sense that if it is not specified, it will inherit the "type" value from a previous row (if no previous row exists, it defaults to a "gel"). Take advantage of this feature by organizing the "nicknames" file by grouping objects of the same types together, as shown above. Otherwise, specify the "type" for each object in "nicknames" to avoid any confusion.

B.1.3. dyes

The "dyes" user file specifies synonyms used for each fluorescent dye. Currently, FAST-MAP supports four classes of fluorescent dye with the names: "blue", "green", "yellow", and "red". Despite the chromatic nature of the dye names, they actually define the order of the dye planes in which they were scanned in the DNA sequencer during data generation:

"blue"	↔	first dye plane
"green"	↔	second dye plane
"yellow"	↔	third dye plane
"red"	↔	fourth dye plane

These chromatic labels of the four dye classes are chosen merely to coincide with the current conventional color scheme for graphical displays. FAST-MAP can be easily extended to handle more than four dyes in future by defining additional classes in the "dyes" user file.

Here is a default "dyes" user file:

```
dye name        aliases
blue            fam
green           tet
yellow          hex
red             tam tamra rox
```

The synonymous dye names can then be used interchangeably in the "dye" fields of user files such as "markers" and "size_stds". For example, with the above "dyes" definitions, we can use any of the labels "tam", "tamra", "rox", or "red" to define a size standard that is run on the fourth dye plane.

Detailed descriptions

Here are the detailed descriptions of the "dyes" user file:

`dye_name`   `aliases`

The header tells FAST-MAP that the "dyes" user file contains two columns. Each row describes a particular class of dyes, followed by a list of one or more common dyes that belong to that class.

Column 1:   dye_name

There are currently only four dye classes defined in FAST-MAP: "blue", "green", "yellow", and "red". Technically, they define the first, second, third, and fourth dye planes of the DNA sequencer. The chromatic nature of the names merely reflects the conventional color scheme for visualizing the gel images. When new fluorescent dye technology emerges, we will extend the `"dye_name"` column to contain as many new dye classes as necessary.

Column 2:   aliases

For each `"dye_name"`, the user can define the synonymous fluorescent dyes under `"aliases"`. For example, the "tamra" dye and the "rox" dye may both be defined as aliases to the "red" dye class.

B.1.4. size_stds

The "size_stds" user file specifies the size standards. Each size standard is defined by the fluorescent dye used and the sizes (in base pairs) it contains. Each size standard's definition must fit one a single line. Here's a typical "size_stds" file:

```
name   dye   sizes

% Genescan 500
%
GS500  TAM   35 50 75 100 139 150 160 200 250 300 340 350 ... 490 500

% Bioventure's 20-bp ladder
%
BVMap  TAM   70 80 90 100 120 140 160 180 190 200 220 240 ... 380 400
```

A new name must be given (for example, with a version number appended to the original name) when a size standard has been changed (for example, by inserting or deleting a reference size, or by using a new fluorescent dye). However, there is no need to define a new size standard in "size_stds" for cases where a gel is incompletely imaged such that not all of the sizes in the size standard appear on that gel. In these situations, the user can define the actual range of the sizes by setting the `"Min_size_standard"` and `"Max_size_standard"` parameters in the gel's "settings" file.

Detailed descriptions

Here is a detailed description of the "size_stds" file:

<u>`name`</u>   <u>`dye`</u>   <u>`sizes`</u>

The "size_stds" file contains three columns as defined in the above headers. A size standard is specified by both the "dye" it is run with, and the "sizes" it contains. If either the "dye" field or the "sizes" field has been changed, the size standard must be defined under a new "name".

`Column 1:        name`

The first item is the name of the size standard being defined. As with all names in FAST-MAP, it should contain no "%" or white-space characters.

`Column 2:        dye`

The fluorescent dye used for the size standard. This dye name must be defined in the "dyes" user file.

`Column 3:        sizes`

The list of sizes (in base pairs) of the size standards. They must be ON THE SAME LINE and delimited by spaces or tabs.

B.1.5. markers

The "markers" file is where the user defines the characteristics of the markers used. The definition of a marker consists of:

- name: the name of the microsatellite marker;
- min_size: the size (in base pairs) of the smallest known allele for the marker;
- max_size: the size (in base pairs) of the largest known allele for the marker;
- dye: the fluorescent dye used to tag the marker;
- repeat_size: the size of the repetitive DNA unit in the microsatellite;
- plusA_handling: special handling (if any) which suppresses or enhances the plusA artifact;
- ladder: whether the peak sizes of the marker's alleles usually fall on a regularly spaced allelic ladder.

Here's a typical "markers" file:

```
name     min     max     dye     repeat    plusA        ladder

% Markers for "panel1"
%
p1m1     103     161     FAM     2         enhance      yes
p1m2     316     366     FAM     2         enhance      yes
p1m3     176     224     HEX     2         enhance      yes
p1m4     290     326     HEX     2         enhance      yes
p1m5     178     238     TET     2         enhance      yes
```

Note that if a marker characteristic (e.g. primer, buffer, enzyme) changes, the PCR stutter pattern typically changes as well. The user must make a new entry of the marker in the "markers" file and use a different marker name to help FAST-MAP distinguish it from any previous versions of the same marker locus. We suggest using a version number naming scheme: for the first version, the user names the marker without a version number (e.g., "p1m1"). For later versions, the user appends a version identifier (e.g., "p1m1_2", or "p1m1b").

Detailed descriptions

Here's a detailed description of the "markers" file:

`name` `min` `max` `dye` `repeat` `plusA` `ladder`

The "markers" file contains seven columns as defined by the above headers. Altogether, they form the requisite marker information used by FAST-MAP in genotyping.

`Column 1:      name`

The first item is the name of the marker being defined. As mentioned before, the user must make a new entry under a different marker name if any characteristics of a marker has been altered.

`Column 2:      min`

The second item is the size (in base pairs) of the smallest allele thus far detected for the microsatellite marker.

`Column 3:      max`

The third item is the size (in base pairs) of the largest allele thus far detected for the microsatellite marker.

`Column 4:      dye`

The fourth item specifies the fluorescent dye used to tag the microsatellite. The dye name should be defined in the "dyes" user file.

`Column 5:      repeat`

The fifth item specifies the size of the repetitive DNA unit (e.g., "1", "2", "3", or "4") in the microsatellite. For example, "2" indicates a di-nucleotide repeat.

`Column 6:      plusA`

`(none)`	No special handling.
`(enhance)`	PlusA was enhanced.
`(suppress)`	PlusA was suppressed.

```
(default)    none
```

The sixth item specifies how the plusA artifact for the marker has been handled experimentally. If there are two occurrences of the same marker with different plusA handling types, then they should be tagged with different version identifiers, to distinguish them from one another.

```
Column 7:         ladder
```

`(yes)`	The peak sizes of the marker's alleles fall on an evenly spaced allelic ladder.
`(no)`	The peak sizes of the marker's alleles do not fall on an evenly spaced allelic ladder. That is, odd-even alleles may be common for this marker.
`(default)`	yes

This final item specifies if an evenly-spaced allelic ladder is expected of the marker. That is, whether the genotypes of the marker contains only alleles from an evenly-spaced ladder.

Even though FAST-MAP will automatically use the appropriate default values or inherit from a previous marker if a column is left unspecified, we strongly advise the user to always fill in every column for each marker to avoid any confusion.

B.1.6. panels

The "panels" user file contains the definitions of the marker panels. A panel is defined as a list of markers. Here's a typical "panels" file:

```
name     markers
panel1   p1m1 p1m2 p1m3 p1m4 p1m5
panel1a  p1m1 p1m2
```

Detailed descriptions

And here is a detailed description of the "panels" file:

```
name                    markers
```

The "panels" file's header, as shown above, indicates that each panel entry must contain two fields: the name of the panel, followed by a list of marker names which has been defined in the "markers" file.

```
Column 1:       name
```

The first item is the name of the genetic marker panel being defined. As with all names in FAST-MAP, it should contain no "%" or white-space characters.

```
Column 2:       markers
```

A list of marker names delimited by spaces or tabs ON THE SAME LINE. Each of the marker names must be defined in the "markers" file.

B.1.7. pedigrees

The "pedigrees" file is a header text file which has a similar format with LINKAGE pedigree files. In this file, the user specifies the relationships of individuals (samples) whose DNA's are being genotyped. For example:

```
pedigree   sample    father     mother     sex

Ped01      Ped01P1   0          0          1
Ped01      Ped01P2   0          0          2
Ped01      Ped01C1   Ped01P1    Ped01P2    0
Ped01      Ped01C2   Ped01P1    Ped01P2    0

Ped02      Ped02P1   0          0          1
Ped02      Ped02P2   0          0          2
Ped02      Ped02C3   Ped02P1    Ped02P2    0
Ped02      Ped02C4   Ped02P1    Ped02P2    0
```

In the current version, there is only one "pedigrees" file. This means that the pedigree information of <u>all</u> the individuals in <u>all</u> the gels being analyzed must be specified in a single file. The pedigrees are only used in the viewing program *allele_view* to display the genotypes of related individuals together. Therefore, instead of keeping all the pedigrees together in the "pedigrees" file, it may be better to only import the relevant pedigrees into the "pedigrees" file when viewing their genotypes. We will provide a better organization of the pedigrees files in a future version.

Also, each individual can belong to only one pedigree at one time.

Detailed descriptions

Here is a detailed description of the "pedigrees" file:

```
pedigree      sample      father      mother      sex
```

The header of the "pedigrees" file, as shown above, indicates that each individual must be defined on a single line by five attributes: the name of the pedigree the individual belongs to, the sample ID of the individual, the IDs of the two parents, and the sex of the individual. In other words, "pedigrees" is a file containing five columns.

`Column 1:        pedigree`

The first item is the name of the pedigree to which the individual belongs.

`Column 2:        sample`

The second item is the sample ID used in labeling the DNA from the individual.

`Column 3:        father`

`(default)     0`

The third item is the sample ID of the father. If the father is unknown, enter "0".

```
Column 4:       mother

   (default)    0
```

The fourth item is the sample ID of the mother. If the mother is not known, enter "0".

```
Column 5:       sex

   (1)          Male.
   (2)          Female.
   (0)          Unknown.
   (default)    0
```

This final item specifies the gender of the sample. If it is not known, enter "0".

For users who do not already have pedigrees files in this format, we provide an utility function *add_pedigrees* to help the user annotate the "pedigrees" file from the sample names provided in the "layout" files automatically.

B.2. Specific information: Input files

The *input files* are so called because they reside in the "<gel>/input/" and "<study>/input/" directories. The input files contain information that are specific to a particular <gel> or <study>, unlike the user files which are globally accessible by all the gels and studies in the system. It is therefore necessary to create and annotate the input files for every new gel or study.

There are only two input files for both gels and studies:

- In the "settings" file, the user (1) provides gel or study-specific information, (2) controls the operations of the different FAST-MAP programs on the gel or study. Attributes for (2) appear in both the "settings" file for a gel and the "settings" file for a study. The user specify in the study's "settings" file whether to use these values from the study's "settings" file, or to use each individual gel's settings instead.
- The "layout" file of a gel contains information about the marker multiplexing on that gel, while the "layout" of a study simply lists the names of the gels in the study.

B.2.1. settings (gel)

The "settings" file for a gel has two functions: (1) it lets the user annotate specific details (e.g. gel file name, DNA-sequencer type) about the gel itself, and (2) it lets the user control the operation of the different FAST-MAP programs.

For convenience, we divide the "settings" file accordingly:

- `Section 1: Gel-specific settings.`
  This is the only section that the user MUST go through and fill in the appropriate values (where applicable) when setting up a gel;

- `Section 2: Program animation settings.`
  This section contains settings to control the messages and animation of program execution;

- `Section 3: Allele_call settings.`
  This section contains settings related to the program *allele_call* ;

- `Section 4: Allele_results settings.`
  This section contains user preferences for generating the results file. Since *allele_view* displays the genotypes in the results file, these *allele_results* settings also affect the *allele_view* indirectly;

- `Section 5: Allele_view settings.`
  This section, together with the section for *allele_reults*, contain settings to customize the display in *allele_view*;

- `Section 6: Marker_view settings.`
  This section allows the user to customize the display in *marker_view*;

- `Section 7: Allele_printout settings.`
  This section is for controlling the layout of the printouts generated by the program *allele_printout*.

When setting up a new gel, it is important to create the "settings" file first (using the command *edit*), and then followed by the "layout" file. This is because the "layout" file is created based on information such as the sample file name, the panel name, and the size standard name provided in Section 1 of the gel's "settings" file.

Here is a typical "settings" file for a gel:

```
attribute                   value

% Section 1. Gel-specific settings (Please fill in this section)
%
Gel_file_name               R211a.gel
Matrix_file_name            MATR211
Sample_file_name            R211a.samples
Sequencer_type              ABI
Number_of_lanes             34
Size_standard               BVMap
Min_size_standard           70
Max_size_standard           400
Panel_name                  panel1
Experiment_condition
Noise_threshold             50

% Section 2. Program animation settings
%
Verbose_mode_on             yes
Show_plots                  no

% Section 3. Allele_call settings
%
Redo_import_planes          no
Redo_manifold               no
Redo_quantitation           no
Redo_allele_calling         no
Analyze_noisy_data          yes

% Section 4. Allele_results settings
%
Output_noise_genotypes      no
Output_sort_by              markers
Latest_sample_only          no
Round_evenly_spaced         no

% Section 5. Allele_view settings
%
Prioritize_results          worst_first

% Section 6. Marker_view settings
%
Lanes_per_view              5

% Section 7. Allele_printout settings
%
Send_to_printer             yes
```

```
Show_figure                no
Rows_per_page              4
Columns_per_page           2
Include_electro_plots      no
```

Detailed descriptions

Here are the detailed descriptions of the attributes in a gel's "settings" file:

<u>attribute</u> <u>value</u>

The header indicates that each subsequent line in "settings" contains two entries: the attribute name describing the property to be set, and the value of that property.

`% Section 1: Gel-specific settings`

`Gel_file_name             R211a.gel`

The name of the gel file or collection file residing in the "input/" directory of a gel directory. There is no need to specify the complete path as the gel file must reside in "<gel>/input/" directory. Remember to replace each SPACE with the special "`Space_char`" defined in the "preferences" file if there are any SPACE characters in the file-name.

`Matrix_file_name          R211_MATRIX`

If the gel data is in the form of any ABI/377 or ABI/XL data file, or an ABI/373 "collection" file, the corresponding matrix file <u>must</u> be specified as the value of this attribute. Any of the following can be a valid value:

- a full path-name,
- a file-name in the gel input directory, or
- a valid nickname specified as a shared matrix file in "nicknames".

`Sample_file_name          R211a.samples`

If an ABI sample file containing the sample names and the gel layout information is available, FAST-MAP can create a "layout" file by reading from this sample file. This saves the user from having to type in all the sample names in "layout". Like the matrix file, any of the following can be a valid value:

- a full path-name,
- a file-name in the gel input directory, or
- a valid nickname specified as a shared sample file in "nicknames".

```
Sequencer_type              ABI
```

`(ABI)`	Any ABI sequencer. FAST-MAP can determine the machine model (e.g. ABI/373, ABI/377, or ABI/XL) automatically.
`(ALF)`	Pharmacia sequencer.
`(default)`	`ABI`

The type of DNA sequencing machine used in generating the gel data file.

```
Number_of_lanes             34
```

The total number of lanes on the gel. Some or all of these lanes may be loaded with size standards and DNA samples. The user will specify the actual loading pattern in the companion "layout" file.

```
Size_standard               BVMap
```

This attribute is used to help FAST-MAP in creating the "layout" file. The value must be a valid size standard defined in the "size_stds" file.

Since the sole purpose of this attribute is to help FAST-MAP set up the "layout" file, FAST-MAP refers to the "layout" file for the name of the size standard that has been run on the gel. Therefore, if the "layout" contains a different size standard name from the one in the "settings", the name in the "layout" is used.

```
Min_size_standard           70
```

The actual range of the size standards imaged may not contain all the sizes from the size standard specified. At the lower bp region, the primer peaks may interfere with peaks from the smaller sizes of the size standard. To get past these primer peaks, the attribute `Min_size_standard` allows the user to specify the smallest size standard visible across all loaded lanes on the gel. FAST-MAP's grid construction is able to handle incorrect specifications of this field if they are off by one or two rows. However, when in doubt, the user should leave this value blank first, and then use *prep_call* followed by prep_view to look at the gel image to ascertain the smallest visible size standard bands. While in *prep_view*, the user can further help FAST-MAP by cropping away the primer regions (that is, setting the "Start Scan").

If left unspecified, FAST-MAP will assume that the `Min_size_standard` is the smallest size in the size standard as defined in the user file "size_stds".

```
Max_size_standard          400
```

Similarly, the gel may be incompletely imaged such that the bands from the larger sizes are not captured on the gel. The attribute `Max_size_standard` allows the user to specify the largest size standard visible across all loaded lanes on the gel. If left unspecified, FAST-MAP will assume that the `Max_size_standard` is the largest size in the size standard as defined in the user file "size_stds".

```
Panel_name                 panel1
```

The `Panel_name` attribute is also used to help FAST-MAP create the default "layout" file. The value must be a valid maker panel defined in the "panels" file. As before, if the panel names in the "layout" file differ from that specified in "settings", FAST-MAP uses the values in the "layout".

```
Experiment_condition
```

The user can use any character string to describe the experimental conditions. This string denotes certain experimental conditions that can vary between gels when using the same panel. The reason for distinguishing experimental conditions is that certain data features (e.g., size binning) of a marker may vary slightly under different conditions (e.g., machine type, primers, gel temperature, etc.). For example, suppose that both GelA and

GelB used the same panel and size standard, but GelA was run on an ABI/373 and GelB on an ABI/377. To distinguish between the two, enter a different string inside the "`Experiment_condition`" slot for the gels. An empty (or absent) string is a valid value, and can be used initially.

```
Noise_threshold          50
```

This attribute allows the user to set the maximum signal intensity caused by background noise in the electropherograms. For ABI sequencers, the typical value is a number between 50 (default) to 100.

```
% Section 2: Program animation settings
```

```
Verbose_mode_on          yes
```

`(yes)`	Display informative text output in the main Matlab window during program executions.
`(no)`	Execute programs in silence, but display warnings and errors.
`(default)`	`yes`

```
Show_plots               no
```

`(yes)`	Display graphical animation during *allele_call* and *image_call* (may slow down computation considerably).
`(no)`	Do not display graphical animation.
`(default)`	`no`

```
% Section 3: Allele_call settings
```

```
Redo_import_planes       no
```

`(yes)`	Re-extract gel images in *allele_call* even if extracted images already exist.
`(no)`	Do not re-extract gel images in *allele_call*.

(default) no

Note that calling *prep_call* directly will <u>always</u> re-extract the gel images from the gel data file even if `Redo_import_planes` has been set to "`no`".

`Redo_manifold` `no`

(yes)	Always re-build the size-grid in *allele_call*.
(no)	Do not re-build in *allele_call* the sizing grid if one already exists for that gel.
(default)	no

Note that in the following cases, FAST-MAP will <u>always</u> re-build the manifold even if `Redo_manifold` has been set to "`no`":

- calling *image_call* directly on the gel;
- the gel images have been re-extracted from the gel data file; or
- the user has modified the manifold using *image_view*.

`Redo_quantitation` `no`

(yes)	Ignore any quantitation results from a previous run, re-quantitate the data for all experiments.
(no)	Use the quantitation results from a previous run if they already exists.
(default)	no

Note that if a new manifold has been constructed, FAST-MAP will <u>always</u> re-quantitate the data even if `Redo_quantitation` has been set to "`no`".

`Redo_allele_calling` `no`

(yes)	Performs allele_calling again on all experiments.
(no)	Do not genotype again those experiments that have already been genotyped.

(default)   no

If an experiment has been re-quantitated (in the various ways as described above), FAST-MAP will always re-genotype the data even if `Redo_allele_calling` has been set to "`no`".

`Analyse_noisy_data        yes`

(yes)	Do attempt to genotype experiments containing only data which are deemed to be noisy.
(no)	Do not genotype experiments containing only noise.
(default)	yes

In FAST-MAP, those experiments that are considered to contain noise only are assigned a quality measure of the value 0.

`% Section 4: Allele_results settings`

`Output_noise_genotypes    no`

(yes)	In the results file, output all computer genotypes even if they have 0 quality (i.e. noise).
(no)	Do not output, as the final allele calls, those genotypes of which FAST-MAP has deemed as noise. Instead, call these noise genotypes (0,0).
(default)	no

This attribute also affects the display of called alleles in the main window of *allele_view*. The third panel of the main *allele_view* window displays the final allele calls as output in the results file. For noise genotypes, if `Output_noise_genotypes` has been set to "`no`", a "NOISE ONLY" label is displayed in the main window instead of the attempted computer calls on the noise genotypes. If `Output_noise_genotypes` has been set to "`yes`", the noise genotypes will be labeled "noise", but the attempted computer allele calls will also be displayed in the window.

`Output_sort_by` `markers`

`(markers)`	In the results file, output the allele calls sorted by markers.
`(samples)`	Output the allele calls sorted by samples.
`(default)`	`markers`

`Latest_sample_only` `no`

`(yes)`	In the results file, output the latest allele calls of the sample if there are multiple copies. Within a study, the gels are ordered chronologically according to their order of appearance in the study's "layout"; within a gel, the genotypes are ordered chronologically according to their lane numbers.
`(no)`	Output all the allele calls, including any duplicate samples.
`(default)`	`no`

`Round_evenly_spaced` `no`

`(yes)`	Round all the alleles so that they fall on a strictly evenly-spaced ladder. This is useful with low resolution size calibration and microsatellites with a long repeat unit, for example, tri-nucleotides and tetra-nucleotides.
`(no)`	Do not adjust the alleles.
`(default)`	`no`

`%` <u>`Section 5`</u>`: Allele_view settings`

`Prioritize_results` `worst_first`

`(worst_first)`	In the "Prioritized" mode, order the experiments in increasing genotyping quality (i.e. view the bad genotypes first).
`(best_first)`	In the "Prioritized" mode, order the experiments in decreasing genotyping quality (i.e. view the good genotypes first).
`(default)`	`worst_first`

`% Section 6: Marker_view settings`

`Lanes_per_view            5`

Number of lanes to be viewed simultaneously in a single *marker_view* window.

`% Section 7: Allele_printout settings`

`Send_to_printer           yes`

`(yes)`	The plots are printed immediately on the printer specified in "preferences"
`(no)`	Plots are not sent to the printer. Instead, they are saved as a post-script file to be printed by the user later (see "allele_printout" for further details).
`(default)`	`yes`

`Show_figure               no`

`(yes)`	The plots are displayed on the screen while they are being saved or printed.
`(no)`	Plots are not displayed while they are being compiled as hardcopy.
`(default)`	`no`

`Rows_per_page             4`

Several plots can be printed on one page by arranging them in multiple rows and columns. The value of this setting sets the number of rows on a single page. The default value is 4.

`Columns_per_page          2`

Multiple plots can be printed on one row by setting this slot to a value greater than 1. The default value is 2. (The default total number of plots is 8 per page.)

`Include_electro_plots     no`

`(yes)`	Include electropherograms in the printouts.
`(no)`	Do not include electropherograms in the printouts.
`(default)`	`no`

By default, only the fitted peak profiles (along with called genotypes highlighted as filled peaks) are saved inside a hard-copy. This slot allows the long-range electropherogram trace (which spans the entire allele window) to be included inside the hard-copy.

B.2.2. layout (gel)

In the gel's "layout" file, the user lays out for the computer the marker multiplexing in a simple way. There are four header entries:

- lane, the lane number on the gel. The lane should still be specified even if it has not been loaded. In other words, the layout must consist of lanes 1 through "`Number_of_lanes`" as specified in "settings".
- sample, an identifier for the DNA sample tested.
- panel, the marker panel used in this lane for PCR amplifying the sample.
- size_std, the molecular weight size standards used in the lane. In the current software version, the identical size standards must be used in every loaded lane.

If an ABI sample file is supplied, FAST-MAP can automatically generate a "layout" file from this sample file together with the panel and size standard information supplied in the gel's "settings" file.

The following example is a partial listing of a gel's "layout" file. The actual file has 34 lanes specified in this way for a marker panel (panel1) and the Bioventures molecular weight markers (BVMap) loaded in lanes 1 through 32.

```
lane      sample     panel     size_std
1         ped01P1    panel1    BVMap
2         ped01P2    panel1    BVMap
3         ped01C1    panel1    BVMap
4         ped01C2    panel1    BVMap
```

5	ped01C3	panel1	BVMap
6	ped01C4	panel1	BVMap
7	ped01C5	panel1	BVMap
8	ped02P1	panel1	BVMap
9	ped02P2	panel1	BVMap
10	ped02C1	panel1	BVMap
...			
32	ped05C4	panel1	BVMap
33	blank	blank	blank
34	blank	blank	blank

Since there is often only one marker panel or size standard set used on a given gel, we provide a simpler way to specify layouts for these gels. The rule is that the panel or size standard needs to be specified only the first time it is used. After the first occurrence, the panel or size_std is assumed to be unchanged. For example:

lane	sample	panel	size_std
1	ped01P1	panel1	BVMap
2	ped01P2		
3	ped01C1		
4	ped01C2		
5	ped01C3		
6	ped01C4		
7	ped01C5		
8	ped02P1		
9	ped02P2		
10	ped02C1		
...			
32	ped05C4		
33	blank	blank	blank
34	blank		

When multiple panels are used, a similar rule applies. For example, if panel1 is used for lanes 1-7, and panel2 is used for lanes 8-10, the following layout file can be used:

lane	sample	panel	size_std
1	ped01P1	panel1	BVMap
2	ped01P2		
3	ped01C1		
4	ped01C2		
5	ped01C3		
6	ped01C4		
7	ped01C5		
8	ped02P1	panel2	
9	ped02P2		
10	ped02C1		
...			

It is important to remember that FAST-MAP is an *expectation-based* system. It relies unquestioningly on the loading patterns specified by the user in the "layout" file. To help FAST-MAP, the user must enter the correct gel loading pattern in "layout". If necessary, the user can use the program *prep_view* to quickly review the extracted gel images and verify the correct layout before allowing FAST-MAP to proceed with lane tracking (which depends on the "size_stds" column in "layout"), MW size calibration (which depends on the "Min_size_standard" and "Max_size_standard" attributes in "settings").

Detailed descriptions

Here's a detailed description of a gel's "layout" file:

`lane        sample       panel       size_std`

The "layout" file contains four columns as defined by the above header. The user specifies the loading pattern by telling the computer which of the lanes are loaded with DNA samples and which of the lanes are loaded with molecular weight size standards. With this simple specification scheme, we can easily specify complex loading patterns such as (1) different panels being loaded in different lanes on the same gel, and (2) DNA samples being loaded in alternate lanes from those loaded with the molecular weight standards (as typical on a single-dye system).

`Column 1:       lane`

This column contains the lane numbers, which must be from 1 to the value of "`Number_of_lanes`" as specified in "settings".

`Column 2:       sample`

The second column contains the sample names. As with any names in FAST-MAP, the sample names should not contain any SPACE or "%" characters. To use the "View Family" functionality in *allele_view*, these sample names should also be defined in the "pedigrees" file.

If a lane is unloaded, use the keyword "blank" for the sample name.

```
Column 3:       panel
```

The third column contains the names of the marker panels used for PCR amplifying the respective samples in the lanes. The marker panels need not be the same, but they must be defined in the "panels" file. If a lane is unloaded, use the keyword "blank".

```
Column 4:       size_std
```

The last column contains the names of the molecular weight standards used in the lanes. Use the keyword "blank" for unloaded lanes. Although this specification scheme allows the flexibility of using different size standards on the same gel, FAST-MAP's automatic size calibration program requires that each gel uses only one size standard.

B.2.3. settings (study)

The "settings" file for a study is the same as the "settings" file for a gel except for the gel-specific section (Section 1). Instead, this section is replaced by a section which allows the user to specify whether to use the values in the study's "settings" to process all its component gels or to use each individual gel's own "settings" instead.

Here is a typical "settings" for a study :

```
attribute                  value

% Section 1. Study-specific settings (Please fill in this section)
%
Use_study_settings         yes


% Section 2. Program animation settings
%
Verbose_mode_on            yes
Show_plots                 no


% Section 3. Allele_call settings
%
Redo_import_planes         no
Redo_manifold              no
Redo_quantitation          no
Redo_allele_calling        no
```

```
Analyze_noisy_data          yes


% Section 4. Allele_results settings
%
Output_noise_genotypes      no
Output_sort_by              markers
Latest_sample_only          no


% Section 5. Allele_view settings
%
Prioritize_results          worst_first


% Section 6. Marker_view settings
%
Lanes_per_view              5


% Section 7. Allele_printout settings
%
Send_to_printer             yes
Show_figure                 no
Rows_per_page               4
Columns_per_page            2
Include_electro_plots       no
```

Detailed descriptions

We describe here only the study-specific section (for the other sections of "settings", see "settings (gel)"):

```
% Section 1: Study-specific settings

Use_study_settings        yes
```

(yes)	Use the values in the study's "settings" to process each component gels in the study.
(no)	Use the gels' own "settings" instead.
(default)	yes

B.2.4 layout (study)

In FAST-MAP, a study is defined as a set of gels. Therefore, the "layout" of a study is a list of nicknames of the component gels in the study. The order of these nicknames defines the chronological order of the gels -- newer gels should be appended to the end of the gel list.

Here's a typical study "layout" file:

```
gel
%
% Gels for panel1 study
%
GEL1     % 4/15/97, Ped01-Ped06
GEL2     % 4/16/97, Ped06-Ped11
GEL3     % 4/18/97, Ped12-Ped17
```

Detailed descriptions

And here's a detailed description of the "layout" file for a study:

`gel`

The header of the "layout" file for a study indicates that the file is a single-column file consists of gel names.

`Column 1:       gel`

The nicknames of the study's component gels are entered one on each row. They must be defined under the type "gel" in the "nicknames" file. The order of these nicknames reflect the chronological order of the gels, with the older ones on top and the newer ones appended to the end of the list.

As with any user/input files in FAST-MAP, the user is free to add comments anywhere in the files as long as they are marked with the "%" character (as shown in the example "layout" file).

Appendix C: FAST-MAP's Libraries

For a system to be useful in processing real data in realtime, it must be efficient in its data processing as well as robust against the various nuances present in real data. In FAST-MAP, the major means for the requisite speed and robustness is intelligent expectation-based computation. For speed, FAST-MAP uses well-informed expectations to focus its attention directly on the relevant features of the input, instead of spending time on the less important features. For robustness, FAST-MAP uses its learned expectations to distinguish between data and random noise, thereby avoiding making mistakes due to data singularities.

The key, then, is to construct useful, realistic, and specific expectations about the data. For this, FAST-MAP relies on its knowledge acquired on similar data over time. Because the data knowledge is specific to the types of data being processed, we organize the acquired information in data structures called *libraries,* implemented in FAST-MAP as individual files that are compiled automatically. As FAST-MAP processes more data, it is able to augment the appropriate library files with more information and automatically becomes more knowledgeable over time.

There are four main classes of information learned by FAST-MAP:

- *binning information of size standards:* FAST-MAP uses previous relative pixel locations of MW bands to construct reliable expectations on where to rapidly search for MW bands on the current gel;
- *binning information for markers:* FAST-MAP uses relative pixel information on markers from previous gels to guide its search for marker bands on the current gel. The binning information is also used in assigning consistent integral allele labels to the marker bands;
- *stutter information for markers:* FAST-MAP uses previously observed stutter patterns to deconvolve for genotypes; and
- *relative amplification information for markers:* FAST-MAP uses the amplification ratios computed from previously analyzed marker data to adjust for relative amplification in the current stutter patterns .

In the following sections, we describe the different types of information that FAST-MAP stores in its libraries.

C.1. Binning libraries for size standards

DNA fragments do not migrate linearly on an electrophoretic gel (Southern, 1979). To robustly predict the whereabouts of MW bands on a gel without assuming any global functional forms for DNA migration, we use the locations of MW bands detected on previous gels that were ran under similar experimental conditions. For this purpose, FAST-MAP keeps a customized MW library file for every size standard and experimental condition used by a particular user.

To predict the locations of the MW bands on a gel, the corresponding size standard binning library must provide the relative pixel information for the MW standard. In FAST-MAP, we store the absolute pixel values from the previous gels since it is only a trivial computation to compute the relative pixels from the absolute pixel values (the absolute pixel values are useful for debugging the binning libraries). Thus, a FAST-MAP size standard binning library contains the following information:

- the sizes of the size standard DNA fragments;
- the actual absolute image pixels of each MW band detected previously on a gel lane.

Here is an example of a size standard (GS350) binning library:

MW (bp)	50	75	100	139	150	160	200	250	300	340	350
Gel1, L1	31	173	314	562	624	688	945	1253	1557	0	0
Gel1, L2	31	175	316	564	626	690	948	1258	1559	0	0
Gel2, L1	0	2660	2964	3490	3630	3775	4363	5116	5963	6622	6801
Gel2, L2	0	2597	2898	3423	3561	3706	4295	5049	5899	6558	6737
:	:	:	:	:	:	:	:	:	:	:	:

Each row in the library is labeled by the gel name (e.g. Gel1) and the lane number (e.g. L1), followed by the absolute pixel values for each of the MW size in the standard. Missing MW fragments are entered zeros as their pixel values. In the above example, Gel1 contains only size fragments of up to 300 bp, while Gel2 starts at 75 bp instead of 50 bp. Indeed, the range of MW bands that actually showed up on the gel image tend to vary from gel to gel: the smaller MW may have been mixed up with the primer signals, while the larger MW may not have been separated because of insufficient gel run time.

To predict the relative pixels of a new gel that has the size standard GS350 running, say, from 75 bp to 300 bp, we take the element-to-element column vector division of the nonzero columns in the binning library:

$$p_i = \ mean((c_i\text{-}c_{start})./(c_{end}\text{-}c_{start}))$$

where p_i denotes the predicted relative pixel of *i* bp, c_i denotes a nonzero library column for *i* bp, and in this case, *start* = 75 and *end* = 300. The operator "./" denotes element-to-element column vector division. The predicted relative pixels (based on the 4 rows shown) for the new gel are:

p_{75}	p_{100}	p_{139}	p_{150}	p_{160}	p_{200}	p_{250}	p_{300}
0	.0967	.2659	.3093	.3544	.5365	.7623	1

C.2. Binning libraries for markers

Like the size standard binning libraries, the marker binning libraries store relative pixel information about marker bands. FAST-MAP uses the relative pixel information acquired from previous gels to predict where to efficiently search for marker bands in the current electropherogram. Additionally, the corresponding binned sizes allows FAST-MAP to consistently assign integer allele labels to the detected marker bands.

Unlike the size standard binning libraries, we store the predicted relative pixels in the marker binning libraries since it is nontrivial to re-compute them on the fly using stutter crawling. In addition, we also store the training data within the libraries so that they can be incorporated with new data for re-training to facilitate incremental learning.

Here is a typical marker (D16S415) binning library in FAST-MAP:

Allele labels	194	195	196	197	198	199	…
Relative pixels	.008	.023	.038	.053	.068	.083	…
Repeat unit	2 bp						
Alleles binned so far	203	205	207	223	225	227	…
Allele ladder's start bp	195						
Gel1, L1 (band locations)	.474	.504	.535	.565			
Gel1, L1 (band heights)	.085	.239	.456	.290			
Gel1, L2 (band locations)	.231	.261	.291	.306	.323	.443	…
Gel1, L2 (band heights)	.025	.085	.361	.129	.999	.031	…
:	:	:	:	:	:	:	:

The binning library file can be divided into three main sections:

- The first section (rows 1 and 2) defines the allele bins with an assignment of the integral allele labels to the predicted relative pixel values;
- The next section provides more information about the marker, such as the expected length of a repeat unit for the marker, the alleles that FAST-MAP has actually binned so far, and the start base pair of the predicted allelic ladder to indicate the expected parity of the possible marker alleles (in this case, we expect a ladder of 195 bp, 197 bp, 199 bp, and so on for D16S415);
- The final section stores the binning training data that we have acquired so far. For each gel lane that was used for binning, we record the relative pixels of the stutter trail together with the stutter band's normalized heights. When stutter crawling, the heights of the bands are used as a quality and confidence measure about the stutter bands.

C.3. Genotyping libraries for markers

Automatic genotyping has been precluded (mainly) by two PCR artifacts: stuttering and relative amplification. These reproducible artifacts are recorded in FAST-MAP's genotyping libraries; they are then used to predict the expected stutter and amplification patterns for deconvolution. As the stutter and amplification patterns are highly specific, FAST-MAP maintains a detailed genotyping library for each marker under every experimental condition used by an user. Each of the genotyping libraries consists of two components: a stutter pattern matrix, and a relative amplification ratio table.

Stutter pattern matrix

Like the marker binning libraries, we store the training data together with the stutter pattern matrix A in order to facilitate *incremental* learning. The training set can be augmented with more data as they become available, allowing FAST-MAP to refine its stutter matrix over time.

The stutter pattern matrix A is stored as a two dimensional matrix indexed by integer allele labels, such as:

	114 bp	112 bp	110 bp	108 bp	106 bp
114 bp	1.000	0	0	0	0
112 bp	0.500	1.000	0	0	0
110 bp	0.250	0.600	1.000	0	0
108 bp	0.125	0.300	0.700	1.000	0
106 bp	0	0.150	0.350	0.800	1.000
104 bp	0	0	0.160	0.400	0.900
102 bp	0	0	0	0.200	0.450
100 bp	0	0	0	0	0.220

For the associated training data, we store the known genotypes, the amplification ratios, the quality measures, and the observed data vectors:

	genotype	ratio	qual	observed data vector							
Gel1, L1	112, 114	.992	.895	.246	.388	.219	.089	.034	.004	.019	.000
Gel1, L2	106, 114	1.23	.908	.219	.137	.042	.035	.234	.186	.081	.062
Gel2, L1	108, 112	1.02	.921	.017	.228	.145	.277	.187	.083	.064	.000
:	:	:	:	:	:	:	:	:	:	:	:

Relative amplification ratio table

For the relative amplification ratio tables, we generally store only the "compressed" versions because of insufficient initial data. In a "compressed" amplification ratio table ϑ, we assume that the relative amplification ratio depends *only* on the allele size difference, such that $\vartheta(a_1, a_2) = \vartheta(a_3, a_4)$ whenever $|a_1 - a_2| = |a_3 - a_4|$. As more data becomes available, we can expand ϑ into a full-blown relative amplification ratio table with one entry for every genotype. However, it has been our experience that the compressed ϑ is itself generally adequate for handling the relative amplification problem.

In FAST-MAP, the compressed relative amplification table is stored as a vector of triples

<minimum ratio, mean ratio, maximum ratio>

to indicate the range of possible ratios for each fixed size difference in a pair of alleles. For example:

allele difference	0 bp	2 bp	4 bp	6 bp	8 bp
minimum ratio	1	.952	.899	1.02	1.01
mean ratio	1	.994	1.09	1.13	1.17
maximum ratio	1	1.32	1.25	1.28	1.32

As with the stutter matrix, the amplification ratio table can also be incrementally refined by re-computing it using the training data stored in the corresponding stutter pattern matrix library.

Appendix D: FAST-MAP's Programs

FAST-MAP (Fluorescent Allele-calling Software Toolkit – Microsatellite Automation Package) is a software package that we have written for automated microsatellite genotyping. Since its public release[44] in May 1996, it has been used by geneticists in academic research laboratories, national research institutions, and commercial genotyping centers throughout the world. TrueAllele™, the successor to FAST-MAP, continues to improve upon the system as well as the technology[45].

As a practical genotyping system, FAST-MAP consists of a comprehensive assortment of programs for data analysis, visual display, and result outputs. In general, we can classify the programs in FAST-MAP in three categories: core programs, viewing programs, and utility. The core programs implement the various algorithms that we have discussed in this dissertation, providing the necessary computational power for fully automated microsatellite genotyping. The viewing programs provide interactive graphical interface with which the user can review, edit, and assist the computer at the various programmatic checkpoints provided in FAST-MAP. The utility programs provide useful functionalities such as setting up data for analysis and customizing the genotyping results generated by FAST-MAP.

The following sections contain abstracts from the FAST-MAP user manual that describe the various FAST-MAP programs in details. In the descriptions, we adopt the following notation:

- Italicized words refer to computer *programs*, and double-quoted words refer to computer "files" or directory "nicknames/";
- Underscores are used for <u>emphasis</u>;
- Brackets refer to a <variable> that is filled in with a specific value when it is actually used.

---

[44] `http://www.cs.cmu.edu/~genome/FAST-MAP.html.`
[45] `http://www.cybergenetics-inc.com.`

D.1. Overview

FAST-MAP is a <u>fully-automated</u> genotyping system. When analyzing clean gel data, the program *allele_call* will analyze the data directly from raw gel files and generate genotyping results without user assistance. In this mode, the user can leave the computer to analyze the gel data on its own, and then return to view the genotyping results using an interactive graphical interface *allele_view*. The sequence of programs to call in this fully-automated mode is therefore:

allele_call → *allele_view*

For the more cautious users, we provide numerous checkpoints during computation for the user to verify (and if necessary, edit) the intermediate results at various stages of the data analysis. For example, the user might want to first ensure that the lane tracking and size calibration are done properly before proceeding to calling the alleles based on the sizing grid. For this, the user can use the program *image_call* to perform lane tracking and size calibration on raw gel data, and then use the program *image_view* to graphically inspect the sizing grid constructed by the computer. If the computer was having problems with the gel, the user can assist it in *image_view* by providing more information about the gel. For example, if there was a high degree of gel smile on the gel, the user can tell the computer the actual shape of the grid in *image_view*. Or, if the computer misplaced a few size standards bands, the user can repair the grid by indicating where these bands should be. After an accurate sizing grid has been generated, the user can then invoke *allele_call* to genotype the size-calibrated data and then use *allele_view* to inspect the results. A typical sequence of programs to call in this "cautious" mode is:

image_call → *image_view* → *allele_call* → *allele_view*

Sometimes, when a new marker panel is used for the first time, the allele ranges covering all the genotypes of the population being studied may not be known beforehand. In this case, the user can use the program *marker_view* to graphically scans the markers' electropherograms (that has been size-calibrated with *image_call*) and then set the correct enclosing allele windows. A typical calling sequence for this case is:

image_call → *image_view* → *marker_view* → *allele_call* → *allele_view*

For the "extra cautious" user, before even proceeding with any computer analysis of the gel from the raw data files, the user might want to inspect the gel images to make sure that the information provided to FAST-MAP are accurate. In particular, the user may wish to view the pre-analysis gel image and do any of the following: (1) crop away any primer regions; (2) check the minimum and maximum size standards that are actually captured on the gel ; and (3) verify the number of loaded lanes. In this "extra cautious" mode, the user can use the program *prep_call* to extract the gel images from the machine-specific raw gel data files, and then use *prep_view* to view the gel images. Here's a typical calling sequence:

prep_call → *prep_view* → *image_call* → *image_view*
→ *marker_view* → *allele_call* → *allele_view*

In general, the graphical interface for users to interactively assist FAST-MAP in its analysis are named with a suffix *"_view"*. Although these interface are typically used in the sequences described above, they can also be used at any checkpoints as long as the requisite intermediate results have already been computed. For example, a user may run FAST-MAP in the fully-automated mode overnight, and return to inspect the genotypes. If the sizing of the alleles were found to b inaccurate, the user can then use *image_view* to inspect and edit the sizing grid, and then call *allele_call* to re-analyze the gel. An example of such a calling sequence is:

allele_call → *allele_view* → *image_view* → *allele_call*

D.2. Program list

As we have mentioned earlier, the programs in FAST-MAP can be classified in one of three categories: core programs, viewing programs, and utility programs. For a quick overview, we give a brief summary listing of the various programs in each of the categories here. The individual programs will be described in greater details later in this Appendix.

Core programs

prep_call This program prepares the raw data image files from various DNA sequencers by extracting the image data from the files into a sequencer-independent format. It is automatically called by FAST-MAP programs such as *image_call* and *allele_call*, but the user can also call it directly typing:

```
>> prep_call <nickname>
```

As usual, `<nickname>` can be for a single gel or a study. To process multiple gels or studies at a single command, simply enter all the nicknames:

```
>> prep_call <nickname1> <nickname2> ...
```

image_call This program performs automatic lane/size determination and electropherogram data extraction. It is automatically executed by the *allele_call* program, but the user can also invoke it by typing :

```
>> image_call <nickname>
```

where "`<nickname>`" can be for either a gel or a study. To process more than one gel or study in a single command, type

```
>> image_call <nickname1> <nickname2> ...
```

allele_call This workhorse program automatically calls the alleles. It can be configured on clean data to work <u>fully automatically</u>. It is invoked by typing

```
>> allele_call <nickname>
```

Both gels and studies can be analyzed with *allele_call*. To batch-process multiple gels and studies in a single command, simply type:

```
>> allele_call <nickname1> <nickname2> ...
```

Viewing programs

prep_view This companion program can be called after *prep_call* to view the extracted gel images. The user can crop away the primer peak region of the gel as well as ascertain the minimum and maximum size standards that actually appear on the gel. To view the extracted images of a gel or study, type:

```
>> prep_view <nickname>
```

As with all other graphical viewing programs in FAST-MAP, only one gel or study may be viewed with a single command of *prep_view*.

image_view This companion program is used to visually inspect and edit the lane/size grid generated by the program *image_call*. It is invoked by typing

```
>> image_view <nickname>
```

where "`<nickname>`" can be for either a gel or a study. Only a single gel or study can be viewed with a single *image_view* command.

marker_view This user-analysis program is used to visually inspect the allele window for a marker in a gel or study (after the electropherograms have been calibrated by the program *image_call*). This is particularly useful when the marker panel is being analyzed for the first time such that the exact range of alleles for the marker is not known with certainty. The user invokes "`marker_view`" by typing

```
>> marker_view <nickname>
```

Again, only a single gel or study can be inspected with a single command of *marker_view*.

allele_view This program is used to visually review the alleles. It is the main graphical interface to the *allele_call* program's results. To invoke it, type:

```
>> allele_view <nickname>
```

As in *allele_call*, the `<nickname>` can be for a single gel or a study comprising of multiple gels. However, only a single gel or study may be *allele_view*'ed at one time.

Utility programs

edit This program allows users to rapidly access and edit the data files without having to type in the full path of the files. For example, to edit the "nicknames" file, type:

```
>> edit nicknames
```

Or, to edit a particular gel or study's "settings" file, type:

```
>> edit <nickname> settings
```

edit will create default files for the user if the file does not already exist. To see a list of files accessible by *edit*, type:

```
>> help edit
```

inspect Like the program *edit*, the program *inspect* allows users to rapidly inspect the content of data files in Matlab without having to type in full pathnames. For example, to inspect the genotyping results of GEL1, type:

```
>> inspect GEL1 results
```

For a list of files accessible by *inspect*, type:

```
>> help inspect
```

setup This FAST-MAP script can be used for setting up a gel, a study, a shared matrix file, or a shared sample file. To use the *setup* script, the user must first transfer all relevant data files into the "FAST-MAP/incoming/" directory, and then invoke *setup* in FAST-MAP:

```
>> setup <object> [<nickname>]
```

The user will be asked a series of questions pertaining to the setting up of the particular FAST-MAP object. Four types of FAST-MAP "`<object>`"s can be set up in this way: a gel, a study, a shared matrix file, or a shared sample file. The optional `<nickname>` can be supplied to specify the nickname that is to be used for the object . Here are some examples:

```
>> setup gel GEL1
>> setup study STUDY1
>> setup matrix S1_MATRIX
>> setup sample S1_SAMPLE
```

Alternatively, the user may prefer to set up the data manually.

allele_results This result-generation program can be used to re-format the result text files that contain the allele calls. These text files can then be examined by users (e.g., in Excel) or by computers (e.g., by databases or other computer programs). The desired format of the result text files can be customize using the "settings" file of the gel or study (using the *edit* command). The user can specify to have the genotypes sorted by markers or by individuals. To generate the results files, type:

```
>> allele_results <nickname>
```

To process more than one gel or study at once, type:

```
>> allele_results <nickname1> <nickname2> ...
```

D.3. Core programs

D.3.1. prep_call

The program *prep_call* prepares the raw data image files generated by various DNA sequencers. It extracts the image data into a sequencer-independent format. This program must be called before using *prep_view*, if the gel has not been analyzed before. The program *allele_call* and *image_call* automatically execute *prep_call* when analyzing a new gel.

Program operation

To use *prep_call*, the user types

```
>> prep_call <nickname>
```

e.g.

```
>> prep_call STUDY1
```

Calling *prep_call* directly will <u>always</u> re-extract the image data. If an image data is re-extracted, previous genotyping results will be assumed obsolete. It will then be necessary to call allele_call and *image_call* to re-analyze the gel or study.

To *prep_call* more than one gel or study with a single command, simply type:

```
>> prep_call <nickname1> <nickname2> ...
```

e.g.

```
>> prep_call STUDY1 GEL3 STUDY2 STUDY3
```

Up to 45 gels and studies are allowed in a single batch.

Example run

To invoke *prep_call* from a gel or a study, type

```
>> prep_call <nickname>
```

For example, type

```
>> prep_call GEL3
```

And here's a computer trace of *prep_call*:

```
FAST-MAP v1.04b (created April 21, 1997, last revised 04/21/97,
system HP700).

[Session started at 19-Apr-97 13:40:16.]

GEL: GEL3
===

Clearing previously computed results: GEL3...Done.

GEL3: prep_call [19-Apr-97 13:40:17]
----

Image file R211c.gel is an ABI/377 collection file (version 2.00).
GEL3 is a 5540 x 194 gel with 4 dye planes.

Reading gel data from R211c.gel ..................................
................................................Done.
Adjusting image contrast........Done.

[Session for "prep_call GEL3" ended at 19-Apr-97 13:46:55.]
```

GEL3 is now ready to be inspected using the companion program *prep_view*.

D.3.2. image_call

This program is used to directly perform the automated lane/size tracking. It is useful when additional editing (in *image_view*) is required to build the two dimensional gel coordinates.

Program operation

To use *image_call,* the user types

```
>> image_call <nickname>
```

e.g.

```
>> image_call STUDY1
```

The *image_call* program will always perform the lane-tracking and size calibration on <nickname>. *image_call* should be followed by user interaction with the *image_view* program especially if the user has reason to suspect the gel contains noise, non-uniform amplification etc. which could lead to an inaccurate size grid.

To *image_call* more than one gel or study with a single command, simply type:

```
>> image_call <nickname1> <nickname2> ...
```

e.g.

```
>> image_call STUDY1 GEL3 STUDY2 STUDY3
```

Up to 45 gels and studies are allowed in a single batch.

Example run

First, make sure that the layout and the settings file correctly describes the size standards, its name, the maximum, and minimum sizes present on the gel. If desired, the user may invoke *prep_call* followed by *prep_view* to view the gel image before *image_call*.

Invoke *image_call*:

```
>> image_call GEL3
```

The computer responds with the following messages:

```
Clearing previously computed results: GEL3...Done.

GEL3: Image_call [19-Apr-97 15:57:48]
----
Reading image data for dye 4...Done.
```

Depending on whether the graphics option in "settings" is set to yes or no, the user may or may not see a window open up with the size standard plane image. This image gets overlaid by interpolated lanes, then size standard rows, as image_call continues with further status reports on its progress. The text output on the screen is also saved in the “image_call.log” file in the gel output directory. Here is the rest of the sample session:

```
Scanning for gel lanes.......___+++++.::::Done.
Scanning for size std bands................__________________+++.:::
:::::::::::::::::::::::::::Done.
Refining MW peaks .................................Done.
Computing pixel-to-bp calibration .........................Done.
Saving size calibration..................................Done.
Reading image data for dye 1...Done.
Converting image to profiles...............................Done.
Reading image data for dye 2...Done.
Converting image to profiles...............................Done.
Reading image data for dye 3...Done.
Converting image to profiles...............................Done.
Reading image data for dye 4...Done.
Converting image to profiles...............................Done.

[Session for "image_call GEL3" ended at 19-Apr-97 16:09:44.]
```

D.3.3. allele_call

The *allele_call* program is a fully automated computer program that genotype alleles from microsatellite gel data. When fully automated lane/size tracking is problematic, we recommend running the *image_call* and *image_call* programs prior to running *allele_call*.

Program operation

To run the *allele_call* program, the user types the command

```
>> allele_call <nickname>
```

The `<nickname>` can be for a single gel or for a study. In a study, FAST-MAP can make use of all the data on all the gels simultaneously. For example, consistency of the size calling across gels can be ensured by binning across all the gels in the study.

Therefore, instead of calling *allele_call* on each gel individually, we recommend that the user bundle related gels together as a study.

The *allele_call* program performs four main steps:

(1) Extracting image data. This step prepares *allele_call* by extracting the gel images from raw sequencer data files into a format that is sequencer-independent. If the user wishes to preview the gel image (using *prep_view*) before allele-calling the gel, this step can be performed prior to running *allele_call* by explicitly calling the *prep_call* program. Otherwise, it will be called automatically as part of *allele_call* .

(2) Tracking lanes and sizes. If desired, this step can also be performed prior to running *allele_call* by using the separate *image_call* program. Otherwise, it is called automatically as part of *allele_call* (which internally calls *image_call*). The *image_call* algorithms analyze all the gel's molecular weight sizing data simultaneously in two dimensions. The user provides (in "/<gel>/input/settings") estimates of the DNA size range that will appear on the gel, and the computer handles the rest. This tracking lets the analysis program move interchangeably between the observed 2D gel image and the expected (lane, base_pair) coordinates. The separate *image_view* program is a user interface for checking the computer's tracking results, and for performing other editing tasks.

(3) Quantitating DNA bands. For the allele deconvolution methods to work well, accurate estimates of DNA size and concentration for each band (including artifactual stutter bands) are needed. This *allele_call* module performs this quantitation.

(4) Allele determination. Once the data has been quantitated, the *allele_call* program determines the alleles by deconvolving the quantitative data relative to the marker's calibrated stutter artifact. This stutter calibration is learned by *allele_call* as it studies data on one or more gels; the more gels studied, the more accurate the allele calling. If the PCR conditions (buffer, enzyme, primers, thermocycling, etc.) change, then the learning has to start anew.

The *allele_call* program's four steps are invisible to the user: the program works fully automatically. However, the user can control much of the inside workings of the program by providing settings for parameters and options in the "settings" file (see "Settings").

To analyze multiple gels and studies with a single command, type

```
>> allele_call <nickname1> <nickname2> <nickname3> ...
```

Up to 45 gels and studies are allowed in a single batch.

The user can view the status of the batch processing by typing:

```
>> inspect <nickname1> status
```

And a status report will be displayed, such as the following:

```
ALLELE_CALL for 2 gels/studies started at 21-Apr-97 12:24:03.

STUDY1: study
        Started   :     21-Apr-97 12:24:03
        Completed :     21-Apr-97 13:48:34

GEL3: gel
        Started   :     21-Apr-97 13:48:34
        Completed :     21-Apr-97 14:57:24

ALLELE_CALL for 2 valid gels/studies completed successfully at 21-
Apr-97 14:57:24.
```

The user can also review the detailed traces of each nickname by typing:

```
>> inspect <nickname1> logfile
```

Example run

Before running *allele_call*, the user must (1) set up the gel or study (see "setup"), and (2) annotate the user files (see "Annotating Genotyping Data"). Then, in FAST-MAP, type:

```
>> allele_call <nickname>
```

This command will start running the genotyping program on the user's gel data according to the specifications that were provided in the user-annotation files. The program prints out line-by-line descriptions of what it is doing; each "." in the "..." messages indicates completion of a program substep. The user can largely ignore this computer monologue.

It is useful primarily for knowing that the program is still running, and may assist the software support group in helping the user out when there is a problem. It is also recorded in a "logfile" ("<nickname>/output/allele_call.log"), so the user can review it using the command

```
>> inspect <nickname> logfile
```

A typical run starts off as:

```
>> allele_call STUDY1


FAST-MAP v1.04b (created April 21, 1997, last revised 04/21/97,
system HP700).

[Session started at 21-Apr-97 12:24:05.]

STUDY: STUDY1
=====

Clearing previously computed results: GEL1...GEL2...Done.

Marker p1m1 : [21-Apr-97 12:24:10]
------

Scanning GEL1 for marker
windows.................................Done.
Quantitating GEL1 lane 1.....a.b..Done.
Genotyping GEL1 lane 1....Done.
Quantitating GEL1 lane 2.....a.b..Done.
Genotyping GEL1 lane 2....Done.
Quantitating GEL1 lane 3.....a.b..Done.
Genotyping GEL1 lane 3....Done.
Quantitating GEL1 lane 4.....a.b..Done.
Genotyping GEL1 lane 4....Done.
Quantitating GEL1 lane 5.....a.b..Done.
Genotyping GEL1 lane 5....Done.
Quantitating GEL1 lane 6.....a.b..Done.
Genotyping GEL1 lane 6....Done.
......
```

Depending on the "settings" parameters, the user may see:

- A "size grid" window for animating the lane-tracking and size calibration.
- A "quantitation window" for animating the DNA size and concentration determinations.

As the program runs, the following steps are done:

(1) lane/size tracking for the entire gel image;

(2) DNA size and concentration determination for all genotypes; and

(3) allele determination by deconvolution for each genotype.

At the end of the run, the computer signals that it has completed the requested genotyping services. A tabbed text file of the genotyping results is then stored in the "/<nickname>/output/" directory with the name:

"/<nickname>/output/<nickname>.results",

i.e., with ".results" appended to the gel or study nickname. In FAST-MAP, the user can view it using

```
>> inspect <nickname> results
```

gel	lane	sample	marker	allele1	allele2	comput1	comput2	qual	edit1	edit2
GEL1	1	Ped01P1	p1m1	139	141	139	141	0.9610		
GEL1	2	Ped01P2	p1m1	141	153	141	153	0.9473		
GEL1	3	Ped01C1	p1m1	141	153	141	153	0.9410		
GEL1	4	Ped01C2	p1m1	139	153	139	153	0.9434		
GEL1	5	Ped01C3	p1m1	139	141	139	141	0.9532		
GEL1	6	Ped01C4	p1m1	141	153	141	153	0.8545		
GEL1	7	Ped01C5	p1m1	141	153	141	153	0.9440		
GEL1	8	Ped01C6	p1m1	141	141	141	141	0.9336		
GEL1	9	Ped01C8	p1m1	139	141	139	141	0.8973		
GEL1	10	Ped02P1	p1m1	141	143	141	143	0.9491		
:	:	:	:	:	:	:	:	:		

Although the user can edit the results file using FAST-MAP's *edit* command or an external spreadsheet program such as Excel, the only way to let FAST-MAP know that a genotype has been edited by is to do it in allele_view. The new results file will include the user edits in the "edit1" and "edit2" columns, as well as in the final allele calls columns ("allele1" and "allele2").

On UNIX, the user can print out the results file to the printer specified in the "preferences" file by:

```
>> print_now <nickname> results
```

On Macintosh, the user can print using the "Print" command under the "File" menu directly.

D.4. Viewing programs

D.4.1. prep_view

This program can be used after *prep_call* to view gel images prior to *image_call* and *allele_call*. In *prep_view*, the user can verify the gel layout, check the minimum and maximum size standards that are captured on the gel, and crop away excess gel regions (e.g. the primer regions).

Program operation

To use *prep_view,* the gel images must first be extracted using *prep_call*. Then, the user types:

```
>> prep_view <nickname>
```

e.g.

```
>> prep_view STUDY1
```

In *prep_view*, the user can do three things:
(1) crop the image (e.g. to remove the primer region) by setting the start and/or end scan;
(2) view a lane to verify the smallest and largest molecular weight standards that were captured on the gel; and
(3) view a "row" to verify the loaded lanes.

Example run

To view the extracted gel images, type

```
>> prep_view <nickname>
```

For example,

```
>> prep_view GEL3
```

The computer responds by:

```
Initializing gel settings for GEL3.....Done.
Reading image data for dye 4...Done.
```

A "Prep View" window is opened, displaying the size plane of the current gel.

Prep View window

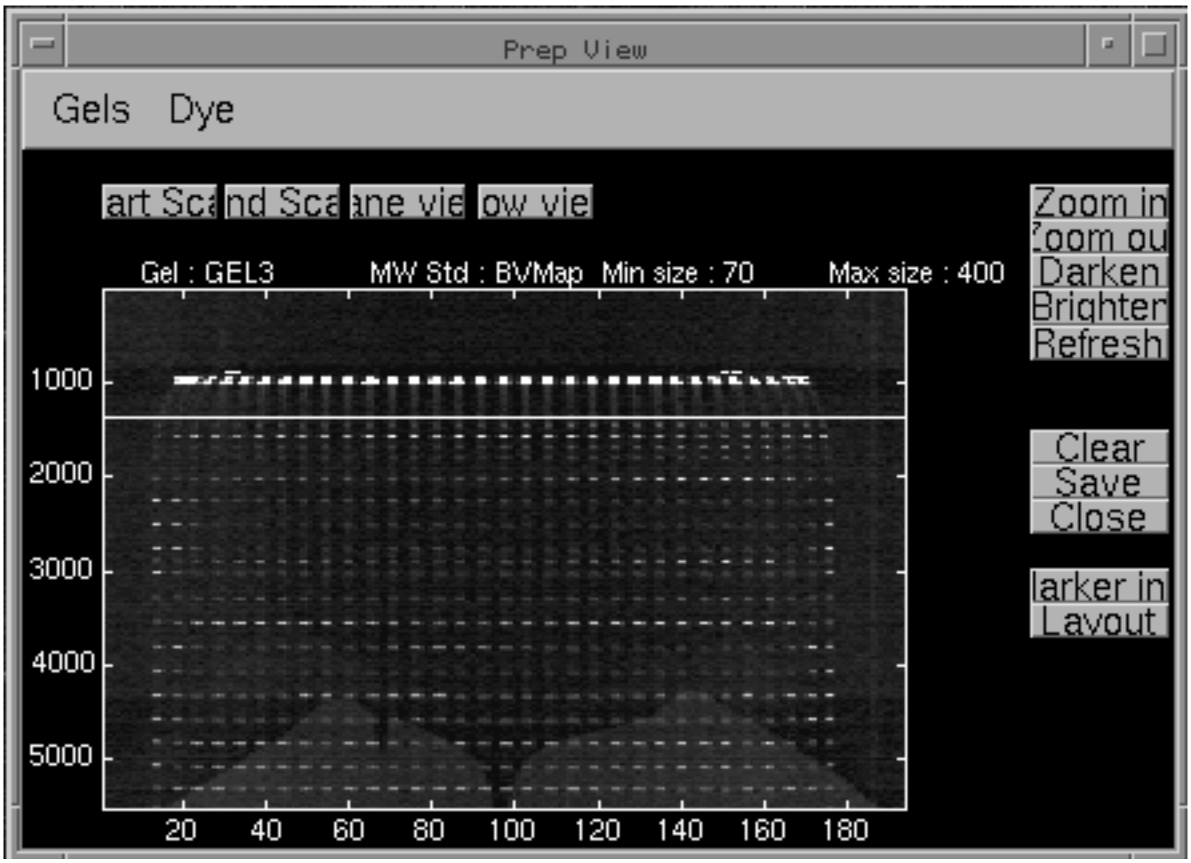


Figure D.1. The Prep View window with Start Scan already defined (Gel_orientation 'reverse').

There are two menu items for *prep_view*:

(1) Gels : for loading a different gel in the study;

(2) Dyes : for loading the image of a different dye of the current gel.

To the right of the gel image, there are eight control buttons:

(1) Zoom in: for zooming in to a selected region of the image;

(2) Zoom out: for zooming out;

(3) Darken : for darkening the image display;
(4) Brighten : for brightening the image display;
(5) Refresh : for re-displaying the image (on certain display device, the image may not automatically refresh);
(6) Clear : for deleting all the current edits;
(7) Save: for saving the current edits;
(8) Close : for closing and exiting *prep_view*.
(7) Marker info. Brings up a window which graphically displays the markers that have been loaded on the gel.
(8) Layout. Brings up a window which displays the "layout" file for the gel.

There are four buttons in the area above the gel image: Start Scan, End Scan, Lane View, and Row View.

The "Start Scan" and "End Scan" buttons are for cropping away excess gel regions. The "Start Scan" button is particularly useful for removing the primer region. To crop the image, click on the "Start Scan" button, and then click on a point in the image. A horizontal line indicating the selected start scan will be drawn. The user may click on the image repeatedly to shift the start scan line until satisfied. The "End Scan" is defined similarly by clicking on the "End Scan" button. Click on the "Save" button to save the start and end scans.

The "Lane View" button is useful for viewing a lane and ascertaining the molecular weights that are captured on the gel. Click on the "Lane View" button, then select a rectangular region enclosing a lane. A new specialized window, "Prep View : Lane", is opened and it shows the electropherogram of the selected lane region. We can inspect the electropherogram to ascertain the minimum and maximum size standard captured on the gel. When necessary, "edit <gel> settings" to change the values of the parameters "Min_size_standard" and "Max_size_standard".

The "Row View" button is useful for verifying the gel's loaded lanes. Click on the button, then select a horizontal region on the image that contains a row of molecular weight data. A new specialized window, "Prep View : Row", is opened and it shows the cross-sectional profile of the selected row region. Each peak corresponds to a size standard band in the cross-sectional region and can be interpreted as corresponding to a loaded lane. When necessary, "edit <gel> layout" to correct the gel's layout.

Click the "Close" button to exit *prep_view*. A dialog box may pop up to ask to save the start and end scans if one or both of them has been defined.

D.4.2. image_view

This program is used to visually inspect and edit gel images for lane tracking and size standard calibration. This is the companion program for *image_call*.

Program operation

image_view is a graphical user interface that the FAST-MAP user can bring up after (or before) *image_call* has automatically constructed the lane/size grid. To use *image_view,* the user types the command:

```
>> image_view <nickname>
```

e.g.

```
>> image_view STUDY1
```

In *image_view,* user may:
(1) visually INSPECT and assess *image_call*s lane/size grid;
(2) REPAIR grid lines, without rerunning *image_call*.
(3) DRAW grid corners and gel boundaries to help *image_call* in avoiding making its mistakes due to gel smiles or primer bands.

image_view can be called before or after a sizing grid has been constructed by image_call or *allele_call*. If it is invoked after a sizing grid has already been built, then *image_view* goes to the INSPECT mode where the grid is overlaid on top of the image of the size plane. If a size grid doesn't exist, *image_view* goes directly to the DRAW mode where the user can define the gel corners and boundaries.

Here are some *image_view* protocols for solving specific *image_call* problems:

- Verify the correctness of the sizing grid constructed by image_call.

- Go to INSPECT mode.

- The shape of the gel image is warped due to gel smiles.
  - Go to DRAW mode and define the corners and boundaries of the gel.

- There are high amplification intensity of primers in the smaller base pairs regions.
  - Go to DRAW mode and define the corners and boundaries of the gel.
  - Or, use *prep_view* to define the start scan and/or the end scan, and then re-run*image_call*.

- A small number of grid peaks have been misplaced.
  - Go to REPAIR mode, choose the element "Peak". To relocate a misplaced peak, click on the grid peak, and then click on its desired location on the image. To view its new position, click on "Refresh". Repeat for the other misplaced peaks.

- An entire lane has been misplaced.
  - Go to REPAIR mode, choose the element "Lane". First, remove the misplaced lane by selecting DELETE and clicking on the lane. Then select INSERT, and click on a size standard band to insert the lane there.

- An extra lane has been inserted.
  - Go to REPAIR mode and DELETE the extra lane.
  - Or, if the layout was wrong, edit "layout" to put a "blank" under "size_std" in that extra lane, then re-run *image_call*.

- A lane was missing.
  - Go to REPAIR mode and INSERT the missing lane (remember to click on an existing molecular weight band when inserting the lane there).
  - Or, if the layout was wrong, edit "layout" to add the missing lane, then re-run *image_call*.

- An entire row has been misplaced.
  - Go to REPAIR mode, choose the element "Row". First, remove the misplaced row by selecting DELETE and clicking on the row. Then select INSERT, and click on a size standard band to insert the row there.

- An extra row has been inserted.
  - Go to REPAIR mode and DELETE the extra row.
  - Or, if the Min_size_standard and Max_size_standard values in the gel's "settings" were wrong, edit "settings" to correct, then re-run *image_call*.

- An entire row was missing.
  - Go to REPAIR mode and INSERT the missing row (remember to click on an existing molecular weight band when inserting the row there).
  - Or, if the Min_size_standard and Max_size_standard values in the gel's "settings" were wrong, edit "settings" to correct, then re-run *image_call*.

Example run

Invoke the *image_view* program by typing

```
>> image_view <nickname>
```

e.g.

```
>> image_view STUDY1
```

The program responds with the following messages:

```
Initializing study STUDY1 : GEL1...GEL2...Done.
Loading manifold data...Done.
Reading image data for dye 4...Done.
```

and brings up a window with the *image_view* interface in the "Inspect" mode.

Image_view: INSPECT mode

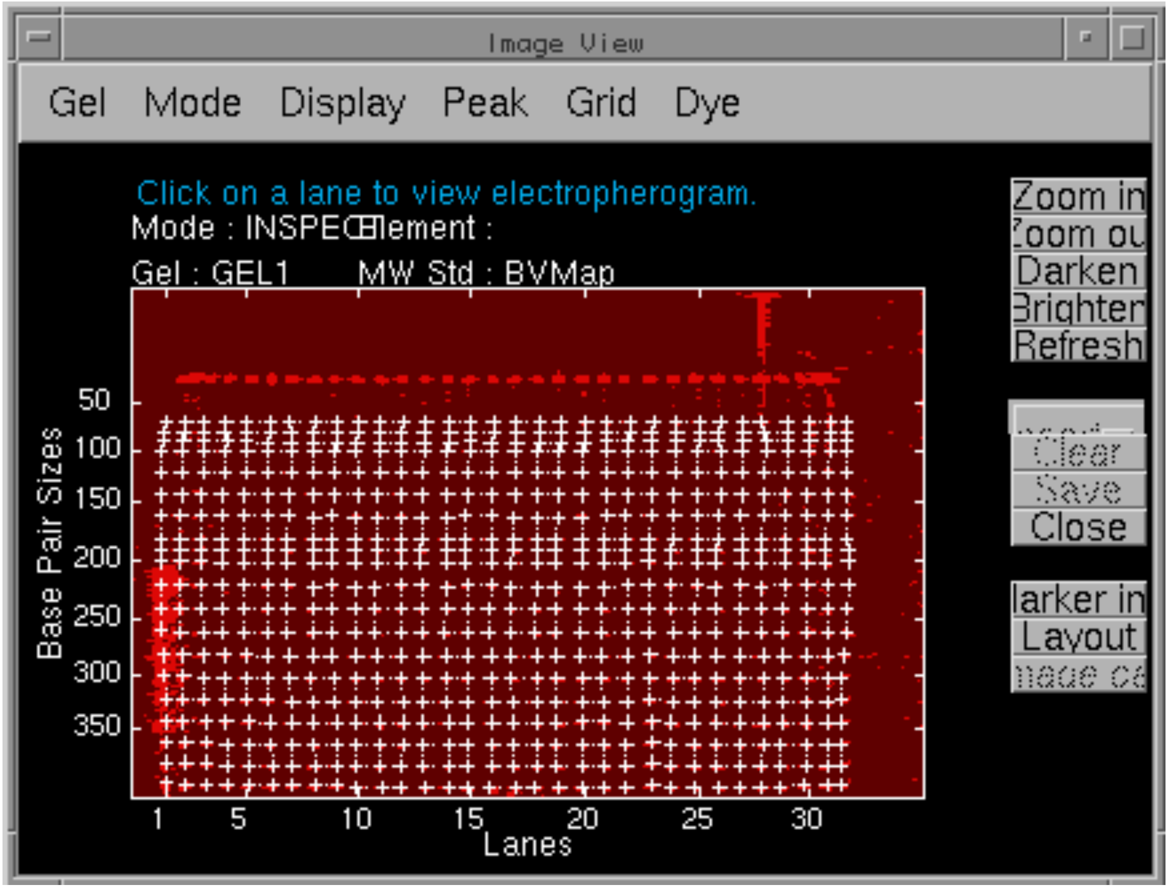


Figure D.2. The "Inspect" mode (Gel_orientation 'reverse').

<u>Mode 1</u>: INSPECT

The pre-z-scaled size marker image is shown with *image_calls* grid superimposed.

Six *image_view* menu items are available in the INSPECT mode:
(1) <u>Gel</u>. This menu item is for going to another gel in the study.
(2) <u>Mode</u>. This menu item is for going to the other *image_view* modes (namely Repair and Draw).
(3) <u>Display</u>. This menu item is for turning the image, grid, or peak displays on/off.
(4) <u>Peak</u>. This menu item is for selecting the pattern for displaying the peaks.
(5) <u>Grid</u>. This menu item is for selecting the line pattern for displaying the grids.
(6) <u>Dye</u>. This menu is for displaying other dye planes from the current gel.

In the pane on the right hand side, there are six active buttons:
(1) Zoom in. Click on this button, then select a region on the image for zooming in.
(2) Zoom out. Zoom out to the previous level.
(3) Darken. Darken the gel image.
(4) Brighten. Brighten the gel image.
(5) Refresh. Refresh the displays.
(6) Close. Close the *image_view* window.
(7) Marker info. Brings up a window which graphically displays the markers that have been loaded on the gel.
(8) Layout. Brings up a window which displays the "layout" file for the gel.

To see the electropherogram of a particular lane, click on the grid line for that lane. The electropherogram of that lane will be displayed in an "Image View : Electropherogram" window.

Information about the current gel is displayed in the area above the gel image. Above these information, a help message is displayed periodically to prompt the user for the relevant actions whenever necessary.

Image_view: REPAIR mode

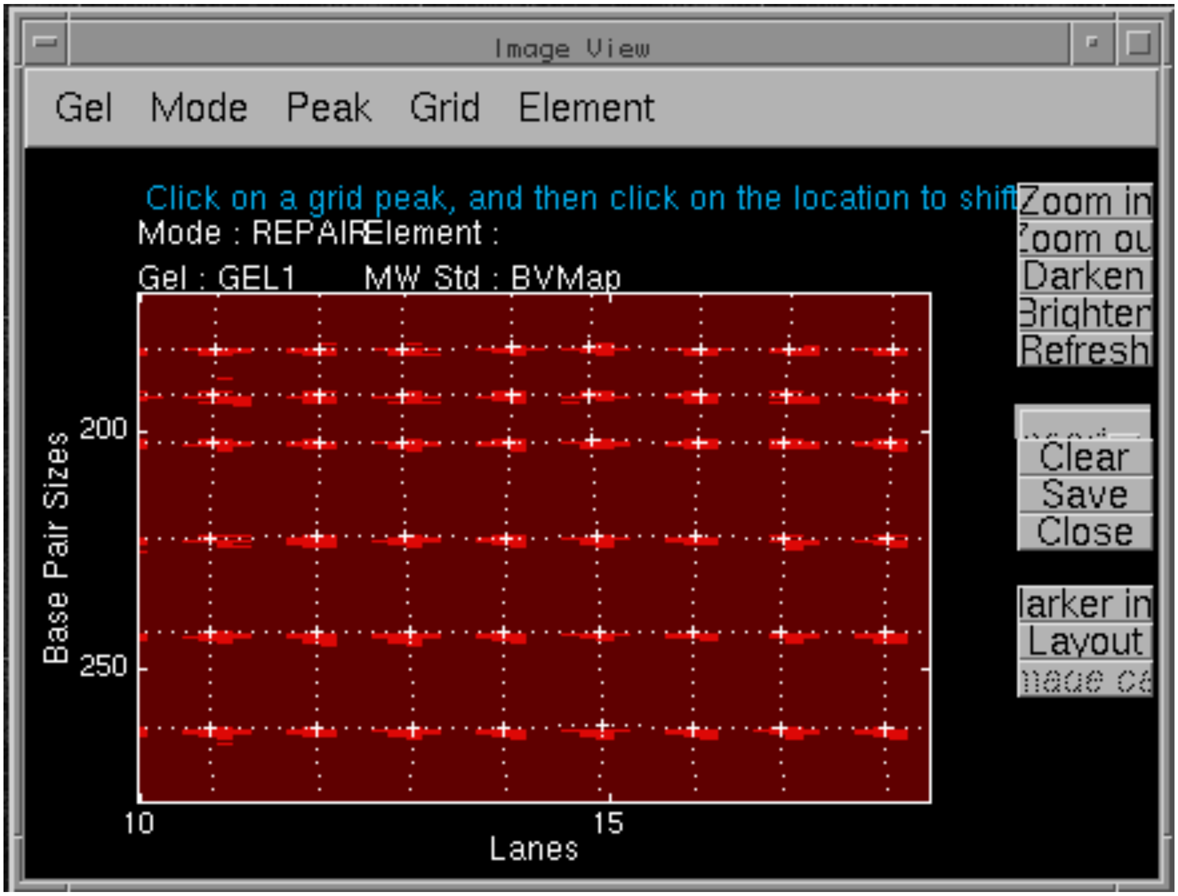


Figure D.3. The "Repair" mode, zoomed in.

<u>Mode 2</u>: REPAIR

The user uses this mode to interactively repair the grid without calling the *image_call* program. There are three independent "elements" of REPAIR mode: lane, row, and peak. Each element's user interface provides INSERT (default) and DELETE as a button with a popup-menu.

When performing REPAIR actions, the user may need to zoom in to particular regions of the image. When zooming in to a region near the edge of the image, it is alright to select a region which includes an area outside the image, as long as the selected region also includes part of the gel image.

Element 1: Grid lane

The "Lane" element is used to add or remove a lane from the grid. A lane is input by clicking on a point inside the image. The peak positions of the new lane is very sensitive to the point entered by the user; the user must click as close as possible to an EXISTING SIZE STANDARD BAND on the lane being inserted.

Element 2: Grid row

The "Row" element is used to add or remove a row from the grid. A row is input by clicking on a point where the user wishes the new row to be located. However, if the grid already contains a row for each of the sizes defined inside the molecular weight standard, insertion is not allowed, and a warning message is displayed. To move a row the user must delete a row first, and then inserting another in the desired location.

Just like grid lanes, the peak positions of the new row is very sensitive to the point entered by the user. The user must click as close as possible to AN EXISTING SIZE STANDARD BAND on the row being inserted.

Element 3: Grid peak

The "peak" element is used to change the location of a peak inside the grid. This is NOT done by dragging the peak to be changed. Instead, there are two clicking actions involved: first, select a peak by clicking as close to it as possible; second, click on the new location inside the image to move the peak there. To display the changes, click on the "Refresh" button.

Chick on the "SAVE" button to save all the edits for the current gel.

Image_view: DRAW mode

Figure D.4. The "Draw" mode.

<u>Mode 3</u>: DRAW

The user uses this mode to provide *image_call* with additional information about the shape of the gel. The user should call *image_call* again after providing the new information. After a correct grid has been constructed with the additional information, the user can then call *allele_call* to genotype the gel.

There are two independent "elements" in the DRAW mode: corner, and boundary. User can select any of these elements from the menu "Element". The default is to start with defining the "corners", and then proceed to defining the various "boundaries".

There is an "IMAGE CALL" button that the user can click on to call *image_call*. Since this may take 10-15 minutes, it may be more practical to close *image_view*, call *image_call* from the Matlab window, and then *image_view* the gel again.

<u>Element 1</u>: Corner

When the user selects the DRAW mode, the default element is "Corner". The user can also select "Corner" under the menu "Element". Under the menu "Corner", the user selects a corner: top-left, top-right, bottom-left, or bottom-right. Selecting a corner from the "corner" menu automatically puts the user into a "zoom" mode, indicated by a "crosshair" cursor; the user should not click on the "Zoom in" button for this purpose. The user then drags a rubber-band rectangle inside the image (note again that it is alright to include the area outside the image, especially when the corner is very close to the edge of the image).

After a portion of the image has been selected, it is displayed inside a second figure labeled with the corner name. In the image region of this figure, the user can perform only one action: click on a point. Each click redefines the selected corner. Clicking on "Close" closes this window and displays the defined corner inside the main Image View window.

The user should define all four corners before choosing the "Boundary" element. When all four corners are defined, *image_view* automatically creates a box from the four corner points. This forms the initial shape of the gel. If this is good enough, the user can "Close" and call *image_call*. Otherwise, the user should define the boundaries to further refine the shape of the gel.

<u>Element 2</u>: Boundary

The pre-requisite for entering the "Boundary" element is that all four corners must be specified. A warning message will explain this to the user in the help message pane on top of the gel image, if the user selects the "Boundary" element before defining all the four corners. Upon entering this element for the first time for a gel, the boundaries contain only the four corner points.

Using the menu "Boundary" the user selects one of the four boundaries: Top, Bottom, Left and Right. This again puts the user into a "zoom" mode indicated by a cross-hair cursor. The user selects an area enclosing the boundary by dragging a rubber-band rectangle across the image containing all or part of the desired boundary. If only part of the boundary was selected, *image_view* automatically includes the entire boundary.

After a portion of the image has been selected, this portion is displayed inside a second figure labeled with the boundary name. In the image region of this figure, the user can perform only one action: click on a point. Each click either adds the point to or deletes it from the boundary. Clicking on "Close" closes the boundary window and displays the whole boundary inside the main *image_view* window.

Click on "Save" to save the information entered. The "Close" button in the main *image_view* window will close the window and exit *image_view*. The user may then call *image_call* to re-build the size grid based on the new information.

D.4.3. marker_view

The *marker_view* program can be used to visually inspect multiple electropherograms for ascertaining the actual allele range of the genotypes for the population being analyzed, especially when using a marker panel for the first time.

Program operation

To use *marker_view,* the gel or study must have the size grids constructed successfully with *image_call* and/or *image_view*. Then, the marker windows can be viewed by typing:

```
>> marker_view <nickname>
```

e.g.

```
>> marker_view STUDY1
```

There are two modes of *marker_view*. The default mode is the "image" mode, where all the lanes on a gel is displayed simultaneously as intensity images for rapid viewing. The user can see each lane's electropherogram by clicking on the lane in the image. This mode is invoked by typing:

```
>> marker_view <nickname>
```

or

```
>> marker_view image <nickname>
```

In the alternative mode, the "lane" mode, the lane traces are displayed. The user can quickly step through the electropherograms for a marker on a gel viewing multiple lanes at a time. The number of lanes displayed each time can be controlled by the user in the "settings" file (See "settings").

In both modes, the user can expand or contract the window by changing the values in "Start bp" and "End bp". The program will also update the "markers" file automatically if any changes have been made.

Example run: "Image" mode

Invoke the *marker_view* program on a gel or study that has been "image_call'ed" by typing

```
>> marker_view <nickname>
```

e.g.

```
>> marker_view STUDY1
```

The program responds with the following messages:

```
Initializing study STUDY1 : GEL1...GEL2...Done.
```

and then opens a Marker View window showing the gel window of the first marker in the gel or study's panel.

Marker View window: "Image" mode

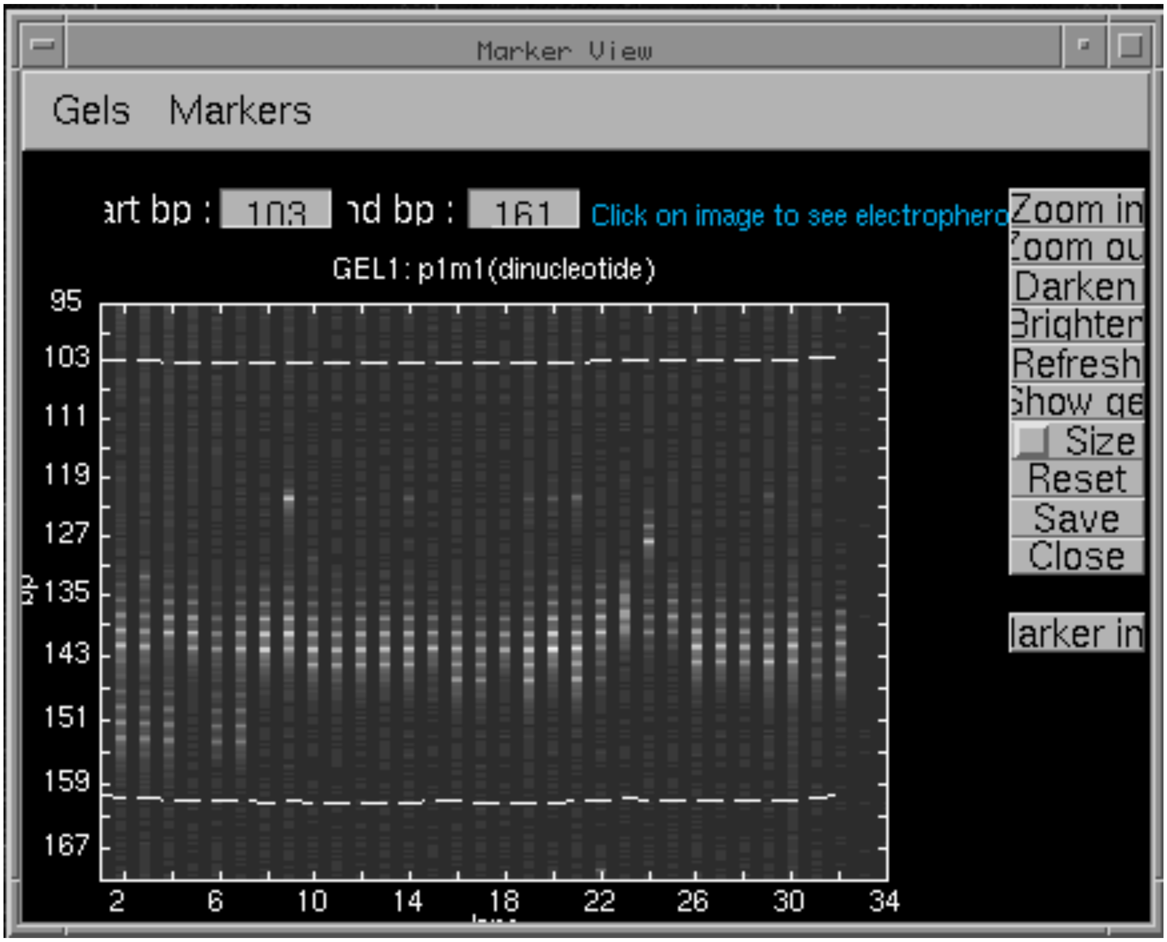


Figure D.5. The Marker View window in "Image" (default) mode.

There are two menu items: "Gels" and "Markers". The current gel and marker is displayed on top of the image. To go to another gel or marker, simply select from the corresponding menu.

There are also two text boxes labeled "Start bp" and "End bp" above the image. Initially, the marker window specified in the "markers" file is displayed. In the image, the two dotted lines mark the marker window. To change the marker window, simply type in the appropriate number (in base pairs) in the text boxes, and hit the "RETURN" key. The image will expand accordingly if a bigger window is needed. To see the electropherogram of a particular lane, simply click on the lane in the image.

The buttons in the right column are:

(1) "Zoom in": to zoom in, click on this button and then select a region in the image;
(2) "Zoom out": zooms out to the last display;
(3) "Brighten": brightens the image;
(4) "Darken": darkens the image;
(5) "Refresh": refreshes the display;
(6) "Show gel": displays the entire gel (to go back to the restricted marker window view, click on "Refresh");
(7) "Size grid": overlays the molecular weight grids on the marker bands (toggles the display of the sizing grid by clicking on the button again);
(8) "Reset": resets back to the original value in the "markers" file for the current marker;
(9) "Save": saves the new marker window in the "markers" file for the current marker;
(10) "Close": done with *marker_view*.
(11) "Marker_info" brings up a window which graphically displays the schematic layout of all the markers that have been loaded on the gel.

Note that the user should re-run *allele_call* with "Redo_quantitation" set to "yes" to re-analyze a gel that has been genotyped previously under an old definition of marker windows. If any problem occurs, it may be necessary to call "reset_now panel <panel_name>" to clear the previous marker libraries under the old marker window definitions (see "reset_now").

Example run : "Lane" mode

Invoke the "lane" mode of the *marker_view* program on a gel or study that has been "image_call'ed" by typing

```
>> marker_view lane <nickname>
```

e.g.

```
>> marker_view lane STUDY1
```

The program responds with the following messages:

```
Initializing study STUDY1 : GEL1...GEL2...Done.
```

and then opens a Marker View window showing the electropherograms of the first marker in the gel or study's panel.

Marker View window: "Lane" mode

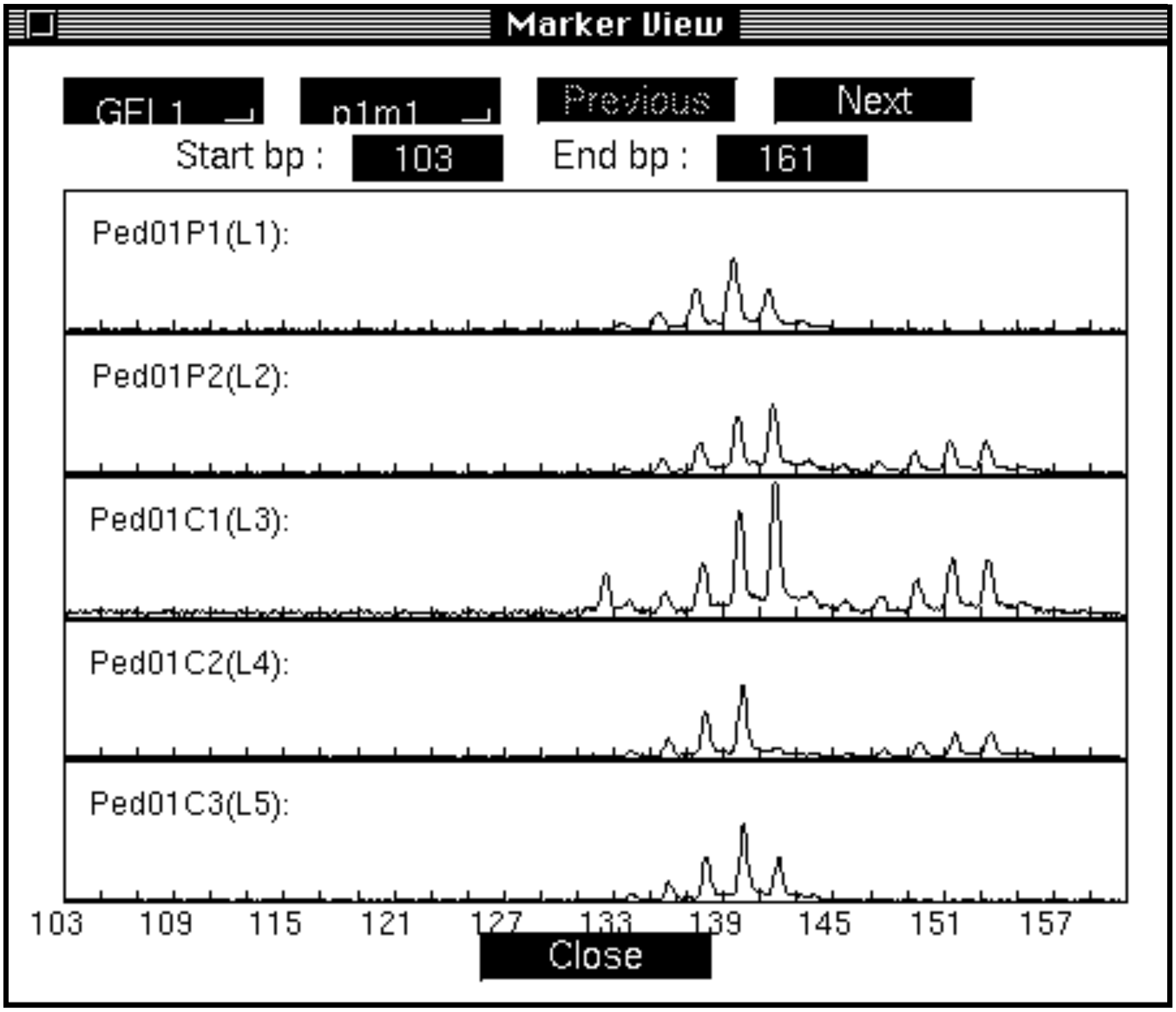


Figure D.6. The Marker View window in "Lane" mode.

There are two rows of buttons in the area on top of the electropherograms. In the first row are four buttons:

(1) The gel button displays the current gel's name. When clicked, it brings up a pop up menu of all the gels in the study, and the user can choose any of them

(2) The marker button displays the current marker's name. When clicked, it brings up a pop up menu of all the markers in the panel, and the user can choose one of them to go to that marker.

(3) The "previous" button goes to the previous screen.

(4) The "next" button goes to the next screen, until all the lanes of the current marker have been viewed. In this case, the user should choose another marker from the marker button.

The second row contains two text boxes:
(1) The "Start bp" box: Typing in a new number (in base pairs) in this box will change the left boundary of the marker windows displayed.
(2) The "End bp" box: Typing in a new number (in base pairs) in this box will change the right boundary of the marker windows displayed.

The "Close" button on the bottom closes the *marker_view* window. When any of the markers' windows has been changed, a dialog box will pop up to ask if the user wants to save the changes to the "markers" file. If the user chooses "yes", the "markers" file's "Min_bp" and "Max_bp" for the markers will be updated. If the user chooses "no", the "markers" file is not changed. Instead, a summary of the marker windows is printed to the Matlab window for the user's reference.

D.4.4. allele_view

The *allele_view* program is a standalone Matlab application that provides graphical visualization of *allele_call*'s genotyping results. After running the *allele_call* program, the user invokes *allele_view* to inspect and edit the allele calls.

For reviewing allele calls, it is often helpful to:
(1) Review graphical presentations of the (a) underlying electropherogram data, (b) inferred DNA concentration determinations, and (c) different allele calls of the genotyping algorithms. These representations can provide the user with far more information than just reading textual output files.
(2) Review the above graphical presentations of the genotyping experiments from all the individuals in the same family as defined in the "pedigrees" file.
(3) Rank the allele calls by confidence. For example, when the allele calls are ranked so that the least confident genotypes appear first, the user can then focus their efforts on the most problematic calls, which represent 1%-10% of genotypes with good gel data. The vast majority (90%-99%) of routine allele calls can then be rapidly skimmed (or, the user can eventually trust the computer on these calls).
The *allele_view* program provides these three crucial graphical review functionalities, along with allele editing capability.

In the main "Allele View" window, one genotype result is displayed at a time. This display has three main panes that respectively show (1) the electropherogram data, (2) the DNA gel band quantitation, and (3) the allele calls. The user can navigate through the set of allele calls for a single gel experiment, including serially reviewing the results in order, from least confident to most confident. Specialized windows can be modularity invoked from this "Allele View" window by clicking on a pane:
(1) "Allele View Electropherogram", which shows the extracted electropherogram trace of the data. No user input is accepted.
(2) "Allele View Quantitation", which shows the fitted peaks that quantitate that electropherogram data. A quality measure is shown. No user input is accepted.
(3) "Allele View Genotype", which shows diverse views of genotype results, including different algorithm determinations and different comparisons. Quality measures are shown. The user may edit the alleles here.
(4) "Allele View Family", which simultaneously displays the genotypes, electropherograms, or quantitations from individuals that belong to the same family.

From the *allele_view* graphical windows, the user can view and edit all the genotyping results thus far computed and stored by the *allele_call* program. When the graphical windows are closed, the user can continue operating FAST-MAP from the Matlab command window.

It may be useful to print out hardcopies of the windows for reference at a later time. Click on the "Print" button of the window and the content of the window will be printed directly to the printer as specified in the "preferences" file. Alternatively, type "P" (upper case) inside the window. If the user does not wish to print the pictures immediately, type "S' (upper case) inside the window and it will be appended to a postscript file called "allele_view.ps" in the gel or study's "output/" directory. (In using "S", first make sure that there is no "allele_view.ps" file in the "output/" directory that remained from previous *allele_view* session, unless the current windows should be collected together with those already in the old "allele_view.ps" file.) Use the operating system's postscript printing program to print out "allele_view.ps" after inspecting and collecting pictures from all the relevant genotypes.

Program operation

To run the *allele_view* program, the user types the command

```
>> allele_view <nickname>
```

in Matlab. The <nickname> can be either a gel or a study. Only one gel or one study can be viewed at one time with *allele_view*. Change the "Allele_view settings" in the gel or study's "settings" file before running *allele_view* (see "settings" in "Annotating genotyping data"). The user does not need to re-run *allele_call* to view the genotypes under the new allele_view settings.

Example run

Once the *allele_call* program has finished its operations, the "/<gel>/display/" files are generated and the *allele_view* program can then be run. Instructions are given here for running the *allele_view* program, using the demo on our CMU computers as an example. The steps are followed by typing in the client Matlab window of the host machine.

Allele View window

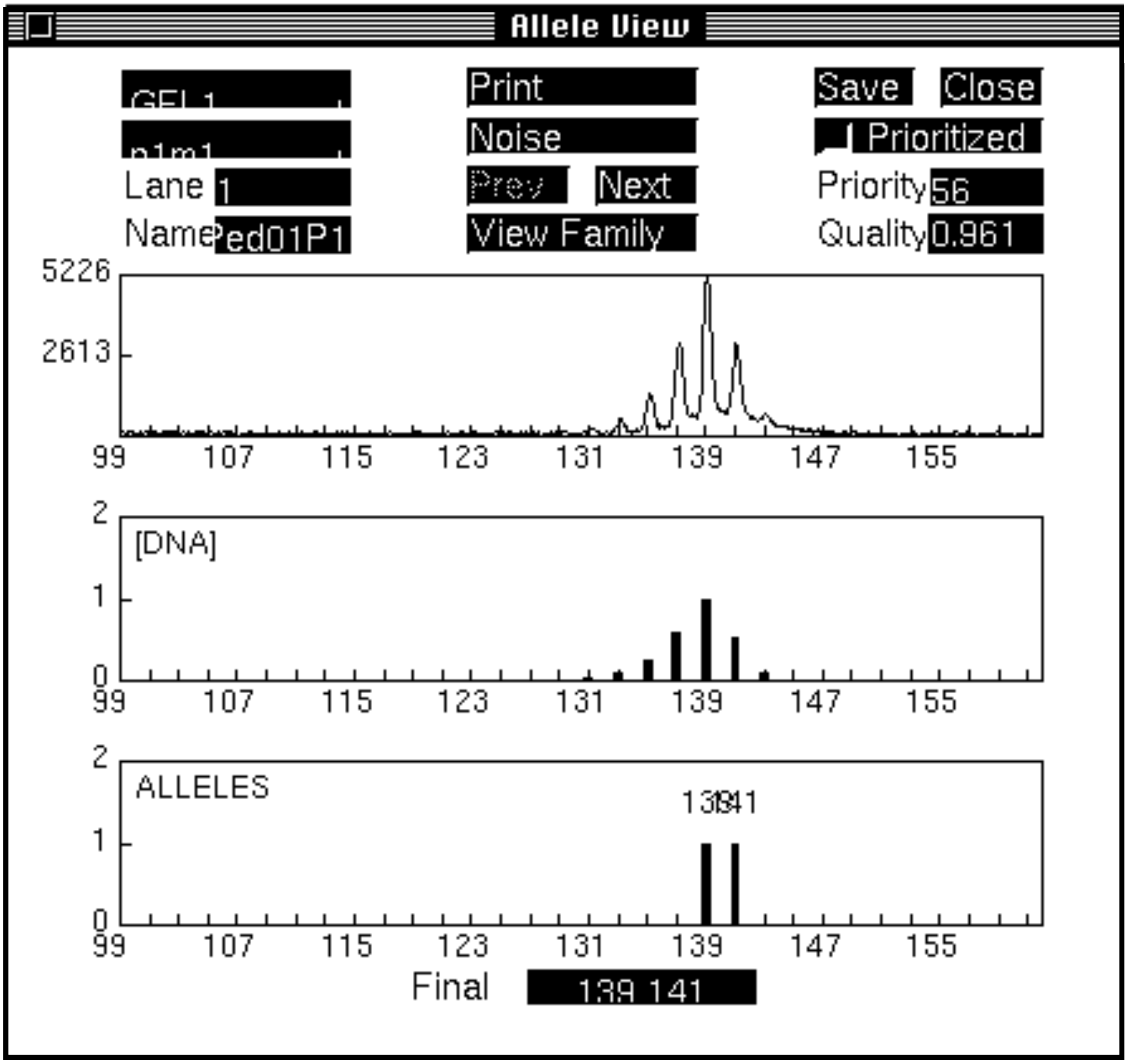


Figure D.7. The main "Allele View" window.

<u>Step 1.</u> View the main "Allele View" window.

The *allele_view* program can be started once *allele_call* has completed its genotyping operations. Be sure that the X-window server is properly configured. At the Matlab prompt, type:

```
>> allele_view <nickname>
```

For example, typing:

```
>> allele_view STUDY1
```

will have the computer reply:

```
Initializing study STUDY1 : GEL1...GEL2...Done.
Reading genotypes for STUDY1 : GEL1...GEL2...Done.
Sorting results (worst_first) ......Done.
```

and then bring up the window shown in Figure D.7.

The Allele View window lets the user visually navigate through the gel experiment. At the top of the window are control pane items. In the first column:

- The current gel's nickname is shown in a pop-up menu of component gel names for the study.
- The marker name is shown in a pop-up menu of valid marker names for the gel study. The user can select a marker name from the pop-up menu; the computer ensures that only valid selections are made.
- The "Lane" number is shown as an editable text The user can enter any valid lane number here to go to a particular lane on the current gel for the current marker.
- The "Name" of the current sample is also shown as an editable text. The user can enter any valid sample name here to go directly to the latest experiment for this sample. If there are duplicate samples, FAST-MAP will go to the one on the last gel in the study's "layout" which contains this sample. If there are duplicate sample on that gel, FAST-MAP selects the one with the largest lane number . If necessary, use the Gel and Lane combination to go to a specific experiment.

In the second column of the control pane items:

- A "Print" button is provided to print the window to the printer specified in "preferences".
- A "Noise" button is provided to set the allele calls to (0,0). This button is pressed to indicate that the experiment shown does not contain useful data signals, e.g., PCR failure.
- The "Next" button moves to the next ordered genotype when "Prioritized" is "on". With "Prioritized" off, FAST-MAP moves to the next lane, marker, and gel in the study.

- The "Previous" button moves to the previous ordered genotype when "Prioritized" is "on". With "Prioritized" off, FAST-MAP moves to the previous lane, marker, and gel in the study.
- The "View Family" button is clicked to bring up a specialized window which displays the genotypes, electropherograms, or quantitations of all the family members of the current individual.

In the third column of the control pane items:

- The "Save" button is clicked to record all allele edits that the user has made, but <u>does not</u> update any of the relevant results files.
- The "Close" button is clicked to close all the Allele View windows. If there are any edited genotypes that have not been saved, a dialog window will pop up to ask the user whether to save them or not.
- The "Prioritized" check box starts in the "off" position. When clicked to set the "on" position, the genotype experiments are ordered based on the "settings" file. When prioritized from worst result to best result, the ordering enables the user to focus on the more difficult allele calls first.
- The "Priority" number is shown as editable text. The priorities range from "1" through "n" where n is the number of genotype experiments on the gel. If "Prioritize_results" is set to "worst_first", then a priority of "1" represents the worst result. If "Prioritize_results" is set to "best_first", then a priority of "1" represents the best result. The user can enter a valid priority number when "Prioritized" is "on" to move to another genotype experiment.
- The "Quality" value shows the computer's assessment of how good the genotype call was. This assessment is a composite of relative signal, the electropherogram fit, the deconvolution fit qualities, and the degree of consensus between the different deconvolution algorithms. The qualities range from "0.0" (worst) to "1.0" (best). This value is not settable by the user.

At the bottom of the window is an edit box labeled "Final". The current allele call is shown inside the box. The user can edit this allele call by typing a new genotype and then hitting the RETURN key. See "Editing and saving genotypes" for other ways of editing the genotypes.

The three graphical panes in the middle of the window show the allele calling at a glance. The panes share a common x-axis of DNA size (shown in base pair units). Specifically:

(1) The electropherogram pane shows the data trace. The y-axis is recorded signal intensity.
(2) The quantitation pane shows the computed DNA sizes and relative concentrations. The y-axis ranges from 0 to 2.
(3) The genotype pane shows the consensus allele calls. If the corresponding line inside the results file contains edits, then these are displayed as text inside the genotype pane. The y-axis ranges from 0 to 2. The called alleles sum to 2, the number of alleles on diploid chromosomes. No allele calls are displayed for an experiment classified as noise by *allele_call* given a quality value of 0.

Clicking anywhere in a pane of the "Allele View" window opens a new window that is specialized for that pane. Each of the three panes opens its own unique specialized window. While the new window is opening, a "watch" cursor is displayed to indicate a prolonged computer operation. This watch cursor feature can be turned off by setting "Watch_cursor" in "preferences" to "no".

Allele View Electropherogram window

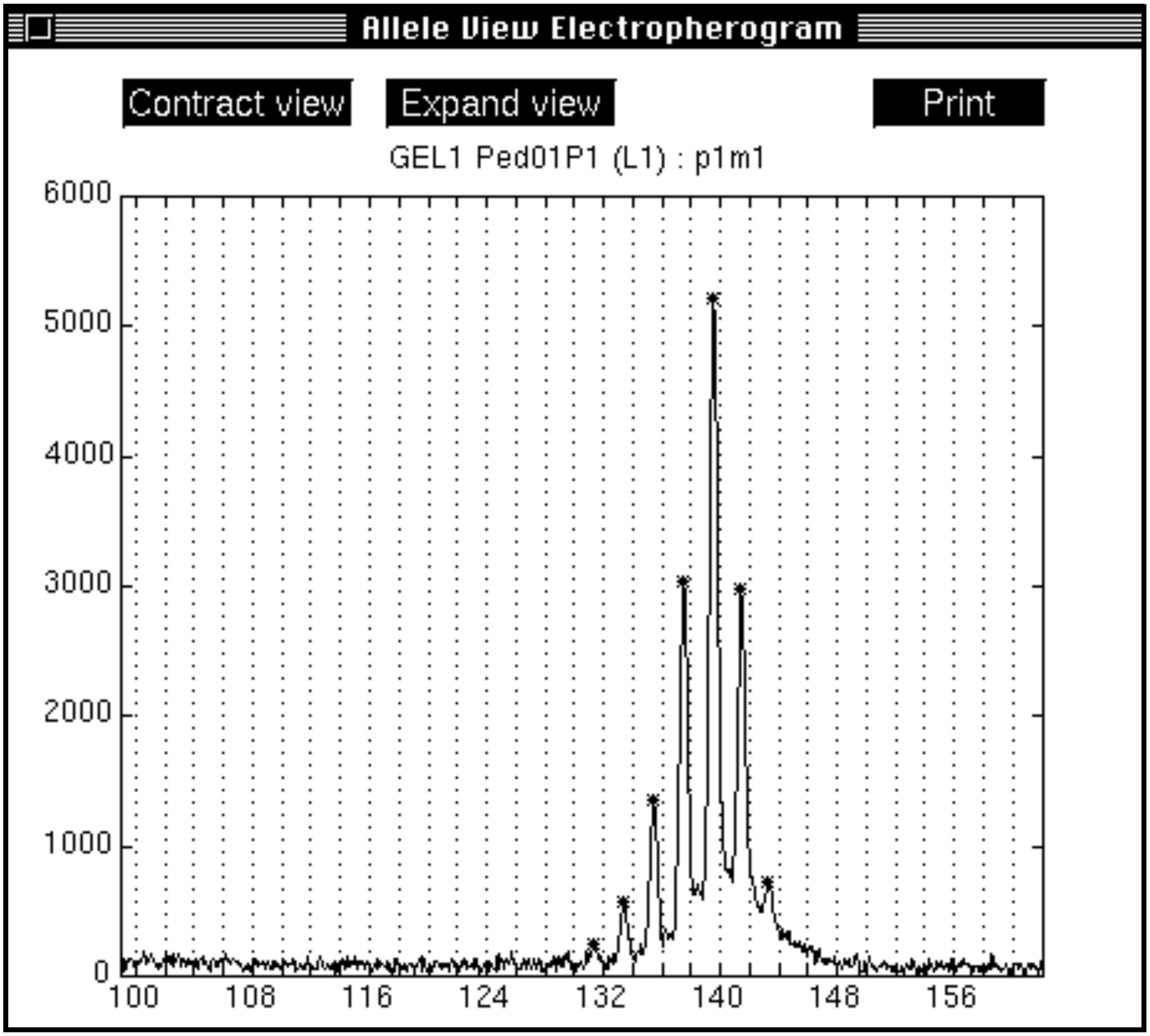


Figure D.8. The "Allele View Electropherogram" specialized window.

<u>Step 2.</u> View the "Allele View Electropherogram" specialized window.

"Allele View Electropherogram" window is a specialized window that shows the raw data recorded from the gel electrophoresis, and then tracked down the one dimensional lane. The DNA signal (y-axis) is plotted against the DNA size (x-axis) shown in base pair units. The asterisks "*" denote peaks detected by the computer program.

Click on the "Expand view" button to look progressively beyond the marker window that has been defined by the marker's "Min_bp" and "Max_bp" values in the "markers" file. The "Contract view" button narrows back the expanded window. If alleles outside the currently defined marker window are discovered, it may be necessary to "edit markers" and then re-run the genotyping with "Redo_quantitation" in the gel or study's "settings" set to "yes".

The "Print" button prints the content of the Allele View Electropherogram window to the printer specified in "preferences". A keyboard shortcut that produces the same result is to type "P" (Capital "P") inside the window. Alternatively, type "S" (Capital "S") to append the current window to a postscript file called "allele_view.ps", which resides in the "<gel>/output/" directory. To obtain a hardcopy of all the collected windows, the user can print out the file "allele_view.ps" at the end of the session using the computer's postscript printing program.

On a Macintosh computer, the window can be closed (click top left corner) or resized (drag bottom right corner) using the mouse pointing device. The window persists until it is explicitly closed by the user.

Allele View Quantitation window

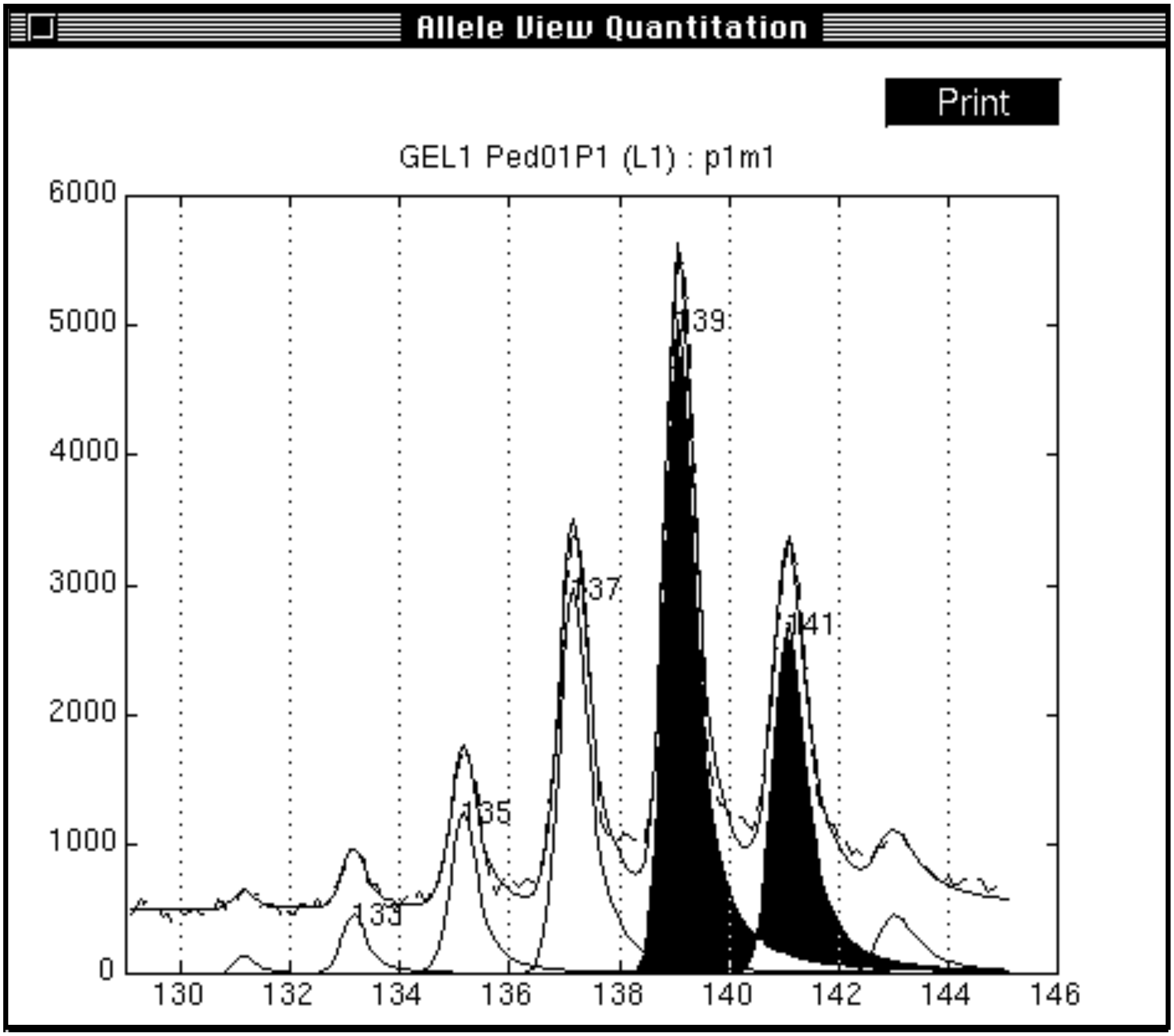


Figure D.9. The "Allele View Quantitation" specialized window.

Step 3. View the "Allele View Quantitation" specialized window.

"Allele View Quantitation" is a specialized window. The top portion of the window shows the fit of the sum of the predicted DNA bands (solid line) relative to the observed data (dashed line). The bottom portion of the window shows each predicted DNA band component (solid line) and its DNA size assignment. The leading left side of each band is modeled as a Gaussian function, and the trailing right side of each band is modeled as a Lorentzian function. The computer searches for the band shapes, locations, and sizes whose sum best fits the observed data. When found, these band shapes provide quantitative estimates of DNA size and concentration. The highlighted bands indicate the called alleles.

There is a "Print" button which prints the content of the Allele View Quantitation window to the printer. Alternatively, type "P" in the window, or type "S" to save to "<gel>/output/allele_view.ps". The window persists until it is explicitly closed by the user, for example, by the clicking the top left corner.

Allele View Genotype window

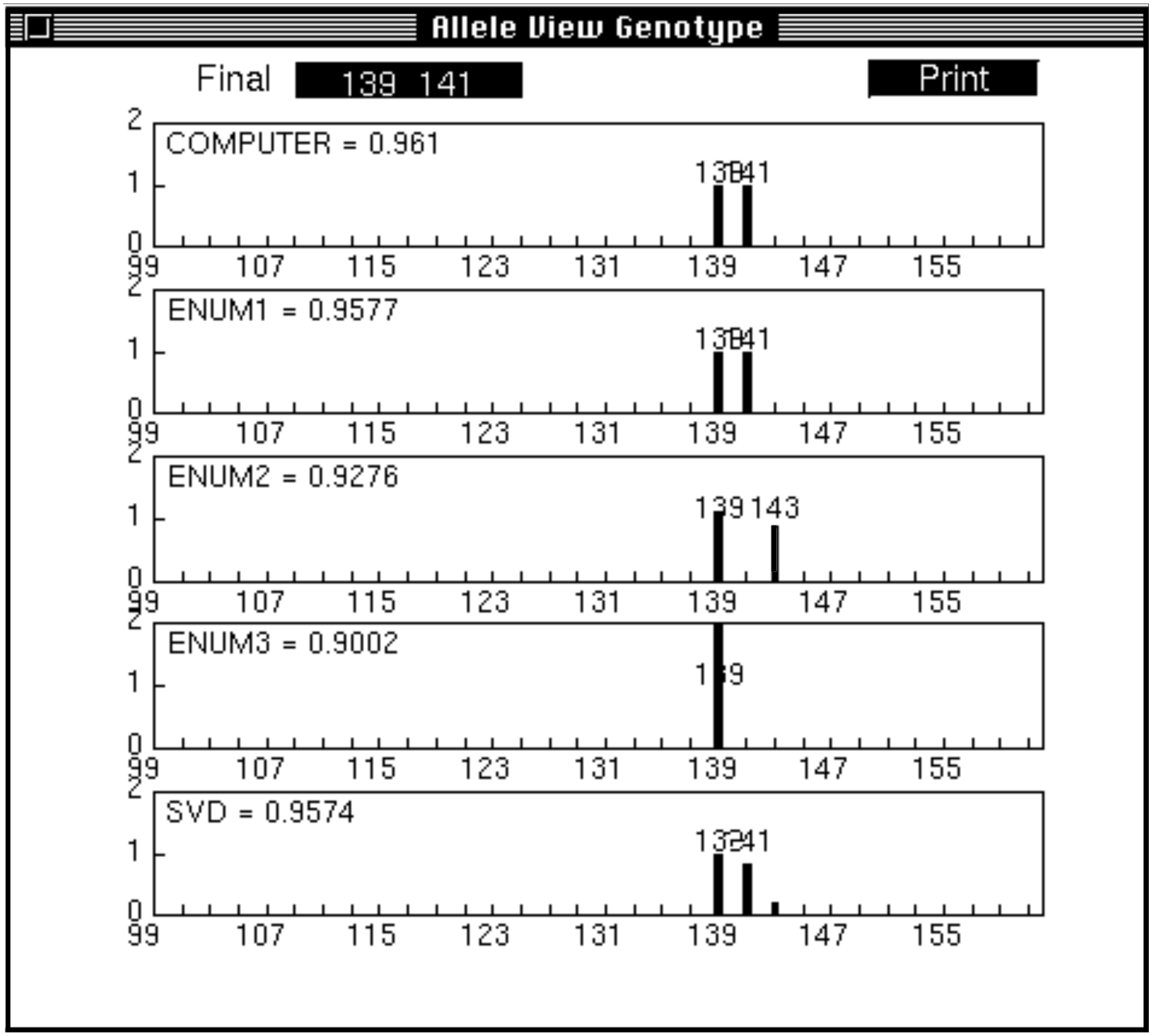


Figure D.10. The "Allele View Genotype" specialized window.

Step 4. View the "Allele View Genotype" specialized window.

"Allele View Genotype" is a specialized window. In the top portion of the window, the user can edit the computer's allele calls. In the bottom portion, the computer visually presents the computer's allele calling results.

To edit the genotype, the user types the new alleles in the editable text box labeled "Final". When both entries are "0", this denotes an uncallable final allele result. The computer accepts the edited values when the user hits the RETURN key to signify completion.

In the current visual presentation, three sets of panes are shown.
(1) The first pane shows the call by the "Computer". The genotype is shown together with its quality score. The two called alleles sum to one, and are displayed both as visual histograms and as printed allele size numbers.
(2) The second pane shows the result of the "SVD" pattern matching algorithm. SVD (short for "singular value decomposition") is a numerical matrix inversion algorithm that solves the equation y=Ax for the allele vector x, given the data vector y and the stutter pattern matrix A. The computer solves for the allele vector x, and performs additional post-processing to account for plus-A artifact and possibly inaccurate band sizing. The quality score quantitates the fit between the predicted alleles and the observed data.
(3) The last three panes show the results of the "ENUM" pattern matching algorithm. ENUM (short for "enumerate") tests out candidate allele pairs by arithmetically superimposing the two alleles' stutter patterns and comparing this sum against the observed data; the closest fit of predicted alleles to observed data determines the call. The quality score quantitates this fit between the predicted alleles and the observed data. The algorithm also varies the relative weight of the two alleles to account for unequal PCR amplification.

Like before, the window can be printed immediately by clicking on the "Print" button or by typing "P" in the window. Typing "S" saves the window content to "<gel>/output/allele_view.ps" that the user can print after the session. The window persists until it is explicitly closed by the user (by clicking on the top left corner, say).

Editing and saving genotypes

FAST-MAP's allele calls are not always correct. There are several ways to graphically edit incorrect allele calls. These editing methods use the main "Allele View" window or the specialized "Allele View Genotype" window.

(1) Some genotyping PCR experiments produce just noise or background signal, without any discernible allele bands. If FAST-MAP has erroneously attempted to call the alleles from such a "noise" experiment, click on the "noise" button in the main "Allele

View" window. This button will reset the genotype to (0,0), which indicates a "noise" result.

Note that turning on the "Output_noise_genotypes" option inside the "settings" file (see "Settings") forces FAST-MAP to attempt allele calling for all the data, noisy or not. Using its pattern matching capability, FAST-MAP can often discern correct alleles in data that (to the human eye) appears to be mere noise. However, this option can be time consuming.

(2) FAST-MAP's incorrect allele calls can always be textually corrected by the user. This correction is done using the editable text button labeled "Final", and is found in both the main "Allele View" window and the specialized "Allele View Genotype" window. After typing in the correct allele values in one of the "Final" boxes, the user must make sure to hit RETURN to tell FAST-MAP that he has finished typing.

(3) FAST-MAP's incorrect allele calls can also be graphically corrected by the user. Often, when FAST-MAP's first choice of alleles is incorrect, FAST-MAP will nonetheless display the correct alleles in alternative pane shown in the "Allele View Genotype" window. To select such a pane's genotype, click inside the pane. This click will set the final alleles to correspond to the selected pane.

The edits made during the course of an "Allele_view" session is saved inside the FAST-MAP environment, but not permanently inside the results file. This feature has been provided, not only to increase the speed of user-interaction, but also as a safety mechanism to avoid making any changes permanent unless the user is absolutely sure. The changes are incorporated into the results by periodically clicking on "Save" inside "Allele View".

Allele View Family window

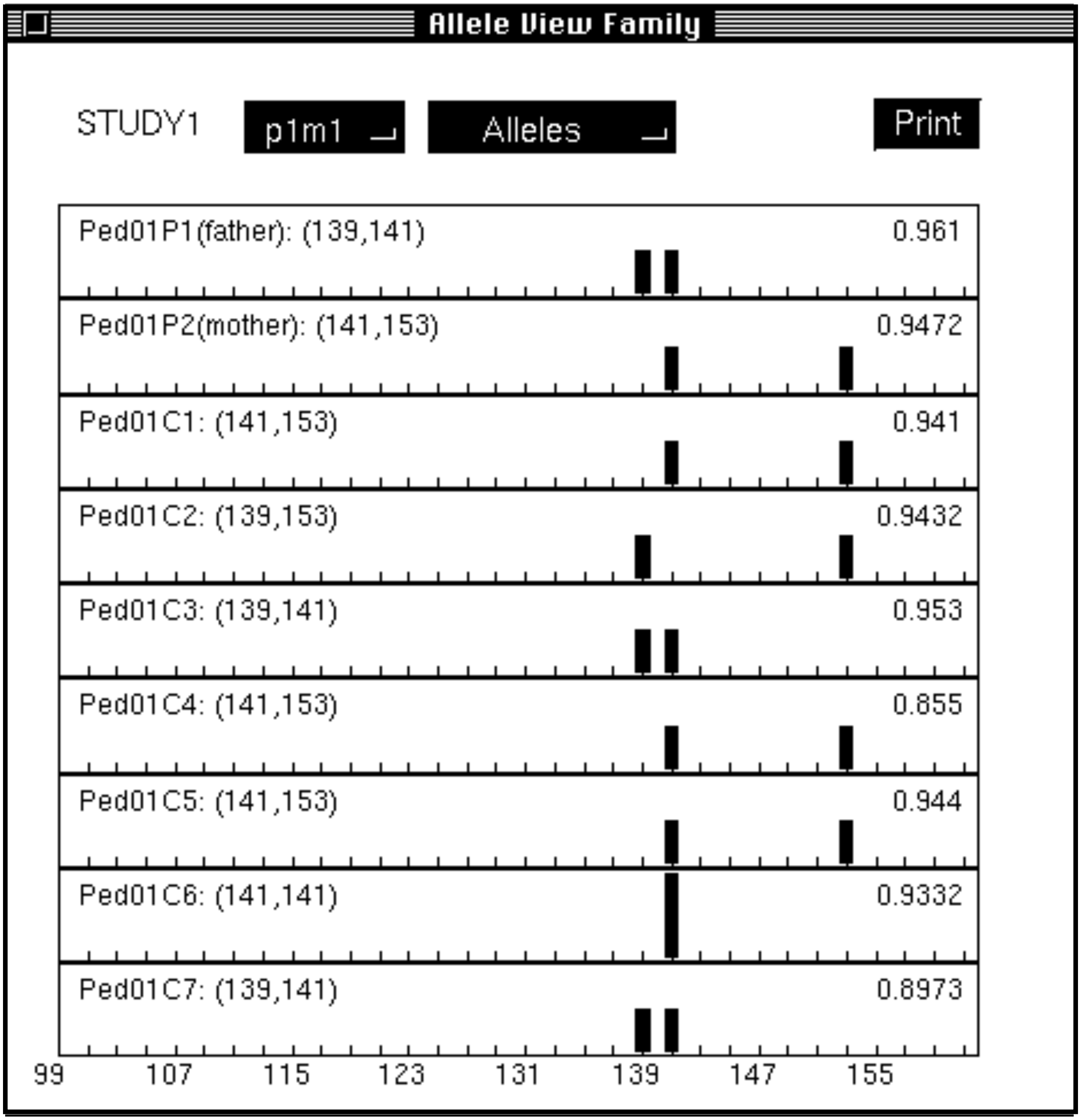


Figure D.11. The "Allele View Family" specialized window: Alleles.

Step 5. View the "Allele View Family" specialized window.

Allele View Family is a specialized window of Allele View which displays the parents and siblings of the current sample. To open this window, click on the "View Family" button inside "Allele View". Only those relatives which are present in the study are displayed. The parents (if shown) are labeled as "father" or "mother" next to their sample names, and the current individual is displayed in a different color. The samples' names must be defined inside the "pedigrees" file.

The Allele View Family also allows the FAST-MAP user to:
(1) Look at current family's genotypes for another marker: click on the pop-up button containing the current marker's name, and select another marker;
(2) Look at the alleles, quantitation, and electropherogram plots of all members in the family: click on the second button and select the appropriate display item. For example, Figure D.12 shows the family's electropherograms being displayed;
(3) Go directly to another individual in the family: click on the pane in Allele View Family to select that individual as the current sample in all the displays.

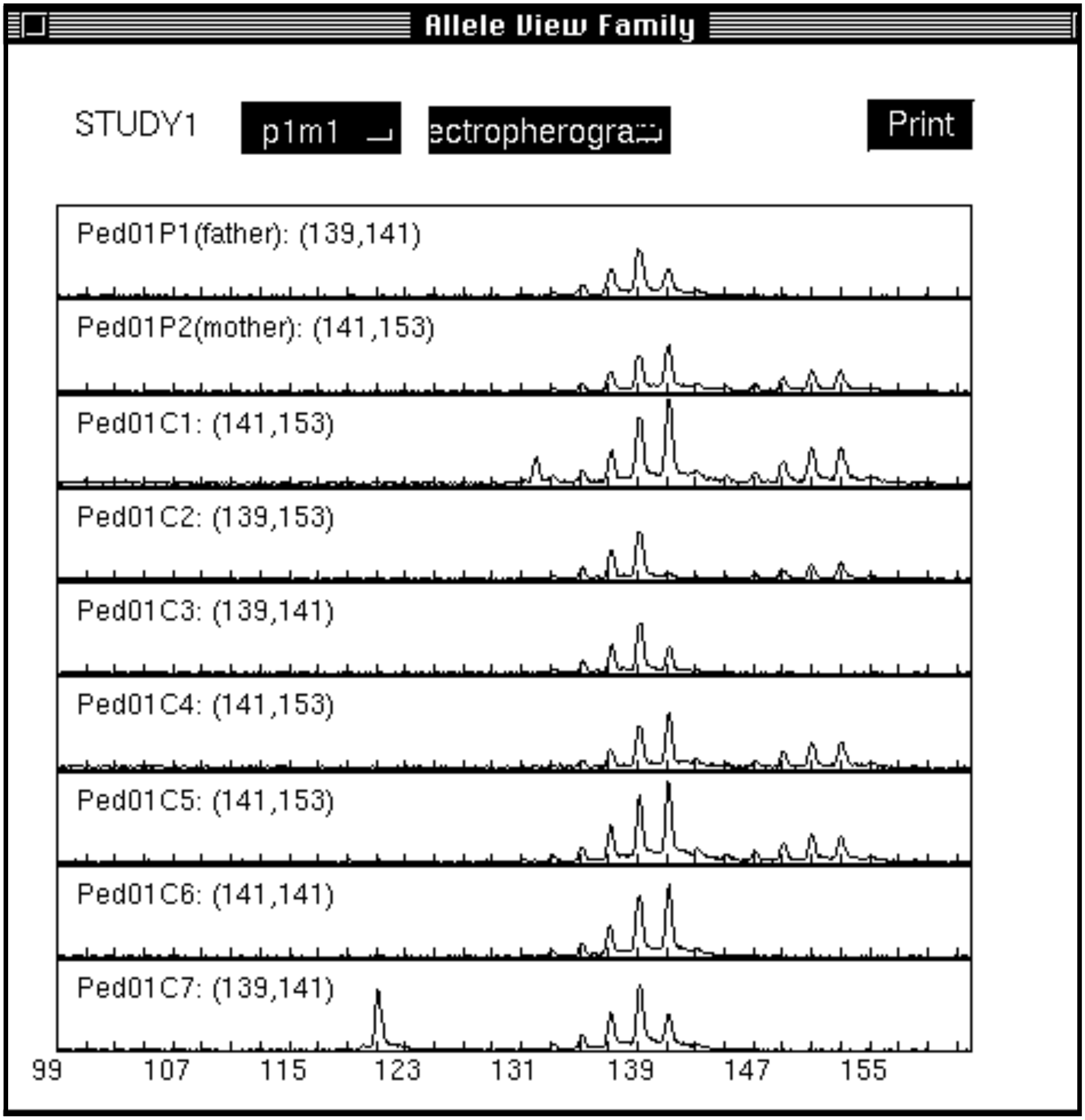


Figure 12. The "Allele View Family" specialized window: Electropherograms.

Viewing subsets of experiments

The *allele_view* program can be customized to view only some of the experiments. This reduction in data viewing (e.g., to only particularly suspect allele calls) can potentially reduce the amount of user operator time spent reviewing genotyping data. One particularly useful data subset is those genotypes that have been identified as inconsistent allele calls by a Mendelian inheritance checker. Although for greater automation we plan to eventually provide Mendelian inheritance checking functionality within FAST-MAP, these instructions assume (for now) that the inconsistency checking is done by some external computer program other than FAST-MAP.

To specify a list of genotype experiments to inspect and/or edit in *allele_view*, do the following:

Step 1: Create a tabbed-text file named `limited.view` and put it in the "<nickname>/input/" directory. The file must consist of at least two columns containing (1) the sample names, and (2) the marker names. For example, `limited.view` can be created by editing a tabbed-text output file from a Mendelian inheritance checker. The first line of `limited.view` is used as a header for indicating what information is contained in each column (for header keywords, use strings with no whitespace). The "sample" column must be labeled with the keyword `sample`, and the "marker" column must be labeled with the keyword `marker`. The sample and marker columns must be the first two columns, any other columns are extraneous for this application, and will be ignored by *allele_view*.

Alternatively, if the user wishes to type in the <sample, marker> pairs manually, then type, in FAST-MAP:

```
>> edit <nickname> limited_view
```

This will create a `limited.view` file in the "<nickname>/input/" directory for the user to edit.

Note that the file `limited.view` must reside in the "input/" directory of the gel or study that the user will be calling *allele_view* on. For example, if the user placed `limited.view` in STUDY1's "input/" directory, then *allele_view STUDY1* will display the limited list of experiments in the "Prioritized" mode. On the other hand, if the user placed `limited.view` in, say GEL2's "input/" directory, *allele_view STUDY1* will not display the limited subset even though GEL2 is a component gel of STUDY1. In this case, use *allele_view GEL2* instead to view the selected subset.

Step 2: In *allele_view*, view the selected subset of experiments inside under the "Prioritized" mode.

Step 3: The "Prioritized" mode of *allele_view* will always show the limited subset until the file `limited.view` is removed from the gel or study's "input/" directory, for example:

```
>> remove <nickname> limited_view
```

Viewing and comparing other allele calls

If the user has other allele calls and wishes to view and compare them with FAST-MAP's calls, a tabbed-text file called `other.results` can be created and placed in the gel or study's "input/" directory that will be *allele_call*ed(just as in the case with `limited.view`).

The format for `other.results` is very similar to the <nickname>.results files generated by FAST-MAP. The first line serves as the header line, in which the first 6 items (columns) must contain the following:

```
gel     lane     sample     marker     allele1     allele2
```

The columns must follow the above order. If there are more than 6 columns, the extraneous columns will be ignored.

Step 1: Create `other.results` which contains the allele calls by a different means, and put it in the "<nickname>/input/" directory. To type in the alleles manually, the user can use FAST-MAP's "edit" utility:

```
>> edit <nickname> other_results
```

This will create a `other.results` in <nickname>'s "input/" directory.

Step 2: In *allele_view*, view and edit any mismatches between FAST-MAP's calls and the allele calls in `other.results` under the "Prioritized" mode. Only those experiments where the allele calls disagree will be shown.

Step 3: To resume to the normal mode of *allele_view*, remove the file `other.results` from the gel or study's "input/" directory, for example:

```
>> remove <nickname> other_results
```

D.5. Utility programs

D.5.1. setup

The *setup* program is a script for guiding the user through the steps of setting various FAST-MAP objects for analysis. Currently, *setup* is programmed to help user in setting up a gel, a study, a shared matrix file, as well as a shared sample file.

Alternatively, the user may set up the data manually.

Program operation

The model is as follows:

- The user runs one or more gels on a DNA sequencer and transfers all the data files (gel files, sample files, matrix files) into a centralized "drop box" in:
  "FAST-MAP/incoming/"
- The user, or another user in charge of running FAST-MAP, then runs the *setup* script which will ask a series of questions about the gel or study, automatically create the gel or study directories, and transfer the data files from "FAST-MAP/incoming/" to the respective "input /" directories.

Note that to run the *setup* script, all the requisite data files must be transferred into the centralized "FAST-MAP/incoming/" directory, not into the individual "<nickname>/input/" directory. In fact, the user does not even need to manually create the directories, as *setup* will automatically create them and move the data files from "FAST-MAP/incoming/" into the correct places.

To run the *setup* script, the user types the command:

```
>> setup <type> [<nickname>]
```

where <type> can be one of the following: gel, study, matrix, and sample; and <nickname> (optional) is the nickname of the object to be created.

e.g.

```
>> setup gel GEL1
>> setup study STUDY1
>> setup matrix MATR211
>> setup sample SAMP211
```

Here are the data files that must be transferred into "FAST-MAP/incoming/" before running *setup*:

- To set up a gel:
  - the gel file;
  - the matrix file (if necessary); and
  - the sample file (if necessary/available).
- To set up a study:
  - none; except that the user must set up the component gels first.
- To set up a shared matrix file:
  - the matrix file.
- To set up a shared sample file:
  - the sample file.

Example run: gel

Suppose we have already transferred the gel file "R211c.gel" and the sample file "R211c.samples" into the "FAST-MAP/incoming/" directory, and that we will be re-using a shared matrix file MATR211 that we have set up previously (thus, we do not need a copy of the matrix file again). Let's run *setup* without supplying a nickname:

```
>> setup gel

What is the nickname for the new gel?
```

FAST-MAP prompts for the nickname, and we type in GEL3. The next question is:

```
Have you defined the panels, markers, and size_stds used in "GEL3"?
Y/N [N]:
```

If we haven't, type 'N' or hit return (the default reply is in the square bracket) and *setup* will advise:

```
Please use "edit <userfile>" to define the panels, markers, and
size_stds used in gel "GEL3" first.
```

Once we have defined the panels, markers and size standards using *edit*, we can call *setup* again. Let's suppose that we have already provided the requisite information and we type 'Y' instead:

```
Step 1: Define "GEL3" as a nickname of type "gel".
        Please add "GEL3" to 'nicknames' under type "gel" now.
```

The "nicknames" file is automatically opened by an editor for the user to add "GEL3" as a "gel". Note that the "pathname" for "GEL3" is what the gel directory will be, not where the data files are currently. That is, do not type in the path "FAST-MAP/incoming/" for GEL3.

When we have done editing the "nicknames" file, FAST-MAP creates the desired gel directory that we have just specified in "nicknames", and proceeds to the second step:

```
Reloading nicknames...Done.

Step 2: Define the "settings" for gel "GEL3".
        Please fill in the 'settings' for GEL3 now.

Creating default user file "settings"...Done.
```

This time, the editor opens up a default "settings" file that FAST-MAP has created for GEL3. We fill in Section 1 to provide the gel-specific settings. In particular, we put in "R211c.gel" as the `Gel_file_name`, "MATR211" as the `Matrix_file_name`, and "R211c.samples" as the `Sample_file_name`. Don't forget to fill in the other attributes in section 1, especially `Size_standard` and `Panel_name`. When done, FAST-MAP responds as follows:

```
Reloading settings...Done.
Moving gel file "R211c.gel" from "incoming" to
"/afs/cs/project/genome/demo/gels/Gel3/input/"...Done.
```

```
Moving sample sheet "R211c.samples" from "incoming" to
"/afs/cs/project/genome/demo/gels/Gel3/input/"...Done.

Step 3: Define the 'layout' for gel "GEL3".
        Please fill in the 'layout' for GEL3 now.

Creating default user file "layout"...Reading ABI sample file
R211c.samples.........Done.
Creating GEL3's layout from R211c.samples......................
.............Done.
```

The automatically-created "layout" file for GEL3 is presented in our editor of choice. When done inspecting, the set-up is complete:

```
Reloading layout...Done.

Thank you.
```

In the event that the user has to exit *setup* before the whole procedure is completed successfully, the setup process can always be resumed by calling *setup* again:

```
>> setup gel GEL3
```

and then answer the questions accordingly.

Example run: study

Let us set up a study STUDY2 which consists of three gels with nicknames GEL1, GEL2, and GEL3. First, we transfer all the gel files, matrix files, and sample files in the study into "FAST-MAP/incoming/". Then, in FAST-MAP, type:

```
>> setup study STUDY2
```

```
Have you defined the panels, markers, and size_stds
used in study "STUDY2"? Y/N [N]:
```

If 'N', we can use "edit <userfile>" define the panels, markers, and size standards used in the study. Let's assume that we have already done so. We type 'Y':

```
Step 1: Define "STUDY2" as a nickname of type "study".
------
```

```
- Please add "STUDY2" to 'nicknames' under type "study".
- Please also add *all* the component gels of STUDY2
  to 'nicknames' under type "gel".
```

We define STUDY2 as a study in "nicknames", taking care to define the "pathname" for STUDY2 as the desired study directory to be created by *setup*. At this point, we also need to define all the component gels of STUDY2, namely GEL1, GEL2 and GEL3 in "nicknames". Let us assumed that we have already done so.

```
Reloading nicknames...Done.

Step 2: Define the "settings" for study "STUDY2".
------  Please fill in the 'settings' for STUDY2 now.

Creating default user file "settings"...Done.
```

The default settings for STUDY2 need not be changed, so we exit the editor and FAST-MAP proceeds to the next step:

```
Reloading settings...Done.

Step 3: Define the 'layout' for study "STUDY2".
------  Please fill in the 'layout' for STUDY2 now.

Creating default user file "layout"...Done.
```

We type in GEL1, GEL2, and GEL3 in the "layout" file, save it, and close the editor:

```
You seemed to have already set up all 3 gels of STUDY2.  Stop? Y/N
[N]:
```

FAST-MAP detected that we have already set up GEL1, GEL2, and GEL3 previously, and we type "Y". If not, we will be automatically guided to set up each of the gel as before.

We have thus completed the setting up of the study STUDY3, and any FAST-MAP program can be invoked on STUDY3.

Example run: matrix

In this example, we show how to use *setup* to set up a shared matrix file R212.matrix under the nickname MATR212. First, we transfer R212.matrix into the "FAST-MAP/incoming/", and then type:

```
>> setup matrix MATR212

Step 1: Define "MATR212" as a nickname of type "matrix".
        Please add "MATR212" to 'nicknames' under type "matrix".
```

The first step is to define MATR212 in "nicknames" under type "matrix". In the "pathname" column, we tell *setup* where to put the matrix file in, not where it is currently at (that is, not "FAST-MAP/incoming/R212.matrix"). Since matrix is a file and not a directory, we type in the full path of where the matrix file's destination, including the file-name, such as:

```
nickname   pathname                                        type
MATR22     /afs/cs/project/genome/demo/matrices/R212.matrix   matrix
```

When done, FAST-MAP automatically moves R212.matrix from "FAST-MAP/incoming/" into the destination as specified in "nicknames".

```
Reloading nicknames...Done.
Moving matrix file "R212.matrix" from "incoming" to
"/afs/cs/project/genome/demo/matrices/"...Done.
```

We may now use MATR212 as a shared matrix file in setting up gels which use the matrix file "R212.matrix".

Example run: sample

In some high-throughput study designs, a same set of DNA samples may be used for multiple gels while varying the panels for PCR amplification in each gel. In this case, the gels may share the same sample file. Instead of making a copy of the sample file inside the "input/" directories of each of the gels, we can set it up as a shared sample file in "nicknames".

Let us use *setup* to set up a shared sample file "R212.samples" under the nickname SAMP212:

```
>> setup sample SAMP212

Step 1: Define "SAMP212" as a nickname of type "sample".
        Please add "SAMP212" to 'nicknames' under type "sample".
```

As with shared matrix files, the first (and only) step is to define SAMP212 in "nicknames" as a "sample". Again, type in the full path of where we want the sample file to be, and ending with the file-name (e.g. "/afs/cs/project/genome/demo/samples/R212.samples").

When done, FAST-MAP automatically moves R212.samples into the destination as specified in "nicknames":

```
Reloading nicknames...Done.
Moving sample file "R212.samples" from "incoming" to
"/afs/cs/project/genome/demo/samples/"...Done.
```

We may now use SAMP212 as a shared sample file in setting up those gels which use the sample file "R212.samples".

D.5.2. allele_results

The *allele_results* program generates customized formatted output files. The "<nickname>.results" file are generated in the "<nickname>/output/" directory. These results file are in tabbed-text format and can be opened for post-editing by most spreadsheet programs.

Program operation

The program *allele_results* is automatically invoked at the end of *allele_call* and *allele_view* to generate the results file for the gels and studies. However, user may invoke *allele_results* to create a results file with a different format. First, edit the "settings" file to specify the format:

```
>> edit <nickname> settings
```

Change the *allele_results* settings in Section 4:

```
Output_noise_genotypes  no
Output_sort_by          markers
Latest_sample_only      no
```

- `Output_noise_genotypes`: If "yes", all genotypes will be reported. If "no", FAST-MAP will output (0, 0) as the genotype for experiments that it considered to be noise only, even though it may have attempted to genotype them (that is, when "Analyze_noisy_data" was set to "yes").

- `Output_sort_by`: If "markers", then genotypes of the same marker are grouped together. If "samples", then genotypes of the same sample are reported together (e.g. for studying haplotypes).

- `Latest_sample_only`: If "yes", only the latest copy of any duplicate samples is reported. In FAST-MAP, a gel that appears later in the study's "layout" is considered to be newer. If more than one copy of the sample has been run on a gel, then the one with the largest lane number is considered latest. If "no", then all copies of the sample are reported in the results file.

After defining a new results format, invoke the *allele_results* program to generate new results files:

```
>> allele_results <nickname>
```

To compile new results files for more than one gel or study (up to 45 of them) in a single command:

```
>> allele_results <nickname1> <nickname2> ...
```

Note that (1) the old results files will be overwritten; and (2) there is no need to run *allele_call* again to generate the new results files.

Example run

First, edit the gel or study's "settings" file to change the settings for *allele_results*. For example, let's say we want the output to be sorted by "samples" instead of "markers".

Then, invoke the *allele_results* program to generate new results files:

```
>> allele_results <nickname>
```

For example,

```
>> allele_results STUDY1
```

The computer responds by:

```
FAST-MAP v1.04b (created April 21, 1997, last revised 04/21/97,
system HP700).

[Session started at 19-Apr-97 15:40:31.]

STUDY: STUDY1
=====

Compiling results for STUDY1.......Done.
Compiling results for GEL1.......Done.
Compiling results for GEL2.......Done.

[Session for "allele_results STUDY1" ended at 19-Apr-97 15:40:42.]
```

And here is an excerpt of STUDY1.results:

```
>> inspect STUDY1 results

gel  lane sample  marker allele1 allele2 comput1 comput2 qual
GEL1 1    Ped01P1 p1m1   139     141     139     141     0.961
GEL1 1    Ped01P1 p1m2   336     336     336     336     0.977
GEL1 1    Ped01P1 p1m3   190     190     190     190     0.789
GEL1 1    Ped01P1 p1m4   301     301     301     301     0.963
GEL1 1    Ped01P1 p1m5   208     208     208     208     0.749
GEL1 2    Ped01P2 p1m1   141     153     141     153     0.947
GEL1 2    Ped01P2 p1m2   334     346     334     346     0.971
GEL1 2    Ped01P2 p1m3   190     190     190     190     0.929
GEL1 2    Ped01P2 p1m4   301     309     301     309     0.963
GEL1 2    Ped01P2 p1m5   192     210     192     210     0.874
GEL1 3    Ped01C1 p1m1   141     153     141     153     0.941
  :  :       :       :     :        :       :       :        :
```

D.5.3. allele_printout

The *allele_printout* program generates hardcopies of electropherograms, quantitation, and genotypes for individual genotyping experiments from the graphical Allele View windows. The pictures of each marker are collated and output in a single postscript file "<marker>.ps" stored in the "/output/" directory. Users may customize the layout of the hardcopies by changing the settings within "Section 7: Allele_printout settings" in the "settings" file.

Program operation

Under "Section 7: Allele_printout settings" in the "settings" file, the user can customize the layout of the hardcopies:

```
Send_to_printer                 yes
Show_figure                     no
Rows_per_page                   4
Columns_per_page                2
Include_electro_plots           no
```

- If "Send_to_printer" is set to "yes", allele_hardcopy will send the hardcopies to the printer for immediate printing. Otherwise, a file called "<marker>.ps" for each marker will be kept in the gel's output directory for user to print at a later time. (Note that this file will not be present if it has been automatically printed.)

- If "Show_figure" is set to "yes", the figures are displayed on the computer dynamically as the hardcopies are generated. This may be a good (and way cool) way to display the pre-compiled works of FAST-MAP.

- The options "Rows_per_page" and "Columns_per_page" define the number of windows to be printed row-wise and column-wise in a page. Changing these settings will customize the layout of the hardcopies.

FAST-MAP's analysis detects the gel subregion where the genotype data resides, which sets the limited view of the detected bands; FAST-MAP displays and prints this limited data view. To display a long-range electropherogram view spanning the *entire* allele size range (as defined for the marker), set the option "`Include_electro_plots`" to "`yes`".

In addition to these settings, *allele_printout* also respects the system printing options, as defined in the "preferences" file. These options are "Printer_name", "Printer_type", and "Print_orientation".

Example run

The *allele_printout* program is invoked explicitly by the user by typing:

```
>> allele_printout <nickname>
```

or

```
>> allele_printout <nickname1> <nickname2> ...
```

e.g.

```
>> allele_printout GEL3
```

The program responds with the following messages (for the case when "`Send_to_printer`" is set to "`no`"):

```
GEL3: allele_printout
----
Saving electropherograms for p1m1.......Done.
Gel GEL3's marker p1m1's electropherograms have been saved in
/afs/cs/project/genome/demo/gels/Gel3/output/p1m1.ps to be printed by
user later.

Saving electropherograms for p1m2.......Done.
Gel GEL3's marker p1m1's electropherograms have been saved in
/afs/cs/project/genome/demo/gels/Gel3/output/p1m2.ps to be printed by
user later.

....

---------------------------------------------
Plots for the following gels can be found in their "output/"
directories in the form of <marker_name>.ps to be printed later:
    GEL3

---------------------------------------------
```

WARNING: Even with multiple windows per page, a large number of pages may be generated. To avoid wasting paper or disk space, this function should be used only when absolutely necessary, and delete the "`<marker>.ps`" files when done. For printing hardcopies of individual genotypes, refer to the section on *allele_view*.

D.5.4. size_std_view

New size standards may be used that are not included in the FAST-MAP library. In this case, FAST-MAP may have no prior binning information for these size standards. Although FAST-MAP will try to automatically construct new binning information, a less-than-perfect gel image may lead to incomplete binning. When this happens, the user can help by indicating the location of the size standard peaks for a few lanes on the first gel image. BY PERFORMING THIS CALIBRATION FOR A SIZE STANDARD SET JUST ONCE, FAST-MAP CAN TAKE RESPONSIBILITY THEREAFTER FOR AUTOMATED SIZING AND LANE TRACKING.

The *size_std_view* program is the graphical interface that enables the user to perform this one-time calibration. The interface is similar to the *image_view* interface in many ways.

Program operation

To use *size_std_view* the user types the command

```
>> size_std_view <nickname>
```

e.g.

```
>> size_std_view GEL1
```

A "Size_Std View" window is opened, showing the size plane of the gel. In this window, the user performs the following:
(1) Specify start and end sizes of the size standard;
(2) Specify a lane, and click on *all* the size standard bands on that lane; and
(3) If desired, repeat Step 2 to specify other lanes. Usually a couple of lanes is sufficient to help FAST-MAP create an initial binning library for the new size standard.

When done, click "Close" and answer "yes" when asked whether to save the size points.

Example run

Let us invoke *size_std_view* on GEL1:

```
>> size_std_view GEL1
```

FAST-MAP responds with the following message:

```
Initializing gel settings for GEL1.....Done.
Reading image data for dye 4...Done.
```

A "Size_Std View" window is created, displaying the size plane of GEL1.

Size_std View window

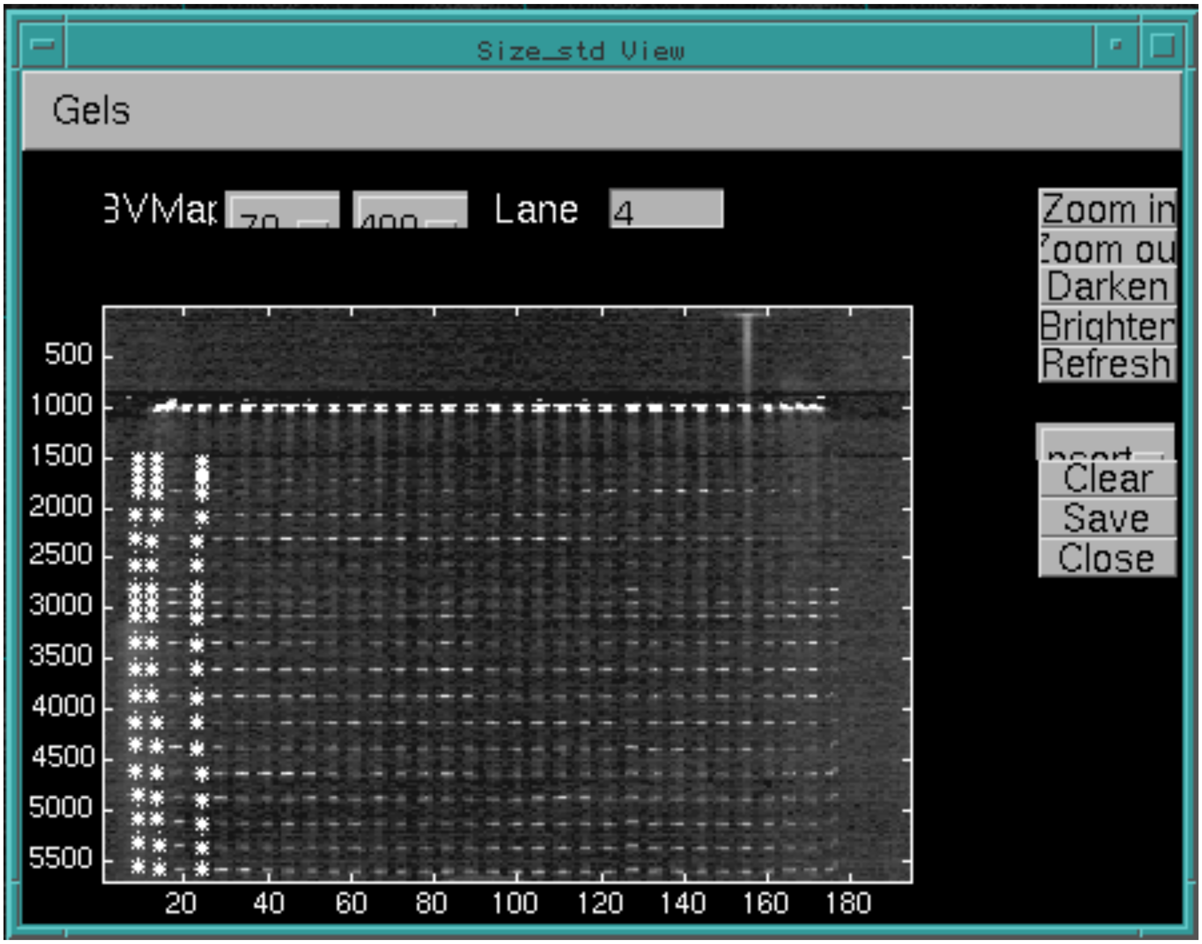


Figure D.13 *Size_std View* window.

There is only one menu item for "Size_std View": "Gels". If the user has invoked *size_std_view* on a study, this menu allows the user to go to another gel in the study.

The size standard name is displayed in the area above the gel image. There are two buttons next to the name: the first defines the minimum size standard on the gel, the second defines the maximum size standard. To change, click on the buttons and select the appropriate size listed in the pop-up menu. To the right, there is a text box labeled "Lane" for the user to specify the current lane that is defined.

Buttons for controlling the display are to the right of the gel image:

- The "Zoom in" and "Zoom out" allows zooming operations on the gel image. To zoom in, click on the "Zoom in" button and then select a region on the gel image. It is alright to include areas that are outside the gel image if the size standard bands are very close to the edges. To zoom out, simply click on the "Zoom out" button and it will return to the previous state.
- The "Darken" and "Brighten" buttons allow user to control the image contrast.
- The "Refresh" button is for user to re-display the current user edits on the image.
- The "Insert/Delete" pop-up button allows user to select the mode (insertion or deletion) of action on a size standard peak for each mouse click.
- The "Clear" button deletes all current input.
- The "Save" button saves all current input.
- The "Close" button asks the user whether to save the current input, and then exit by closing the "Size_std View" window.

Here are the steps to calibrate a new size standard using *size_std_view*:

<u>Step 1:</u> Define the minimum and maximum size standards using the two buttons next to the size standard name;

<u>Step 2</u>: Specify a lane in the "Lane" box. Click on <u>all</u> the size standard bands on the lane. The default mouse click action is "Insert". To see the points, click on "Refresh". To delete a point, select "Delete pts" from the "Insert/Delete" pop-up button, and each subsequent mouse-click will delete a nearest input point. If "Delete lane" is selected, each mouse click deletes a nearest lane altogether. Re-select "Insert" after deleting. When done, click on "Save".

<u>Step 3</u>: Repeat Step 2 with another lane if desired. Usually, a couple of lanes is sufficient to get FAST-MAP started.

<u>Step 4</u>: Click on the "Close" button to exit the program.

D.5.6. edit

FAST-MAP knows where the FAST-MAP "/user/" information files are. And, via the nickname mechanism, FAST-MAP can also determine where the gel "/input/" data and helper files are. The FAST-MAP "edit" makes use of such knowledge to provide rapid accessing and editing of these files.

The *edit* utility allows the user to create and modify user files within Matlab. Without having to remember and type in the full pathnames, the user can edit the user files by typing one of the following:

```
>> edit preferences
>> edit nicknames
>> edit panels
>> edit markers
>> edit size_stds
>> edit dyes
>> edit pedigrees
```

The user can also edit gel or study-specific input files by typing:

```
>> edit <nickname> settings
>> edit <nickname> layout
>> edit <nickname> limited_view
>> edit <nickname> other_results
```

e.g.

```
>> edit GEL3 settings
>> edit STUDY1 layout
```

If the input files are not already in the "/<gel>/input/" subdirectory, default input files will be created automatically in the subdirectory for the user to edit.

If desired, the user can also edit output files such as the results file by typing:

```
>> edit <nickname> results
```

e.g.

```
>> edit STUDY1 results
```

However, keep in mind that FAST-MAP will not detect changes made to the results file inside the editor. Instead, the proper way to edit genotypes involving FAST-MAP is to use the *allele_view* program.

The user must specify the preferred editor in the "preferences" file. The default editor is *emacs*. FAST-MAP prefers an editor that is not executed in the background: if the editor is executed in the background, the user must call the function *reload* explicitly after editing the user library files to reload the new data, or use the command *ed* instead of *edit*.

To save an edited file in Emacs, type Ctrl-X Ctrl-S.

To exit the Emacs editor, type Ctrl-X Ctrl-C.

To see a list of files that can be edited using *edit* , type

```
>> help edit
```

D.5.7. ed

This is for user whose preferred editor is executed in the background. All users of the Macintosh version of FAST-MAP must use *ed* instead of *edit* .

The program *ed* has the same functionality as *edit*. See "edit".

D.5.8. inspect

The *inspect* utility allows the user to review user files within Matlab. Like the *edit* program, it provides rapid file access and alleviates the user from having to remember and type in long pathnames. For example, the user can review user files by typing one of the following:

```
>> inspect preferences
>> inspect nicknames
>> inspect panels
>> inspect markers
>> inspect size_stds
>> inspect dyes
>> inspect pedigrees
```

To go to the next page in "inspect", hit the SPACE bar once. To exit "inspect" at any time, type "q".

The user can also review gel- and study-specific input files by typing:

```
>> inspect <nickname> settings
>> inspect <nickname> layout
```

e.g.

```
>> inspect GEL1 settings
>> inspect STUDY1 settings
```

To review the genotyping results file, type:

```
>> inspect <nickname> results
```

To review the summarized report file, type:

```
>> inspect <nickname> report
```

To review the status files and log files generated by *image_call* ,*allele_call* or any other major FAST-MAP programs, type:

```
>> inspect <nickname> status <program>
>> inspect <nickname> logfile <program>
```

If the user omits the program name, FAST-MAP assumes <program> to be "allele_call". For e.g.

```
>> inspect STUDY1 status
>> inspect GEL1 logfile image_call
```

D.5.9. help

This FAST-MAP utility provides on-line documentation. It is a fast way for the user to get familiarized with the programs of FAST-MAP. For example, typing

```
>> help FAST-MAP
```

prints the following message to the screen:

```
Basic commands in FAST-MAP:
```

```
To setup gels/studies   : setup

To analyze gels/studies :

 - Fully automated mode : allele_call, allele_view
 - Cautious mode        : image_call, image_view,
                          allele_call, allele_view
 - Extra cautious mode  : prep_call, prep_view,
                          image_call, image_view, marker_view
                          allele_call, allele_view

To edit or view files   : edit, inspect

To reset environment    : reload

Misc useful commands    : allele_results, allele_printout,
                          add_pedigrees, reset_now, remove,
                          size_std_view, display, done,
                          check_version

To get more info on any of the above commands, type:
       help <command>
For example, to get this help message again, type:
       help FAST-MAP
```

To see the on-line documentation of the various FAST-MAP programs, the user types:

```
>> help <program>
```

Here is an example:

```
>> help allele_call
```

prints the following on the screen:

```
Automatic lane-tracking, quantitation, and genotyping of
gel experiments.

Usage: allele_call NICKNAME_1 NICKNAME_2 ... (up to NICKNAME_45)

     - <NICKNAME> can be a gel or a study;

     - When done, please use "allele_view" to view the alleles called.


See also allele_view, image_call, image_view.
```

D.5.10. check_version

To check which version of FAST-MAP that is currently running, the user can type

```
>> check_version
```

The computer prints out the following message banner:

```
**************************************

        FAST-MAP Version 1.04b

             BETA Release

          April 21, 1997

       (Last revised 04/21/97)

**************************************


ans =

1.04b
```

D.5.11. update_settings

For users of old versions of FAST-MAP, updating to a new version may mean that the "settings" file for their numerous gel may become outdated. In view of this, FAST-MAP provides an automatic update function just to handle gels and studies' "settings" file. For example, the user can update an old "settings" file for a gel that has been set up with an older version of FAST-MAP by typing:

```
>> update_settings <nickname>
```

Here is an example:

```
>> update_settings GEL3

GEL GEL3:
---
 - Saved the original "settings" file in "settings.BAK".
 - Created a brand new "settings" file for GEL3.
 - Transferred these gel-specific settings for GEL3:
          Gel_file_name
          Matrix_file_name
          Sample_file_name
          Sequencer_type
          Number_of_lanes
          Size_standard
          Min_size_standard
          Max_size_standard
          Panel_name
          Experiment_condition
```

```
           Noise_threshold

  *** Done updating the "settings" file for GEL3. ***
```

Note that *update_settings* can only process a single gel or study at a time. Also, *only* the gel- and study-specific settings (namely, the values for those attributes in "Section 1" of the "settings" file) are carried over from the old "settings" into the new "settings" file. The rest of the attributes are set with default values. A copy of the old "settings" file is saved as "settings.BAK" for reference if necessary.

To update the "settings" file of a study as well as those for the component gels, type:

```
>> update_settings <study>
```

For example:

```
>> update_settings STUDY1


STUDY STUDY1:
-----
 - Saved the original "settings" file in "settings.BAK".
 - Created a brand new "settings" file for STUDY1.
 - Transferred these study-specific settings for STUDY1:
          Use_study_settings

*** Done updating the "settings" file for STUDY1. ***

Proceed to update "settings" files for the 2 component gels of
STUDY1? Y/N [Y]:

GEL GEL1:
---
 - Saved the original "settings" file in "settings.BAK".
 - Created a brand new "settings" file for GEL1.
 - Transferred these gel-specific settings for GEL1:
           Gel_file_name
           Matrix_file_name
                :
           Noise_threshold

*** Done updating the "settings" file for GEL1. ***

GEL GEL2:
---
 - Saved the original "settings" file in "settings.BAK".
 - Created a brand new "settings" file for GEL2.
 - Transferred these gel-specific settings for GEL2:
           Gel_file_name
           Matrix_file_name
                :
           Noise_threshold
```

```
*** Done updating the "settings" file for GEL2. ***
```

D.5.12. print_now

This function is for FAST-MAP UNIX users to print various FAST-MAP files to the printer defined in "preferences". Like the *edit* and *inspect* programs, it provides rapid file access so that the user does not have to remember and type in long pathnames. For example, the user can print any of the user files by typing :

```
>> print_now <nickname> settings
>> print_now <nickname> layout
>> print_now <nickname> results
>> print_now <nickname> report
>> print_now <nickname> other_results
>> print_now <nickname> limited_view
>> print_now <nickname> logfile <program>
>> print_now <nickname> status <program>
>> print_now nicknames
>> print_now panels
>> print_now markers
>> print_now size_stds
>> print_now dyes
>> print_now preferences
>> print_now pedigrees
```

Users of Macintosh FAST-MAP should always use the "Print" menu command under the "File" menu instead.

D.5.13. reload

The function *reload* loads the various user files and input files (see "Annotating Genotyping Data") into the current working environment. Under normal circumstances, there is no need to invoke this function directly, unless the *edit* command was executed in background, or the working environment appears to be corrupted. In the latter case, though, we advise quitting FAST-MAP and restart again instead.

Reloading FAST-MAP user files

When invoked with no arguments, this function reloads all the user files from "FAST-MAP/user/" into the FAST-MAP environment:

```
>> reload
```

To reload a particular user file, type:

```
>> reload <user_file>
```

e.g.

```
>> reload pedigrees
```

Reloading FAST-MAP input files

To reload the user specifications in the input files of a gel or study, type:

```
>> reload <nickname>
```

e.g.

```
>> reload STUDY1
```

To reload a particular input file, type:

```
>> reload <nickname> settings
>> reload <nickname> layout
```

D.5.14. reset_now

This DESTRUCTIVE function deletes previously compiled results in order to provide a clean start for FAST-MAP on a gel, study, marker panel, or size standard. This function should be used with EXTREME CAUTION, since the previously computed results will be lost.

Resetting a gel

To delete all previously computed results for a gel, type:

```
>> reset_now <gel>
```

e.g.

```
>> reset_now GEL3
```

This function deletes the "display/" and "output/" directories inside the gel directory.

Resetting a study

To delete all previously computed results for a study and its component gels, type:

```
>> reset_now <study>
```

e.g.

```
>> reset_now STUDY1
```

This function deletes the "display/" and "output/" directories inside the study directory and the component gels' directories.

Resetting a marker panel

To delete the previously compiled binning libraries and stutter libraries for all the markers in a panel, type:

```
>> reset_now <panel>
```

e.g.

```
>> reset_now PANEL1
```

This function deletes all binning and stutter library files for the markers in the named panels. To delete the libraries of a marker panel under a particular experimental condition, type:

```
>> reset_now <panel> <gel_with_experiment_condition>
```

e.g.

```
>> reset_now PANEL1 GEL3
```

This will delete all the marker libraries of PANEL1 under the "Experiment_condition" of GEL3 (see "settings: gel"). If no gel name is supplied, the marker libraries with no special "Experiment_condition" are deleted by default.

Resetting a size standard

To delete the previously compiled binning libraries for a size standard, type:

```
>> reset_now <size_std>
```

e.g.

```
>> reset_now GS500
```

This function deletes all the library files for the named size standard. To delete the libraries of a size standard under a particular experimental condition, type:

```
>> reset_now <size_std> <gel_with_experiment_cond>
```

e.g.

```
>> reset_now BVMap GEL3
```

This will delete the BVMap libraries under the "Experiment_condition" of GEL3 (see "settings : gel"). If no gel name is supplied, the size standard libraries with no special "Experiment_condition" are deleted by default.

D.5.15. remove

This function allows the user to delete various FAST-MAP files without having to remember and type in long pathnames. For example, the user can delete user files by typing one of the following:

```
>> remove <nickname> settings
>> remove <nickname> layout
>> remove <nickname> results
>> remove <nickname> report
>> remove <nickname> limited_view
>> remove <nickname> other_results
```

```
>> remove <nickname> logfile <program>
>> remove <nickname> status <program>
>> remove nicknames
>> remove panels
>> remove markers
>> remove size_stds
>> remove dyes
>> remove preferences
>> remove pedigrees
```

D.5.16. display

This utility function is for the most technically inclined user to view the various library objects that FAST-MAP has compiled over time.

Viewing the stutter library

To view the stutter library of a marker, type

```
>> display libA <marker>
```

e.g.

```
>> display libA p1m1
```

To view the stutter library of a marker under a particular "Experiment_condition" (see "settings: gel"), include a <gel> after <marker>:

```
>> display libA <marker> <gel_with_experiment_condition>
```

Two windows are displayed: a "Stutter Matrix A" window which shows the stutter library (we call it "libA"), and a "Relative Amplifications" window showing the range of amplification ratios for different allelic differences.

Viewing the stutter patterns

To view the stutter patterns of a marker, type:

```
>> display stutter <marker> <allele1> <allele2> ...
```

Up to 5 alleles are allowed in a single display.

For example,

```
>> display stutter p1m1 121 131 141 151 161
```

For viewing markers under a particular "Experiment_condition", include a <gel> with that "Experiment_condition" after <marker>:

```
>> display stutter <marker> <gel> <allele1> <allele2>...
```

An "Allele Stutters" window is created, showing the stutter patterns of the various alleles for that marker.

Viewing the marker binning

To view the binning of a marker, type:

```
>> display binning <marker>
```

For example,

```
>> display binning p1m1
```

An "Allele Binning" window is created, showing the binning of the alleles for the named marker.

To view the binning of a marker under a particular "Experiment_condition", include a <gel> after <marker>:

```
>> display binning <marker>
<gel_with_experiment_condition>
```

Viewing the size standard binning

To view the binning of a size standard, type:

```
>> display binning <size_std>
```

For example,

```
>> display binning BVMap
```

A "Size Stds Binning" window is created, showing the binning of the alleles for the named size standard.

To view the binning of a size standard under a particular "Experiment_condition", include a <gel> after <size_std>:

```
>> display binning <size_std>
<gel_with_experiment_condition>
```

D.5.17. set_stutter

FAST-MAP's allele calling algorithms are based on stutter deconvolution. If the user wishes to call the alleles without using stutter deconvolution, type:

```
>> set_stutter
```

And FAST-MAP will prompt:

```
Set to stutterless system? Y/N [N]
```

By answering "Y", the stutter handling functionalities of FAST-MAP will be turned off in the subsequent analysis. The user should use this option only under very special conditions, for example, when analyzing data in which there are no stuttering effects (typically microsatellites with a long repeat unit, such as tri-nucleotides and tetra-nucleotides under certain experiment conditions).

D.5.18. recover

When the user edits the manifold, copies are saved in both the "display/" and the "input/SAVED_EDITS" directories. These edited files in the "display/" directory are read in <u>once</u>, and deleted from the "display/" directory immediately afterwards. Therefore, if the user calls *image_call* on a gel twice, FAST-MAP will read in any user edited manifold in the first *image_call*, but not in the second time around. To restore the user edited manifold (created previously in *image_view*'s "Repair" mode) back into the working "display/" directory again, type:

```
>> recover <nickname> mani_edited
```

If restoring a previous sizing grid boundary file (created in *image_view*'s "Draw" mode), type:

```
>> recover <nickname> mani_bounds
```

Then, the next *image_call* on <nickname> will be able to re-use the user edited manifolds.

D.5.19. done

To exit from FAST-MAP and Matlab, type

```
>> done
```

FAST-MAP responds by

```
Thank you for using FAST-MAP.
Good-bye...
```

and exits Matlab.

Glossary

Adenine (A): A *nitrogenous base* that is a member of the *base pair* A-T (*adenine* and *thymine*).

Alleles: Alternative forms of a *gene* or *marker*. Alleles are inherited separately from each parent, as the *chromosomes* from the sperm and the egg pair up in reproduction.

Autosomes: *Chromosomes* that are not involved in sex determination. In human beings, each individual has two copies of each autosome (chromosomes 1 to 22), one inherited from each parent. See also *sex chromosomes.*

Base pair (bp): Two complementary *nitrogenous bases* held together by weak bonds. The two strands of the *DNA* double helix are held together by the bonds between base pairs *adenine* and *thymine* or *guanine* and *cytosine*. The length of a DNA sequence is often given in *bp*.

Candidate gene: A gene with known or suspected biological functions which may be involved in a genetic disorder.

CA-repeats: A *microsatellite* marker that contains tandem repeats of the simple DNA sequence C-A (*cytosine* and *adenine*).

cDNA: See *complementary DNA.*

Centimorgan (cM): A unit of measure of recombination distance. A recombination distance of 1 cM is equal to a 1% chance that a *marker* at one genetic location will be separated from a marker at another location due to *meiotic recombination*. In humans, 1 cM is roughly equal to *1 Mb* (one million *base pairs*).

Chromosomes: The threadlike structures found in a cell containing the cellular *DNA* that encode the information of heredity. *Genes* are arranged in linear order along the chromosomes. The human *genome* consists of twenty-three pairs of matching chromosomes; each chromosome in every pair is inherited separately from each parent.

Clone: A group of genetically identical cells derived from a single precursor.

cM: See *centimorgan.*

Complementary DNA: *DNA* molecules synthesized by *reverse transcriptase* from an *RNA* template.

Complex disease: See *non-Mendelian disease.*

Crossing over: The process during meiosis in which an exchange occur between two corresponding *chromosomes* before one member of the chromosome pair is incorporated into a sperm or an egg. This process results in a reshuffling of *genes*; the further apart two genes are, the more likely it is for them to be reshuffled. This likelihood is used as a measure of how far apart two genes are, and is termed *recombination distance* with *centimorgans* as units.

Cytosine (C): A *nitrogenous base* that is a member of the *base pair* G-C (*guanne* and *cytosine*).

DNA (deoxyribonucleic acid): The molecule that encodes hereditary information. It is a double-stranded molecule held together by weak bonds between *base pairs* of *nucleotides*.

DNA sequence: The linear order of b*ase pairs* along a *DNA* molecule.

Electrophoresis: A technique for separating large molecules (e.g. *DNA*) in a mixture. A medium (e.g. a gel) containing the mixture is exposed to an electric field. Different molecules travel at different rates, depending on their electrical charges and sizes. See also *gel electrophoresis.*

EST: See *expressed sequence tag.*

Expressed sequence tag (EST): A content-addressable label that is a subsequence of an expressed *gene* (*cDNA*).

Exon: The protein-coding *DNA sequence* of a *gene*. See also *introns*.

Gel electrophoresis: *Electrophoresis* of *DNA* molecules using agarose and acrylamide gels as the media containing the mixtures of molecules.

Gene: The fundamental unit of heredity. A gene is a stretch of *DNA* that encodes a specific biological function or functional product (e.g. a protein).

Gene localization: Determination of the physical position of a *gene* on a *chromosome*.

Genetic linkage map: See *linkage map*

.

Genetic markers: See *markers*.

Genome: All the genetic material contained in the *chromosomes* of a particular organism.

Genotype: The genetic makeup of an individual; the genotype of an individual for a *marker* or a *gene* is comprises two *alleles*; genes on the *sex chromosomes* are a special case.

Genotyping: Determination of which pair of *alleles* is present in an individual.

Guanine (G): A *nitrogenous base* that is a member of the *base pair* G-C (*guanne* and *cytosine*).

Haplotype: See *phase*.

Haplotyping: Setting the *phase* for a given individual.

Heterozygosity: Probability that an individual is *heterozygous* for any two alleles at a gene locus. Mathematically,

$$H = 1 - \left(\sum_{i=1}^{n} p_i^{\,2} \right)$$

where p_i is the population frequency of the *i*th allele. See also *polymorphic information content.*

Heterozygous: An individual is heterozygous at a *gene locus* if the two *alleles* in the *genotype* are different. See also *homozygous*.

Homozygous: An individual is homozygous at a *gene locus* if the two *alleles* in the *genotype* are the same. See also *heterozygous*.

Hybridization: The binding of single strands of *DNA* or *RNA* to form double-stranded molecules.

Introns: Portions of the *DNA sequence* of a *gene* that are not part of the protein-coding sequence of the gene. See also *exons*.

In vitro: Outside a living organism.

Linkage: The proximity between *genes* or *markers* along a *chromosome*. The closer the markers are, the less likely they are to be separated due to *crossing over,* hence the more likely that they will be inherited together.

Linkage map: A map of the relative position of the *markers* and *genes* on a *chromosome*, determined by measuring how often the loci are inherited together. The unit of such a map is *centimorgan (cM).* See also *physical map.*

Locus: A location on a *chromosome* that can be identified in a distinctive manner.

Marker: An identifiable location on a *chromosome* that expresses some measurable form of *polymorphism* so that its inheritance can be traced in a pedigree. It can be a gene itself or some segment of DNA with no known function but whose inheritance can be traced in a pedigree.

Megabase (Mb): Unit of length for *DNA* fragments equal to one million *nucleotides* or *base pairs*. It is also approximately equal to 1 *cM*.

Meiosis: The cell division process that results in the creation of the *sex cells*.

Meiotic recombinations: *Recombination* events caused by *crossing over* during *meiosis*.

Mendelian disease: A disease that is caused by a defective genotype at a single gene locus. Usually, only one disease mechanism is operating in a given family, and possession of the high-risk *genotype* is necessary for disease expression. See also *non-Mendelian disease.*

Microsatellites: Genetic *markers* that contain tandem repeats of short simple *DNA sequences*.

Nitrogenous bases: A nitrogen-containing molecule that behaves chemically as a base.

Non-Mendelian disease: A disease caused by a complex interaction of multiple genetic and non-genetic factors. Non-Mendelian diseases are multifactorial: programmed by interacting genes and modified by mostly unknown environmental factors. Most common genetic diseases are non-Mendelian. See also *Mendelian disease.*

Nucleotides: A *DNA* subunit, consisting of a *nitrogenous base* (*adenine, guanine, thymine,* or *cytosine*) chemically linked to a phosphate molecule and a sugar molecule. A DNA molecule is made up of thousands of nucleotides linked to one another.

PCR: See *polymerase chain reaction.*

PCR stutter: Additional trailing shadow bands that occur during gel electrophoresis of PCR-amplified microsatellites. These bands are caused by the extraneous PCR products generated by *polymerase* slippage during *PCR* amplification.

Penetrance: Conditional probability of observing the corresponding *phenotype* given the specified *genotype.*

Phase: The combined *genotypes* of two or more polymorphic loci on the same parental chromosome.

Phenotype: The observable features of an organism that are the result of the *genotype*.

Physical map: A map of the actual location of *genes* and *markers* on a *chromosome* measured in *base pairs.* See also *linkage map.*

PIC: See *polymorphic information content.*

Polymerase: Enzymes that catalyze the synthesis of DNA on preexisting DNA templates.

Polymerase chain reaction (PCR): A technique for rapidly amplifying a *DNA* fragment by alternatively denaturing double-stranded DNA, annealing pairs of *primers* to both ends of the DNA segment, and synthesizing the DNA bracketed by the primers using a heat-stable *polymerase* .

Polymorphic information content (PIC): Probability that the marker genotype of a given offspring can be *haplotyped.* Mathematically,

$$PIC = 1 - \left(\sum_{i=1}^{n} p_i^2 \right) - \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 p_i^2 p_j^2$$

where p_i and p_j are the population frequencies of the *i*th and *j*th alleles. See also *heterozygosity.*

Polymorphism: Genetic variations in the *DNA sequence* of a *gene* or *marker*. A highly polymorphic gene or marker has a large number of possible *alleles*.

Primer: Short polynucleotide chain to which new deoxyribonucleotides can be added by DNA *polymerase.*

Proband: The affected person through which a pedigree is discovered and explored.

Probe: A labeled *DNA* or *RNA* sequence used to detect the presence of a complementary sequence by molecular *hybridization.* For example, a probe can be used to recognize a particular *clone* in a complex mixture of DNA or RNA molecules.

Purine: A single-ring nitrogenous basic compound that occurs in nucleic acids. Examples of purines are *adenine* and *guanine.*

Pyrimidine: A double-ring nitrogenous basic compound that occurs in nucleic acids. Examples of pyrimidines are *cytosine* and *thymine.*

Recombination: The process by which an offspring receives a combination of *genes* different from that of either parent. See also *meiotic recombination.*

Reverse transcriptase: An *RNA*-dependent *DNA polymerase* or enzyme that synthesizes DNA from an RNA template.

RNA (ribonucleic acid): The ribonucleotide polymer into which *DNA* is transcribed.

Sequence tagged site (STS): A short *DNA sequence* (200 to 500 *base pairs*) with known location and base sequence that has a single occurrence in the genome. The STSs are used as unique landmarks in developing the *physical map*.

Sequencing: Determination of the exact order of *nucleotides* in a *DNA*.

Sex chromosomes: The X and Y *chromosomes* in human beings that determine the sex of an individual. Females have two X chromosomes; males have an X and a Y chromosome. See also *autosomes.*

Sex cells: The sperm or egg cells. In the sex cells, there are only twenty-three *chromosomes* and not twenty-three pairs.

Simple disease: See *Mendelian disease.*

STS: See *sequence tagged site.*

Tandem repeats: Multiple copies of the same *nucleotide* sequence on a *chromosome,* useful as *markers* in genetic analysis.

Thymine (T): A *nitrogenous base* that is a member of the *base pair* A-T (*adenine* and *thymine*).

Vector: DNA molecule (e.g. from a virus) which can be inserted with another DNA fragment without losing its self-replicating capacity. Vectors are used to introduce foreign DNA into host cells where it can then be reproduced in large quantities.

Yeast artificial chromosome (YAC): An artificial yeast *chromosome* constructed by cloning DNA fragments (up to 1.5 mb) into *vectors* that can replicate in yeast cells.

References

Adams, M.D., Kelley, J.M., Gocayne, J.D., Dubnick, M., Polymeropoulos, M.H., Xiao, H., Merril, C.R., Wu, A., Olde, B., Moreno, R.F., Kerlavage, A.R., McCombie, W.R., and Venter, J.C. (1991). Complementary DNA sequencing: Expressed sequence tags and human genome project. *Science*, **252:** 1651-1656.

Buetow, K.H. (1991). Influence of aberrant observations on high-resolution linkage analysis outcomes. *Am. J. Hum. Genet.*, **49:** 985-994.

Collins, F. (1992). Positional cloning: lets not call it reverse anymore. *Nature Genet.*, **1**(1)**:** 3-6.

Collins, F.S. (1995). Positional cloning moves from perditional to traditional. *Nature Genet.*, **9**(4)**:** 347-350.

Davies, J.L., Kawaguchi, Y., Bennett, S.T., Copeman, J.B., Cordell, H.J., Pritchard, L.E., Reed, P.W., Gough, S.C.L., Jenkins, S.C., Palmer, S.M., Balfour, K.M., Rowe, B.R., Farrall, M., Barnett, A.H., Bain, S.C., and Todd, J.A. (1994). A genome-wide search for human type 1 diabetes susceptibility genes. *Nature*, **371**(6493)**:** 130-136.

Galat, A. (1989). A procedure for analysis of densitometric spectra. *Electrophoresis*, **10:** 659-667.

Ghosh, S., and Collins, F.S. (1996). The geneticist's approach to complex disease. *Annu. Rev. Med.*, **47:** 333-353.

Ghosh, S., Karanjawala, Z., Hauser, E., Ally, D., Knapp, J., Rayman, J., Musick, A., Tannenbaum, J., Te, C., Shapiro, S., Eldridge, W., Musick, T., Martin, C., Smith, J., Carpten, J., Brownstein, M., Powell, J., Whiten, R., Chines, P., Nylund, S., Magnuson, V., Boehnke, M., Collins, F.S., and FUSION (1997). Methods for precise sizing, automated binning of alleles, and reduction of error rates in large-scale genotyping using fluorescently labeled dinucleotide markers. *Genome Methods*, **7:** 165-178.

Golub, G.H., and Van Loan, C.F. (1989). "Matrix Computations"*, 2nd Edition.* Baltimore: John Hopkins University Press.

Gyapay, G., Morissette, J., Vignal, A., Dib, C., Fizames, C., Millasseau, P., Marc, S., Bernardi, G., Lathrop, M., and Weissenbach, J. (1994). The 1993-94 Genethon Human Genetic Linkage Map. *Nature Genetics*, **7**(2)**:** 246-339.

Hauge, X.Y., and Litt, M. (1993). A study of the origin of 'shadow bands' seen when typing dinucleotide repeat polymorphisms by the PCR. *Hum. Molec. Genet.*, **2**(4)**:** 411-415.

Hearne, C.M., Ghosh, S., and Todd, J.A. (1992). Microsatellites for linkage analysis of genetic traits. *Trends in Genetics*, **8**(8)**:** 288-294.

Hoffman, E.P. (1994). The Human Genome Project: Current and future impact. *Am. J. Hum. Genet.*, **54:** 129-136.

Hopcroft, J.E., and Ullman, J.D. (1979). "Introduction to Automata Theory, Languages, and Computation". Reading, Mass.: Addison-Wesley.

Hyder, R.N., Julier, C., Buckley, J.D., Trucco, M., Rotter, J., Spielman, R.S., and al., e. (1991). High-resolution linkage mapping for susceptibility for genes in human polygenic disease: insulin-dependent diabetes mellitus and chromosome 11q. *Am. J. Hum. Genet.*, **48:** 243-257.

Jeffreys, A.J., Wilson, V., and Thein, S.L. (1985). Hypervariable 'minisatellite' regions in human DNA. *Nature*, **314:** 67-73.

Jones, S.W., Cai, D., Weislow, O.S., and Esmaeli-Azad, B. (1997). Generation of multiple mRNA fingerprints using fluorescence-based differential display and an automated DNA sequencer. *BioTechniques*, **22**(3)**:** 536-543.

Jordan, E. (1992). The Human Genome Project: where did it come from, where is it going? *Am. J. Hum. Genet.*, **51:** 1-6.

Kruglyak, L., Daly, M.J., Reeve-Daly, M.P., and Lander, E.S. (1996). Parametric and nonparametric analysis: a unified multipoint approach. *Am. J. Hum. Genet.*, **58:** 1347–1363.

Lander, E.S., and Schork, N.J. (1994). Genetic dissection of complex traits. *Science*, **265:** 2037-2048.

Lange, K., Weeks, D.E., and Boehnke, M. (1988). Programs for pedigree analysis: MENDEL, FISHER, and dGENE. *Genetic Epidemiology*, **5:** 471-472.

Lawson, C.L., and Hanson, R. (1974). "Solving Least Squares Problems". Englewood Cliffs, NJ: Prentice-Hall.

LeDuc, C., Miller, P., Lichter, J., and Parry, P. (1995). Batched analysis of genotypes. *PCR Methods and Applications*, **4:** 331-336.

Lehrach, H., Drmanac, A., Hoheisel, J., Larin, Z., Lennon, G., Monaco, A.P., Nizetic, D., Zehetner, G., and Poustka, A. (1990). Hybridization fingerprinting in genome mapping and sequencing. *In* "Genetic and Physical Mapping I: Genome Analysis", (Davies, K. E., and Tilghman, S. M., ed.), 39-81. Cold Spring Harbor, New York: Cold Spring Harbor Laboratory.

Liang, P., and Pardee, A. (1992). Differential display of ekaryotic messenger RNA by means of the polymerase chain reactions. *Science*, **257:** 967-971.

Litt, M., and Luty, J.A. (1989). A hypervariable microsatellite revealed by in vitro amplification of a dinucleotide repeat within the cardiac muscle actin gene. *Am. J. Hum. Genet.*, **44:** 397-401.

Luehrsen, K.R., Marr, L.L., van der Knapp, E., and Cumberledge, S. (1997). Analysis of differential display RT-PCR products using fluorescent primers and Genescan software. *BioTechniques*, **22**(1)**:** 168-174.

Magnuson, V.L., Ally, D.S., Nylund, S.J., Karanjawala, Z.E., Rayman, J.L., Knapp, J.I., Lowe, A.L., Ghosh, S., and Collins, F.S. (1996). Substrate nucleotide-determined non-templated addition of Adenine by Taq DNA polymersase: Implications for PCR-based genotyping and cloning. *BioTechniques*, **21**(4)**:** 700-709.

Mansfield, D.C., Brown, A.F., Green, D.K., Carothers, A.D., Morris, S.W., Evans, H.J., and Wright, A.F. (1994). Automation of genetic linkage analysis using fluorescent microsatellite markers. *Genomics*, **24**(2)**:** 225-233.

Marquardt, D.W. (1963). *Journal of the Society for Industrial and Applied Mathematics.*, **11:** 431-441.

MathWorks (1993). "MATLAB Reference Guide". Natwick, MA: The MathWorks, Inc.

Matise, T.C., Perlin, M.W., and Chakravarti, A. (1994). Automated construction of genetic linkage maps using an expert system (MultiMap): application to 1268 human microsatellite markers. *Nature Genetics*, **6**(4)**:** 384-390.

Monaco, A.P., Lam, V.M.S., Zehetner, G., Lennon, G.G., Douglas, C., Nizetic, D., Goodfellow, P.N., and Lehrach, H. (1991). Mapping irradiation hybrids to cosmid and yeast artificial chromosome libraries by direct hybridization of Alu-PCR products. *Nucleic Acids Res.*, **19**(12)**:** 3315-3318.

Moore, S.S., Sargeant, L.L., King, T.J., Mattick, J.S., Georges, M., and Hetzel, D.J.S. (1991). The conservation of dinucleotide microsatellites among mammalian genomes allows the use of heterologous PCR primer pairs in closely related species. *Genomics*, **10**(3)**:** 654-660.

Mullis, K.B., Faloona, F.A., Scharf, S.J., Saiki, R.K., Horn, G.T., and Erlich, H.A. (1986). Specific enzymatic amplification of DNA *in vitro*: the polymerase chain reaction. *Cold Spring Harbor Symp. Quant. Biol.*, **51:** 263-273.

Nakamura, Y., Leppert, M., O'Connell, P., Wolff, R., Holm, T., Culver, M., Martin, C., Fujimoto, E., Hoff, M., Kumlin, I., and White, R. (1987). Variable number tandem repeat (VNTR) markers for human gene mapping. *Science*, **235:** 1616-1622.

NIH/CEPH Collaborative Mapping Group (1992). A Comprehensive Genetic Linkage Map of the Human Genome. *Science*, **258:** 67-86.

Odelberg, S.J., and White, R. (1993). A Method for Accurate Amplification of Polymorphic CA-repeat Sequences. *PCR Methods and Applications*, **3:** 7-12.

Ott, J. (1991). "Analysis of Human Genetic Linkage", *Revised Edition.* Baltimore, Maryland: The Johns Hopkins University Press.

Papoulis, A. (1977). "Signal Analysis". New York: McGraw-Hill Book Company.

Perlin, M.W., Burks, M.B., Hoop, R.C., and Hoffman, E.P. (1994). Toward fully automated genotyping: allele assignment, pedigree construction, phase determination, and recombination detection in Duchenne muscular dystrophy. *Am. J. Hum. Genet.*, **55**(4)**:** 777-787.

Perlin, M.W., Lancia, G., and Ng, S.-K. (1995). Toward fully automated genotyping: genotyping microsatellite markers by deconvolution. *Am. J. Hum. Genet.*, **57**(5)**:** 1199-1210.

Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T. (1992). "Numerical Recipes in C: The Art of Scientific Computing", *2nd Edition.* Cambridge: Cambridge University Press.

Ribeiro, E.A., and Sutherland, J.C. (1991). Quantitative Gel Electrophoresis of DNA: Resolution of Overlapping Bands of Restriction Endonuclease Digests. *Anal. Biochem.*, **194:** 174-184.

Rich, E., and Knight, K. (1991). "Artificial Intelligence". New York, NY: McGraw-Hill, Inc.

Richards, D.R., and Perlin, M.W. (1995). Quantitative analysis of gel electrophoresis data for automated genotyping applications. *Amer. J. Hum. Genet.*, **57**(4 Supplement)**:** A26.

Risch, N. (1990). Genetic Linkage and Complex Diseases, With Special Reference to Psychiatric Disorders. *Genet. Epidemiol.*, **7:** 3-16.

Risch, N. (1991). A Note on Multiple Testing Procedures in Linkage Analysis. *Am. J. Hum. Genet.*, **48:** 1058-1064.

Risch, N., and Merikangas, K. (1996). The future of genetic studies of complex human diseases. *Science*, **273:** 1516-1517.

S.A.G.E. (1997). The SIBPAL S.A.G.E. Program for Sib-Pair Linkage Analysis, ver. 3.0, Program, Department of Epidemiology and Biostatistics ,Case Western Reserve University, http://darwin.mhmc.cwru.edu/pub/sibpal.html.

Schmid, C.W., and Jelinek, W.R. (1982). The Alu family of dispersed repetitive sequences. *Science*, **216:** 1065-1070.

Schwengel, D.A., Jedicka, A.E., Nanthakumara, E.J., Weber, J.L., and Levitt, R.C. (1994). Comparison of fluorescence-based semi-automated genotyping of multiple microsatellite loci with autoradiographic techniques. *Genomics*, **22:** 46-54.

Southern, E.M. (1979). Measurement of DNA length by gel electrophoresis. *Analytical Biochemistry*, **100:** 319-323.

Terwilliger, J.D.a.O., Jurg (1994). "Handbook of Human Genetic Linkage", *Revised Edition.* Baltimore, Maryland: The Johns Hopkins University Press.

Todd, J.A. (1994). The Emperor's New Genes: 1993 RD Lawrence Lecture. *Diabetic Medicine*, **11:** 6-16.

Vohradsky, J., and Pánek, J. (1993). Quantitative analysis of gel electrophoretograms by image analysis and least squares modeling. *Electrophoresis*, **14:** 601-612.

Watson, J.D. (1990). The Human Genome Project: Past, present, and future. *Science*, **248:** 44-49.

Watson, J.D., and Crick, F.H.C. (1953). Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid. *Nature*, **171:** 737-738.

Watson, J.D., Gilman, M., Witkowski, J., and Zoller, M. (1992). "Recombinant DNA", *second Edition.* New York, New York: W.H. Freeman and Company.

Weber, J., and May, P. (1989). Abundant class of human DNA polymorphisms which can be typed using the polymerase chain reaction. *Am. J. Hum. Genet.*, **44:** 388-396.

Weeks, D.E., Sobel, E., O'Connell, J.R., and Lange, K. (1995). Computer programs for multilocus haplotyping of general pedigrees. *Am. J. Hum. Genet.*, **56:** 1506-1507.

Weissenbach, J., Gyapay, G., Dib, C., Vignal, A., Morissette, J., Millasseau, P., Vaysseix, G., and Lathrop, M. (1992). A second generation linkage map of the human genome. *Nature*, **359:** 794-801.

Wijsman, E.M. (1987). A Deductive Method of Haplotype Analysis in Pedigrees. *Am. J. Hum. Genet.*, **41:** 356-373.

Ziegle, J.S., Su, Y., Corcoran, K.P., Nie, L., Mayrand, P.E., Hoff, L.B., McBride, L.J., Kronick, M.N., and Diehl, S.R. (1992). Application of automated DNA sizing technology for genotyping microsatellite loci. *Genomics*, **14:** 1026-1031.