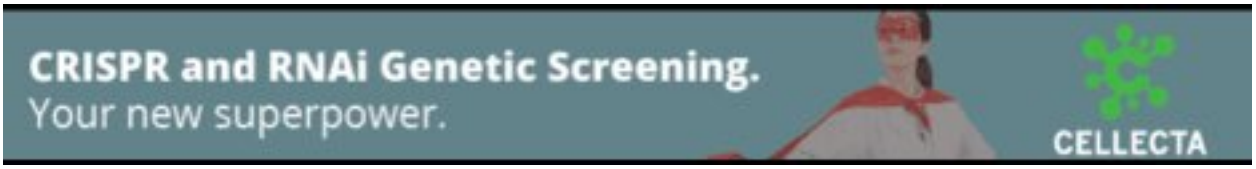
A Software System for Data Analysis in Automated DNA Sequencing

Michael C. Giddings, Jessica Severin, Michael Westphall, et al.

Genome Res. 1998 8: 644-665
Access the most recent version at doi:10.1101/gr.8.6.644



LETTER

A Software System for Data Analysis in Automated DNA Sequencing

Michael C. Giddings,[1] Jessica Severin, Michael Westphall, Jiazhen Wu, and Lloyd M. Smith

Department of Chemistry, University of Wisconsin–Madison, Madison, Wisconsin 53706 USA

Software for gel image analysis and base-calling in fluorescence-based sequencing consisting of two primary programs, BaseFinder and GelImager, is described. BaseFinder is a framework for trace processing, analysis, and base-calling. BaseFinder is highly extensible, allowing the addition of trace analysis and processing modules without recompilation. Powerful scripting capabilities combined with modularity and multilane handling allow the user to customize BaseFinder to virtually any type of trace processing. We have developed an extensive set of data processing and analysis modules for use with the program in fluorescence-based sequencing. GelImager is a framework for gel image manipulation. It can be used for gel visualization, lane retracking, and as a front end to the Washington University Getlanes program. The programs were designed using a cross-platform development environment, currently allowing them to run in Windows NT, Windows 95, Openstep/Mach, and Rhapsody. Work is ongoing to deploy the software on additional platforms, including Solaris, Linux, and MacOS. This software has been thoroughly tested and debugged in the analysis of >2 million bp of raw sequence data from human chromosome 19 region q13. Overall sequencing accuracy was measured using a significant subset of these data, consisting of ~600 sequences, by comparing the individual shotgun sequences against the final assembled contigs. Also, results are reported from experiments that analyzed the accuracy of the software and two other well-known base-calling programs for sequencing the M13mp18 vector sequence.

[The sequence data described in this paper have been submitted to the GenBank data library under accession no. AF025422]

Large-scale sequencing efforts have begun to approach the goals initially outlined for the Human Genome Project (HGP). The project has as its goal to map and sequence the entire human genome, consisting of 3 billion bp of DNA, by the year 2006 (Smith and Hood 1987). The result of worldwide finished sequencing efforts to date for all organisms, as judged by the submissions to GenBank, is ~1.26 billion bp (http://www.ncbi.nlm.nih.gov/Web/Genbank/index.html; December 1997). The goals set forth for the HGP, from the perspective of the sequencing performed thus far, are ambitious. Fortunately, the effort has stimulated extensive developments in automated technologies for sequencing, which has resulted in a significant ramp-up in sequencing throughput, a trend that appears to be continuing. Meeting the goals of the project will require continued improvement of current technologies and the development of new ones.

The process of sequencing can be roughly divided into three significant steps, each with its own challenges (Smith 1993). These steps are commonly labeled Front End, Separation and Detection, and Back End. The Front End is where the molecular biology occurs; its task is to prepare samples for electrophoretic separation, most commonly using the shotgun approach (Hunkapiller et al. 1991). The Separation and Detection step uses a gel electrophoresis instrument to separate fragments in the prepared DNA samples. The data collected are passed to the Back End, consisting of computer analysis, to determine the nucleotide sequence of the input DNA. In addition, there is often a fourth component, the finishing process, that consists of a feedback loop from the Back End into the Front End to provide closure for gaps after initial shotgun sequencing has occurred (e.g., see Fleischmann et al. 1995).

Significant strides have been made in all of these phases of the sequencing process. Increasingly, Front End steps are being automated by robotic systems (Wilson et al. 1990; Garner et al. 1992). Gel separation technologies have been improved constantly with the use of fluorescence-

[1]Corresponding author.
E-MAIL giddings@whitewater.chem.wisc.edu; FAX (608) 265-6780.

 8:644–665 ©1998 by Cold Spring Harbor Laboratory Press ISSN 1054-9803/98 $5.00; www.genome.org

based sequencing in thin slab gels and capillaries (Kostichka et al. 1992; Mathies and Huang 1992; Carrilho et al. 1996; Swerdlow et al. 1997). In some respects, Back End development has lagged behind automation of the previous two steps, creating a potential bottleneck in data flow and handling. One of the reasons for this lag is clear; it is necessary to develop the chemistries and instrumentation before designing algorithms to process the output from those steps. And good software design, particularly when dealing with large amounts of complex data, is not a rapid process.

The Back End covers a large territory of computer analysis that occurs in obtaining finished DNA sequence ready to be submitted to a database. Significant steps in this process include lane finding (for slab gel electrophoresis), base-calling, assembly, and finishing. Much of the effort in developing the Back End portion of sequencing has focused on the latter two steps. Only more recently have published works begun to significantly address the first two steps, particularly for fluorescence-based sequencing, which has become the standard for genomic sequencing (Giddings et al. 1993; Golden et al. 1993; Ives et al. 1994; Berno 1996; Cooper et al. 1996).

Our efforts to improve throughput and technologies in the Front End and Separation and Detection steps resulted in the development of several generations of custom fluorescence-based gel electrophoresis systems (Luckey et al. 1990; Kostichka et al. 1992; M. Westphall, unpubl.). At the time of development of the first of these, the only software available for analysis of data from fluorescence-based systems was bundled with a commercial sequencing system and was not readily adaptable to analysis of data from our instruments. This prompted the development of software for analysis of the data produced by these machines. Because the designs of the instruments were changing frequently as a result of ongoing research and development, it became clear that a flexible, modular approach to software design was needed that would allow the rapid and easy customization of the analysis software to a particular instrument or chemistry without intervention of a programmer.

The initial results of this effort, primarily focused on the base-calling step, were promising (Giddings et al. 1993). Two primary software packages were developed as part of this effort. One package, called GelImager, is aimed at gel image analysis. It allows the viewing and manipulation of the electrophoretic image files collected from the electrophoresis instrument, acts as an interface to the lane-finding software Getlanes which was developed by the St. Louis Genome Center (Cooper et al. 1996), and allows the retracking of lanes determined by that software (those who wish to use Getlanes need to obtain permission from the authors of that package). A second software package, called BaseFinder, is designed to process the trace files output by the lane-finding module, performing analysis and base-calling to produce a DNA sequence from them. The latter program has been the primary focus of our development efforts, and, therefore, is the most extensively described herein. Since the 1993 publication of the base-calling algorithm, an extensive array of additional preprocessing modules and user interface features have been implemented. Many of the additions were made in response to the rigors of testing in our early attempts to perform production sequencing. We discovered that the first version of the program, although working well for test sequences such as M13, was in need of substantial improvement for use in genomic sequencing. We found that especially critical to the effectiveness of the base-calling process was the preprocessing of the electrophoretic trace data. Also significant were improvements to the user interface and program framework, including the script definition facilities, to be described further below.

Another program developed during this period by our group is MatrixFinder, which aids in the visualization and determination of multicomponent separation matrices for multiple color fluorescence-based sequencing (Yin et al. 1996). Since publication, the only changes to the program have been a port to OpenStep to increase portability (described below) and some user interface additions. Since the time of that work further refinement of the multicomponent algorithm has been performed in another laboratory (Huang et al. 1997). The program will be described briefly below in conjunction with the related multicomponent transform step that is part of BaseFinder.

All of these programs and source code are available for free for noncommercial use. More information can be obtained from our web site (http://smithlab.chem.wisc.edu/Research/informatics.html). They were developed using the OpenStep object toolkit (http://www.enterprise.apple.com/openstep). OpenStep consists of a complete set of objects encompassing common user interface elements, data structures, and cross-platform distributed object communications. It acts essentially as a layer-insulating program development from the specifics of individual operating systems. As such, it has provided the benefit of cross-platform deploy-

ment as well as reduced programming and maintenance overhead, particularly for the user–interface portions of the programs. OpenStep run-times have been developed for Windows NT and 95, Rhapsody (Apple), and Solaris (SunSoft). The OpenStep/Solaris run-time is not actively being promoted by Sun because of their focus on JAVA (http://www.sun.com/java/). It was available for free in the past on their web site but is no longer listed. Apple plans a version of the toolkit to run on top of Mac OS 8.x, and is also expanding the toolkit interfaces (application programming interfaces, APIs) to provide complete access from and integration with JAVA. A GNU implementation of the OpenStep standard, called GnuStep, is underway and making good progress (http://www.gnustep.org). Once complete, the latter will provide further vendor and platform independence for the software, and will provide an alternative means of deployment on Solaris, an important target given that it is in use by many genome-sequencing centers. We are working with a GnuStep consulting firm to speed the deployment of the programs in GnuStep on Solaris, and at this time (March 1998) the work is nearly complete.

The remainder of this paper will describe GelImager and BaseFinder, covering the basics of the human user interface and discussing in more detail the algorithms used, particularly in trace preprocessing. The last section will discuss the results of applying the software to several cosmid-sized sequencing projects. This will include a discussion of the problems encountered and the software development that will be required to achieve further automation for hands-off processing. Also presented are results of a comparison of BaseFinder's processing and base-calling with both Phred (Ewing et al. 1998) and the software included with the ABI 377 sequencer (Perkin Elmer Corp.).

GelImager—Gel Image Viewing and Manipulation

Program Overview and User Interface

GelImager arose from a need to inspect and manually find lanes on gel images derived from several sequencing instruments developed in our laboratory. At the time it was created we did not have access to automated lane-tracking software, and it was necessary to have a means of deriving trace-file information from the gel images produced by the sequencing instruments developed in the laboratory. With the more recent incorporation of Getlanes, automated lane-tracking software developed at Washington University (St. Louis, MO), GelImager's purpose has changed but it remains important. It is now most commonly used to allow manual inspection of the lane tracking performed by Getlanes to correct any mistakes. Although the Getlanes algorithm substantially reduces the time and effort involved in deriving lanes from the gel, it generally does not perform perfectly on data from our instruments, necessitating manual inspection and correction afterward. Therefore, GelImager has been tailored primarily to the task of inspection and retracking of lanes on gel image files.

A second purpose for GelImager is in research, development, and debugging of instruments or molecular biology techniques. In the course of such work, it is crucial to have the ability to display the complete results of a run for inspection. This can provide a clear indication of problems with a separation, such as lane wandering or failed reactions. GelImager's flexible viewing and manipulation of gel image files works well for this purpose.

The user interface of the program is straightforward. Figure 1 displays a screen capture of the program running in Windows NT with the two primary windows displayed. The menus at the top of the main window (on the left) provide standard operations such as file opening, saving, and window display. Several file formats can be read and saved to, including the one produced by our current in-house sequencing instrument (''GELI'' format), ABI gel image format (no saving), tiff format, and the lmark format produced by our port of the Getlanes program. The main window displays a gel image file that has been opened—only one at a time can be viewed. This limitation is in place because of the size of the image files; they can be up to 60 MB, requiring a large amount of system resources. It also simplifies program structure for both user and programmer to allow manipulation of only one image at a time.

GelImager attempts to deal with the unwieldy size of these files by accessing and drawing only the data that are necessary for display to the user. This provides for faster initial drawing of the image, but causes a delay in scrolling up or down in the image when expanded to full size. The file scanning and drawing code has been optimized extensively. However, it greatly helps usability to have a computer system available with physical memory greater than the image size, and a modern fast processor.

The gel image is displayed using a false-color scheme, which allows visual distinction of fragment bands labeled with different fluorescent markers. Data are collected by the sequencer at four wavelengths. Each of the four fluorescent markers typi-

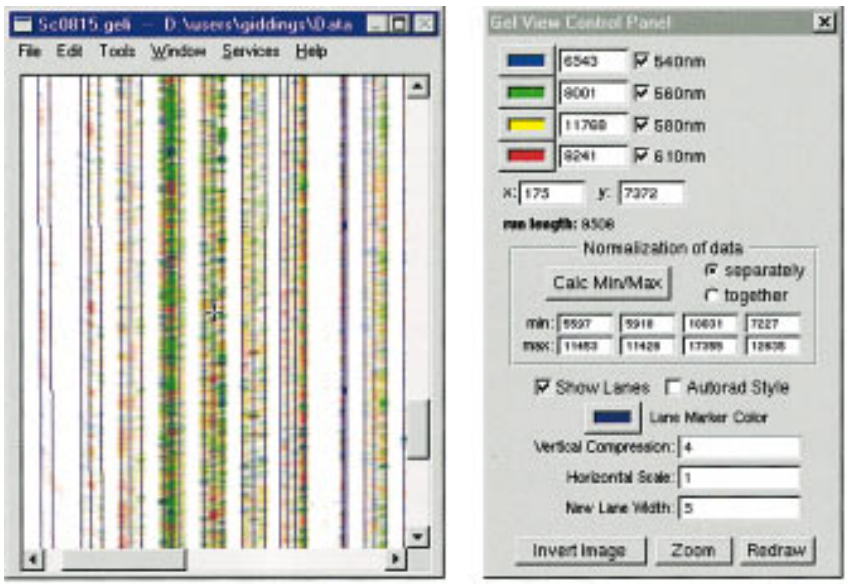


Figure 1 A screen capture of the GelImager program running in Windows NT. The left window is the main gel image view, displaying an image collected from the scanning electrophoresis system developed in our laboratory, with lane marks superimposed on lanes as determined by Getlanes, and then manually edited. The vertical dimension in the image corresponds with scan number (run-time), with earlier scans toward the top. The horizontal dimension corresponds with intensity values collected on a scan by the optical system across the gel plates. The lane fifth from the right of the window is shown selected, with two lane delimiter lines and several inflection points that can be adjusted to follow curves in the lane. The *right* window is the main control panel. It provides for adjustment of items such as color mapping (color wells near top), data display normalization (central position), and image size scaling in the gel view window (values near the bottom). It also shows the position of the cursor in the window (*x:* and *y:*), and the intensity values at the cursor position (the numerical values next to the color wells).

cally used has a unique spectral signature when sampled across the wavelengths. The scheme for displaying these is simple; the user can choose a color to correspond to each wavelength. Each pixel is displayed by mixing the four chosen colors in proportion to the intensity of the light detected at the corresponding wavelengths. Assuming a reasonable choice of distinct colors, this makes it easy to distinguish visually bands corresponding to the four different dyes being used, as each dye presents a unique color mix on the image. An alternative approach we have more recently explored is to apply a multicomponent transform matrix (Frans and Harris 1985) to the gel image itself. Normally, this transform step is applied during trace processing, as will be described below. Applying it at this step has the effect of resolving individual dye intensities (and therefore concentrations) directly; however, the problem remains that a mapping must be made from the four-component dye intensity space to a three-component RGB space. This is done by assigning different RGB colors to each of the four markers (e.g., red, green, blue, yellow) and mixing them according to intensity of the corresponding marker signal. The primary advantage of the multicomponent scheme is that it tends to make the band colors more visually distinct than is usually achieved with the false-color scheme. It is also anticipated that this may aid the automated lane-finding process, but we have not yet had the opportunity to fully test this hypothesis.

Lane-tracking marks can be added, deleted, and manipulated in the main window in GelImager using the mouse pointer. Lane marks are normally displayed as a single line, containing an arbitrary number of inflection points, down the center of the marked lane as displayed in Figure 1. Clicking on a lane marker highlights it, causing it to be displayed with two parallel lines indicating the lane delimiters, along with the line indicating the center. Inflection points can be added, moved, or deleted to allow for variations in the lane. An entire lane can be deleted by first highlighting it, then tapping the delete/backspace key. New lane marks can be added by holding down the control key, which causes a vertical line to appear that follows the pointer. Subsequent clicking of the pointer button causes the line to be placed at the present location. These features allow the easy loading and correction of lane marks produced by Getlanes or other automated lane-finding programs.

Using a distributed client/server approach, GelImager acts as an interface to Getlanes. We have used the source code for Getlanes and wrapped it inside an object-oriented interface. This interface is then made available as a server to the network through a simple program that can be run on any system having Portable Distributed Objects or OpenStep installed (Apple Computer, Cupertino, CA). This server then accepts requests from remote objects, which can be on a physically distinct ma-

chine or in a different process/task space in the same machine, processes each request, and outputs a lmark file as the result. When completed, it messages GelImager that the result is available and can be displayed. One of the reasons for implementing it in this distributed fashion is to allow a single computer to be set up with a fast processor and large memory, and any number of clients running on less expensive systems to easily access its services. Also, for a high throughput sequencing environment, this approach makes it straightforward to distribute the compute load among a number of machines to reduce processing bottlenecks. Furthermore, it is planned that this approach will facilitate the integration of Getlanes with a distributed, object-based laboratory information management system that is under development.

After lane finding has been completed (manually or automatically) and the results inspected, each lane is extracted by GelImager into an individual ''trace'' file. The trace is an intensity versus time (i.e., scan number) profile from top to bottom of a lane along the tracking line. It is derived by averaging each row of intensity values within a lane boundary into a single value for each wavelength. Therefore, each row within a lane marker boundary produces four intensity values at the corresponding time value (scan number) in the trace file. The result is a chromatographic style profile of the data within a lane. It should be noted that there is the potential for information loss in this process, illustrated by the fact that if a lane profile is too wide, resolution is lost, and if too narrow, the signal-to-noise ratio is lowered. These trade-offs are attributable to the often nonideal band profiles present in the data, and the fact that strictly horizontal integration does not account for such variations. Alhough this is a legitimate concern, at the present time enough advantages are gained by reducing the data to a more compact form that it is the standard for automated sequencers. Advantages include reduced storage requirements, enhanced visualization, and decreased signal processing time for steps such as noise filtering and deconvolution (described below).

The control window (Fig. 1, right window) allows adjustment of several parameters for both viewing and lane manipulation. Four color wells at the top of the window allow control over the mapping between wavelengths detected and the colors displayed, as described above, and the boxes next to the color wells display intensity values in the four channels for the last point in the image that was clicked on. The window also allows control over the lane width, image size (both vertical and horizontal scaling), reading direction (top to bottom or bottom to top), and adjustment of the upper and lower intensity thresholds for display of the image.

Getlanes is accessed by selecting Getlanes under the Tools menu. This displays the Getlanes parameter panel. The panel allows the specification of a gel file upon which the Getlanes process will be invoked, the output path for the resulting lmark (lane marker) file, the host upon which to queue the job (on which a Getlanes server process must be running), and parameters for aspects such as boundary determination method and algorithm type. Also, it allows control over some additional preprocessing functions added by us to enhance the performance of Getlanes, which consist of two different types of convolution operations: one using a Gaussian for bandpass filtering, and the second using a ''Mexican Hat'' filter for edge enhancement.

GelImager is a simple but useful tool for interactive editing and manipulation of gel image files. It is our goal that this program be eventually phased out and replaced by fully automated lane-finding software; however, that has not yet arrived, making this software an important component of the sequencing process in slab-gel-based systems.

BaseFinder—Trace Analysis and Base-Calling

Program Overview and User Interface

The BaseFinder program was designed to accept multispectral trace data, collected on fluorescence-based sequencing instruments, to process the data through a set of processing tools, and then to allow saving of processed traces and/or sequences. Each ''trace'' corresponds to a single lane on a slab-gel or a single capillary, and represents the intensity profile versus time of the labeled DNA fragments being separated in that lane.

BaseFinder can read and write trace data in a number of formats. Formats it is capable of reading are SCF (Dear and Staden 1992), ABI 373 and 377, tab-delimited text, binary floating point, and our own ''lane'' format. It can write processed traces to SCF, lane, tab-delimited text numbers, and raw binary floating point. Sequences can be saved directly in text form (similar to ABIs format) or to FASTA format. Collections of files stored within a directory can be saved and loaded in aggregate to allow simple multitrace processing.

The program itself does no processing of the data. All processing is done through tool modules that are developed separately from the program and

then loaded (linked) at run-time. This allows a great deal of flexibility for the types of processing and analysis that can be done. The program acts as a framework for interaction between the user, the processing tools, and the data. It has user interface features that allow data manipulation and viewing, control over parameters used by the tools in processing data, and the definition of scripts that specify the complete set of steps for processing data from particular instrument configurations. Once a script is defined, it can be saved and then recalled later for application to trace data from another lane or a whole group of lanes at one time.

An overview of the user interface of the program is shown in Figure 2 (with the program running on Openstep/Mach), with five windows and a menu. The largest window is the main program window. This displays the trace data, allowing adjustment of a number of parameters controlling its display. These parameters include control over both horizontal and vertical scaling of the data, how many panels the data set is broken down into for simultaneous viewing of subsequent trace segments, assignment of color to the different trace channels, and which data channels or base labels will be displayed. The window also has several tracking options (top left) that display the specific base and data values at any point clicked on by the user with the mouse. The panels displaying segments of trace data can be set to display tick marks with base number or data point number at their top. Base labels are placed below the traces, and the color as well as label used can be adjusted by the user.

On the Windows platform, all menus are associated directly with the main trace display window. On the Mach platform, the menu is floating to the left of the window as in the figure. The menu provides actions such as loading/saving data, printing, control over which windows are being displayed, setting of preferences, and how the data will be scaled for viewing.

The small window on the lower left of Figure 2

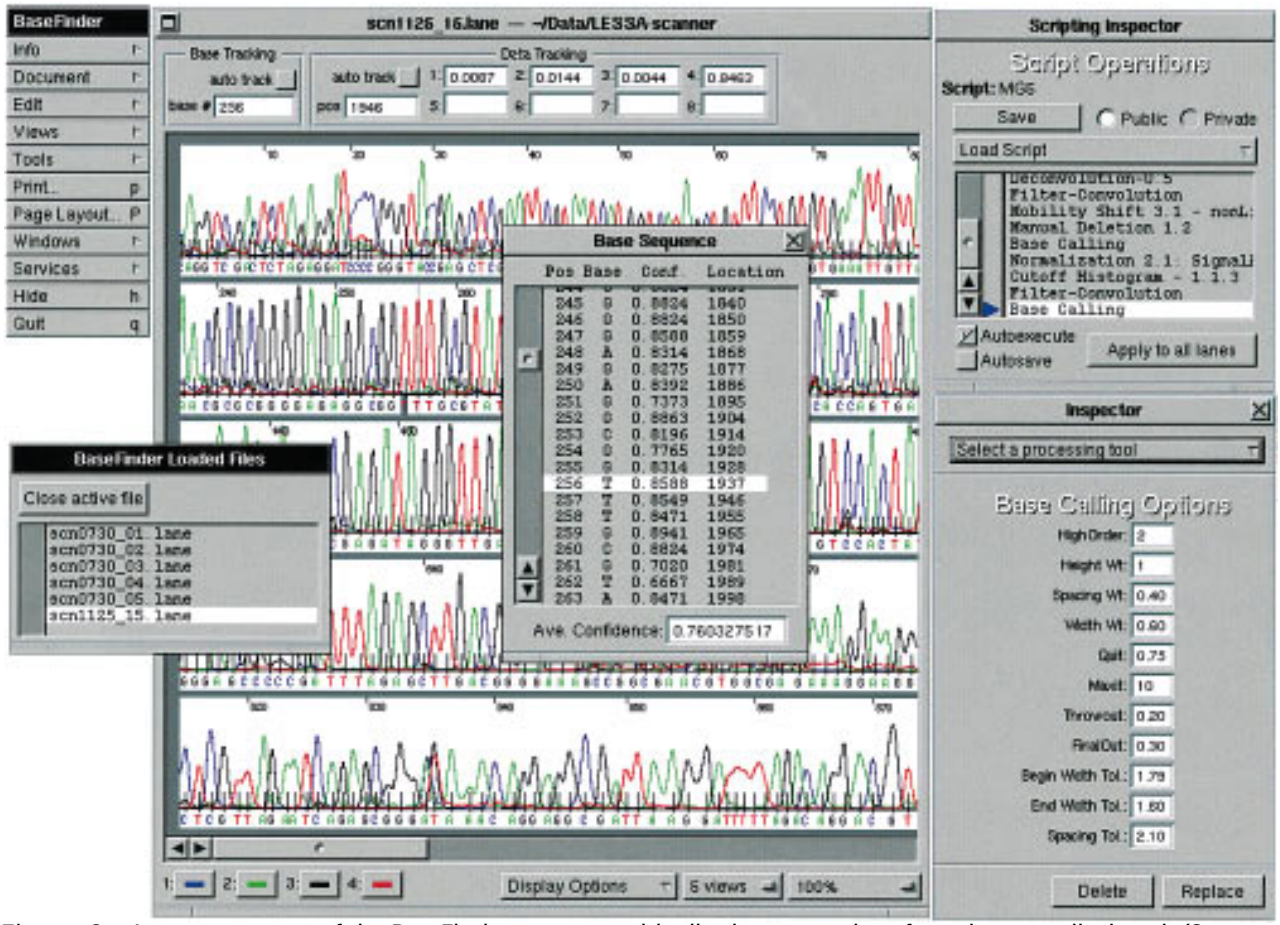


Figure 2 A screen capture of the BaseFinder program with all primary user interface elements displayed. (See text for a detailed description.)

is the multilane manager window, which displays the file names of all currently open lane files. To the end user, these files all appear to be ''loaded'' into the program, and clicking on any one of them in the list displays the corresponding trace in the main display. However, for reasons of memory efficiency, only one actual data set is resident in the program at any one time. The primary purpose of this window is to allow the application of program operations to any arbitrarily large set of trace files. Such operations include saving, processing, and base-calling. Multiple traces are selected for such operations by holding the shift key while clicking on several different trace names, or by clicking on one and dragging upward or downward in the list with the button still pressed.

Overlaid on the top of the main window in this figure is the base sequence panel. This displays the list of base label assignments for the trace, as well as their associated confidences and data point positions. If a base in this list is clicked on by the user, that base is highlighted in the main window so its position in the trace can be ascertained. Likewise, if a base label is clicked on in the main window, the corresponding base in the panel is highlighted.

The scripting window is located to the upper right of the main window. This provides access to one of the most useful features of the program. As mentioned, the program does not itself do processing or analysis. Instead, this is done through loadable tool modules that have a well-defined, object-oriented API. The scripting mechanism provides a means by which a sequence of tool-based processing steps can be applied to the data in an automated way. It functions similarly to a macro in common spreadsheet programs.

A script is defined by initially processing a data sample (trace) manually. Tools are selected from the tool window menu, parameters are adjusted for each tool, and they are applied to the data. The effects on the data are observed in the main window, and if unsatisfactory because of incorrect parameters or some other problem, undo can be selected from the Edit menu. Alternatively, the previous step in the processing can be mouse-selected in the script window, in which case the data will revert to displaying the state at the selected step. In fact, the user can select any step listed in the scripting window, and the data main window will display the state of the data at that particular processing step. This provides the ability to step through a script and learn what each tool and parameter combination does to the data.

There are several actions that can be performed to modify a script if necessary. A particular step can be selected, and a different processing tool can be selected to replace it in the script, parameters can be changed for the existing tool in the script, or the step can be deleted from the script. A new step can be appended after the selected step or inserted before the selected step. In addition, there is an option to allow the selection of steps in the script without synchronizing the data in the main window. This is useful for experienced users who wish to step back in a script and observe parameters used without having the overhead of recalculating and displaying the data at the corresponding steps. A recent change to the program has been moving the data processing into an execution thread separate from the user interface. This means that the user interface is not tied up, and therefore, still responds to the user, when the program is processing data. This is useful, but it can be confusing to the user because there is a delay between the time at which a step in the script is clicked on and the time at which the data is processed and displayed for that step. To rectify this, a blue triangular indicator is shown next to the name of the currently displayed step in the script, and a black one next to the current step in processing. By indicating progress in this way it helps orient the user as the interface it is no longer inherently synchronized with the state of processing.

To avoid excessive recomputation times when stepping back and forth through a script, a data-caching mechanism has been implemented that can be used by the more time-consuming steps such as deconvolution (described below). To explain how this is implemented a brief overview of the object structure of the program will be needed. At the same time this will provide a useful reference for those interested in further implementation details.

The program framework is implemented in Objective-C, a small and simple set of extensions to the C language providing a dynamic, object-oriented language. Objective-C provides dynamic, run-time binding (late binding) of object messages, which allows new class frameworks and object instances to be added to the program at run-time or after it is already running. Objective-C also allows polymorphism, where a message (similar to a function/procedure call in procedural languages) can be sent to any object the sender is aware of that responds to the specified message, regardless of the receiving object type or inner workings.

Tool modules that are loaded into the program use both of these features. They must respond to a given set of messages, as defined by the API, to integrate into the program. These messages consist

primarily of tasks related to getting data into and out of the tool, telling the tool when to process the data, and storing of parameter values in a defaults database. A tool can be of any class as long as it adheres to this API (this is polymorphism). However, to ease the implementation burden, several classes that define the basic common functionality necessary for most tools are already implemented in the program (GenericToolCtrl, GenericTool, ResourceTool, ResourceToolCtrl). New tools can then be made to inherit from these, and need only to implement additional behavior that is unique to the tool.

This leads back to the caching example. By default, the tool superclasses do not cache the data. They implement a method shouldCache: that returns a NO value by default to the scripting mechanism. This means that each time the step is performed in a script, the tool's standard apply: method is called and its action performed on the data. However, a tool can easily override this by implementing a method shouldCache:, thereby overriding the superclasses method to return the value YES (YES and NO are standard 1/0 truth values defined in Objective-C). Then instead of being sent an apply:, the results of the previous apply: will be cached and reused.

There are two general types of tools defined by the provided tool superclasses. The first is a generic tool that simply uses a set of parameters input by the user to process the data in some way. As with any tool, the parameters chosen by the user are stored in a script as it is defined. A more complex type of tool is called a resource tool. This type of tool is designed to manage sets of more complex parameters (resources) than would typically be entered by hand, such as multicomponent transform matrices. The basic resource tool functionality includes a menu for organizing and accessing the resources, and a means for saving, deleting, and creating resources. Of course, the specific functionality, such as the type of resource being manipulated, must be defined by the tool itself.

The program allows the saving of defined scripts, and later recall of a saved script for application to a single or multiple lanes of trace data. To make this useful, tools are generally defined with as much flexibility as possible. This allows a tool to be included in a script that can be applied to a variety of data while still providing useful results. The details of a number of tool implementations will be provided in the next section.

One final aspect of the program structure worth noting is that a command line interface has been created, allowing the program to be used remotely through a telnet session, or incorporated in a shell-script. Much of the command line functionality mirrors the graphic user interface (GUI) functionality described above, allowing access to tools, scripts, and data, and application of scripts to data. It is not particularly useful for defining scripts, but once they have been defined and saved, they can then be accessed and applied to data this way.

Data Analysis Methods: Tools

Typically in fluorescence-based sequencing, data go through a number of processing steps before base-calling occurs. The processing steps act to transform the data set so that it is presented to both the user and/or base-calling algorithm in the standard processed trace format. This format consists of four channels of information, each channel a time series of dye concentration values (e.g., Fig. 3D). Bases are indicated by the presence of peaks in any of these four channels. The specific base is indicated by which channel a peak appears in. Although this is an easy form for the data set to be analyzed and visualized in, it requires substantial processing to effectively get it into this format from its original raw format created by selection of a lane on a gel image.

There are a variety of techniques that can be used to perform preprocessing of the trace data. From time to time new techniques are developed that need to be added to the program. For this reason, BaseFinder was designed so that tools can be developed separately from the main body of code for the program, and then added at run-time. The set of tools currently being maintained by our group for use with BaseFinder is summarized in Table 1. These tools reflect the common processing steps currently in use in the laboratory for analyzing data from both a commercial sequencing system (ABI) and an in-house scanning fluorescence-based sequencer. The details of the processing that these tools perform will be presented here roughly in the order that they are typically applied in the lab in scripts we have developed. Figure 3 shows a section of a typical trace before and as it undergoes the following described processing steps.

Preprimer Data Removal

In the typical sequencing run, data set collection begins soon after the sample is loaded and electrophoresis begins. Data collected before the first portion of the sample arrives at the detection region generally contains little useful information. In fluo-

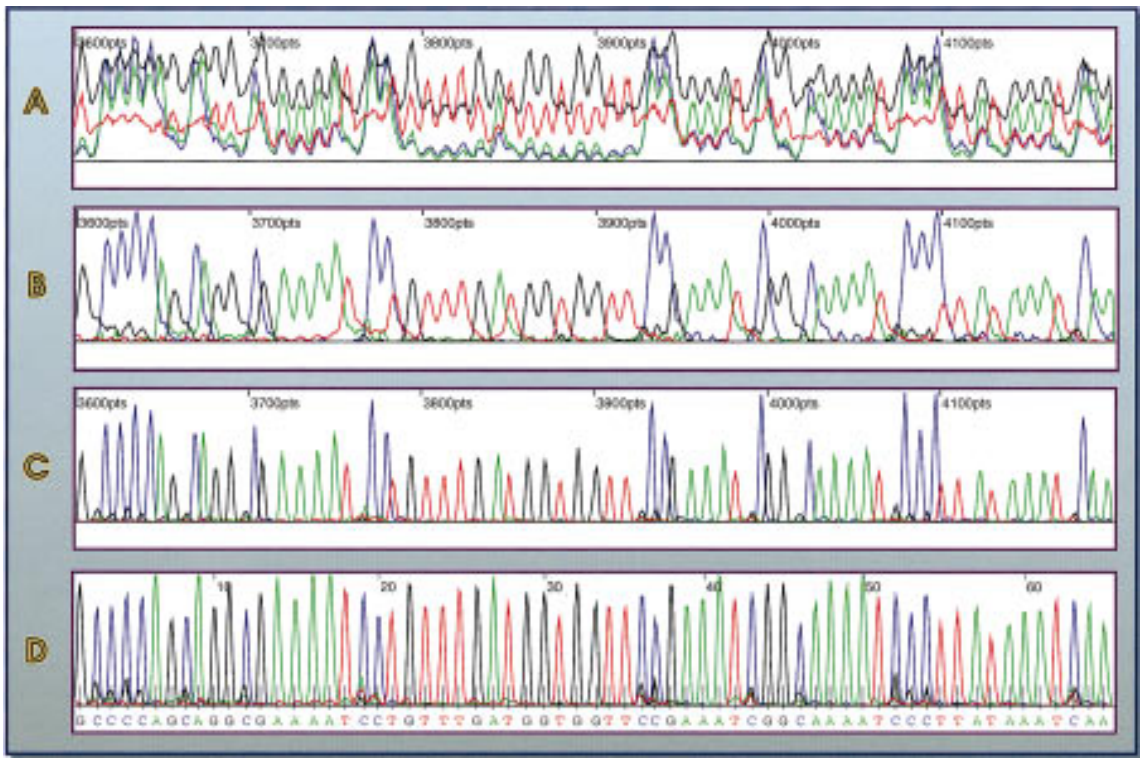


Figure 3 Shown is a short region of trace data extracted from an M13mp18 sequencing experiment performed on the scanning electrophoresis system developed in our laboratory. The data are shown in four different subsequent states of processing: (*A*) ''raw'' data as it looks after extraction from a lane by GelImager; (*B*) data after multicomponent transformation and noise filtering; (*C*) data after the addiction of a deconvolution step; and (*D*) data after the addiction of a polynomial mobility-shift correction, histogram cutoff adjustment, normalization, and base-calling.

rescent primer-based sequencing, the first portion of the sample that arrives at the detector is the ''Primer Peak'' because it contains dye primer material that is not attached to newly synthesized DNA. It is usually the largest and widest peak observed in the data stream, and thus can be easily recognized. Sequence data generally follows immediately after the primer peak.

It is useful for later processing to remove all data before the primer peak. This reduces the total quantity of data that must be processed, and it increases the accuracy of steps that perform automatic calibrations by removing data that does not contain any sequence information. A tool has been designed to locate the primer peak and remove all data before it. It works by searching for the largest peak in the data, and can be set to look for this in all data channels or in a single channel. Generally the primer occurs in all four channels, but the ability to use one channel only is good for cases such as an experiment with less than four fluorescent markers. The routine then finds the subsequent minimum in each of the four channels from the location of this maxima, and deletes all data before the farthest minima of the four channels from the location of the maximum. The only user-settable parameter for this tool is the choice between locating the primer peak in all channels or only one channel.

This algorithm does not work for dye terminator data, as there is no primer peak to detect. An alternative in that situation is the manual deletion tool, described below, which will perform a data cutoff at a fixed location. This should work adequately as long as the run conditions are kept constant, which in our experience will result in the fluorescence data starting consistently near a given point in the run. For a situation where the fluorescence data begins at a variable location from one run to the next, a new tool could be developed that detects the point at which significant fluorescence signal has emerged from the background and removes all data preceding that point.

Baseline Adjustment

Fluorescence from the polyacrylamide gel, ''junk'' in the sequencing reactions, and glass plates causes

Table 1. Commonly Used BaseFinder Tools and Typical Parameter Values

Tool name	Tool function	Typical parameter values
ToolDeletePrimer2	deletes all data before dye primer	none
FittedBaselineAdjust	removes baseline drift	width = 75
ToolMatrix2	transforms data from spectral to dye concentration	matrix appropriate for dye and instrument combination
ToolConvolve	smooths data to reduce noise	$\sigma = 1.5$; $M = 10$
DeconvolutionTool	improves resolution by reducing band spreading	iterations = 5, $\alpha = 0.25$, spread function polynomial based on typical data
ToolManualDeletion	removes selected regions of data, e.g., at end or beginning	none
ToolMobility3	corrects polynomials for data shifts due to dye mobility differences	shift values appropriate to dye set
ToolNormalize2	normalizes peak heights	none
ToolCutoffHisto	Removes highest and lowest points from the data to improve normalization	threshold = 2%
ToolBaseCalling	labels peaks with base-calls	see text

a substantial background signal to be observed in the data. This background signal can vary slowly with time. It is useful to remove this background and normalize the baseline before further processing occurs.

A tool has been implemented to perform background subtraction to remove these effects. It works by searching for the minimum signal intensity in successive windows of size *n*, where *n* is user selectable. It uses a linear interpolation between these minima found in successive windows, subtracting the resulting line connecting the adjacent minima from each data point between them. It does this in a piecewise fashion for each *n* point window in the data set. The parameter *n*, entered by the user, represents the minimum scale at which baseline variations are removed. Typical values used for *n* are 50–100, corresponding to ~1 min of data collection on our in-house sequencer. It is important to note that the window *n* must be set larger than the width of any of the expected peaks (including multiplet peaks) to avoid removal of desired signal data. In our experience this has not been a problem with the parameters specified above. The method works well for a wide variety of data sets. The window size is the only parameter for this tool.

Noise Filtering

Various aspects of both the separation instrument as well as the detection apparatus generate small amounts of noise in the data. This noise is particularly observable in the later portions of a data set where the signal from the sequencing reactions has substantially dropped off giving the noise a larger relative effect. It is useful to remove this noise to aid in subsequent processing and analysis. For this processing step, a tool that implements a Gaussian convolution is used. In effect, the convolution performs a weighted averaging of each point with its neighbors, that weighting function being provided by the Gaussian (equation 7, below). Essentially this performs a ''blurring'' of the data. To not average the data excessively, which would reduce interpeak resolution, a narrow Gaussian is used. Typically, the width parameter (σ) is chosen in the range 1–1.5, meaning that the range of the blurring is only a few points wide. A typical peak attributable to a fragment population in the data is in the range of 7–12 points wide, therefore a filter with such a narrow width does not cause excessive resolution loss. This tool tends to be quite effective at eliminating noise.

Some have suggested the use of a fast Fourier transform (FFT) to perform this filtering step. However, convolution with a Gaussian is an equivalent operation to applying a bandpass filter in the Fourier domain, which multiplies the frequency spectrum of the data set against a filter of Gaussian form with its origin at zero (Mathews and Walker 1970).

The only anticipated advantage to performing this step using FFT is the ability to set the filtering parameter based on a specific frequency, as opposed to width. However, in this case it does not seem that frequency is a substantially more intuitive parameter to use. Another possible filter method, which is in common use for spectroscopic applications, is that proposed in Savitsky and Golay (1964). This technique uses a convolution with a function that has the effect of approximating a localized least-squares fit. However, we have not yet experimented with this filter enough to warrant reporting on its usefulness for this application.

The parameters for this tool are σ, the Gaussian width, and W, the width of the window within which this Gaussian is applied. The latter is necessary because the Gaussian function has area (albeit very small) distributed to $\pm$infinity on the x-axis, so some cutoff must be used or it would be a nonterminating computation. Typical W values are 10–20 points.

Multicomponent Transformation

This step is one of the more important of those performed in preprocessing. The data collected is four channels representing spectral intensity, with each channel containing detected response values at a given wavelength, with four total wavelengths at which samples are collected. However, for analysis, the data are simpler to work with if they are transformed to represent the dye-linked DNA fragment concentration versus time for each of the four different dyes, with one channel per dye. The process of multicomponent transformation uses a specified 4×4 transform matrix that, when multiplied against each four-point vector of spectral data, results in a four-point vector of dye concentration (Frans and Harris 1985; Smith et al. 1987; Giddings et al. 1993). The BaseFinder tool implemented to perform this (ToolMatrix-2) consists of two modes: matrix definition and matrix application to data.

Matrix definition can occur in one of several ways. The theoretically ideal way is to know the exact spectral characteristics of each dye being used, as well as the excitation and detection properties of the sequencing system, and to use these values to create a 4×4 matrix of dye intensity versus detection wavelength values. This matrix is then inverted numerically and can be applied to the data.

In reality, this technique is not altogether practical because of the large time investment involved, and the existence of easier techniques resulting in equivalent performance. One of the alternate techniques is to use a typical data set as input to the MatrixFinder program, which determines automatically the best separation matrix. The algorithm that this program is based on was described by us in a separate publication (Yin et al. 1996). This program outputs a matrix that can then be used by BaseFinder's matrix application tool. More recently, Huang and colleagues (1997) described a means of quickly pinpointing the best matrix for any given data, given a reasonable initial estimate. This method appears to be intended to give better dye separations by tailoring the matrix for optimal application to each individual data set. We have not yet attempted to incorporate the technique, but implementation as a tool for BaseFinder appears to be straightforward.

A third method for matrix definition is to use the manual definition mode built into the Matrix tool for this purpose. This mode provides a facility for the user to select peaks in the spectral trace data, corresponding to known base labels, by clicking on them. The four spectral values at each selected peak are entered into a table along with the known base/dye label for the peak. The user must select at least one peak corresponding to each base (dye) present in the run. After selection of peaks has been completed, the spectral values for all peaks selected corresponding to each dye/base label are averaged together, and entered as a column in a 4×4 matrix. The resulting matrix is then numerically inverted and applied to the data set or stored for future use. This manual peak selection technique was in use before the automated algorithm was created, but is more tedious and not necessarily more accurate, and therefore is little used at present.

Currently, the most common technique used for matrix definition is application of MatrixFinder followed by manual fine tuning of the transform vectors to best align with the data clusters visually. The program allows the loading of four-column text-based tabular trace data, the visualization of three of its columns at a time on a pseudo three-dimensional display, and visualization of the associated vectors that the matrix-finding algorithm produces. In a typical display there will be several data clusters, each representing the spectrum of a particular dye, as shown in Figure 4. The coordinate system can be rotated arbitrarily using the mouse in the main window to allow for optimal viewing of the distinct clusters in a data set.

The goal is to produce vectors that best fall in the centers of these clusters. The automated algorithm does a good job of finding these, but manual tuning can often improve them by accounting for

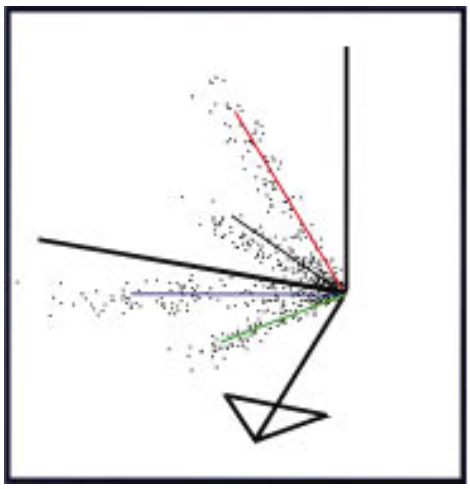


Figure 4 A simulated three-dimensional display of multispectral data as shown in MatrixFinder. The black lines are the *x, y,* and *z* axes. Each axis represents intensity at different detection wavelength. The dots are data points from a section of trace data plotted in the coordinate system, using three of the wavelength intensity values from each. In the MatrixFinder program, the display can be adjusted so that arbitrary axes correspond to arbitrary wavelengths, allowing viewing of the data in several different coordinate systems to compensate for the inability to display four dimensions at once. The four colored lines are the vectors found by the matrix-finding routine, representing the four principal dye spectra in the data. These are used directly as entries in a matrix for BaseFinder's multicomponent transform tool.

noise or other effects that cause nonsymmetrical distributions. Once the vectors are determined and fine-tuned, a corresponding matrix can be saved by the program, which can be used directly by BaseFinder's multicomponent tool.

Using BaseFinder's multicomponent tool to apply the resulting matrices to data is straightforward. The tool shares in common the functionality described for resource tools above. It allows the selection of any stored matrix from a menu of those stored in one of several predefined locations. It also allows renaming of matrices and deletion of matrices. Scripts defined that include this tool store a specification for the matrix name to be used as part of the script.

Two additional options provided by this tool are the ability to normalize the matrix before application and the option to remove all negative peaks after transformation of the data. Normalization simply finds the maximum diagonal value and normalizes all matrix entries by the multiplier necessary to make the largest entry have a value of one. This helps preserve the intensity values through the transformation process on roughly the same scale. Removal of negatives is important if there is either a slight imprecision in the transform matrix, or if there are spectral components present that are not accounted for by the matrix (e.g., because of noise or extraneous fluorescence), as transformation in such a case will often produce values falling below the baseline. This undesirable effect can be compensated for by removing all data below baseline. However, caution must be used in doing this, as it can not compensate for a poorly formed transform matrix. It is best when applying a new matrix to first look at the data without negative removal, and if there are regular dips below the baseline in one of the channels, this indicates a potential problem with the transform matrix that should be fixed before proceeding. If this is not observed, and assuming other aspects of the transform look good such as little overlap among distinct peaks, then negative removal can be turned on and the tool incorporated in a script.

In sum, the tool's parameters are choice of transform matrix, negative peak removal, and matrix normalization. Figure 3B shows a section of trace data that has undergone the noise filtering step followed by multicomponent transformation.

Deconvolution

The electrophoretic separation used in DNA sequencing introduces an effect called zone broadening (Giddings 1991). This effect reduces interpeak resolution by widening the fragment populations during separation. The zone-broadening process is generally equivalent in effect to convolution with a Gaussian-based function (one that is substantially wider than the Gaussian used for noise filtering described above). The reduced resolution attributable to band spreading causes reduced accuracy in base-calling. The effect becomes more prominent as the electrophoresis time increases, thereby contributing to the difficulty of accurately base labeling the later portions of data collected from a sequencing separation.

The deconvolution process partially reverses the effects of band spreading, thereby increasing resolution and base-calling accuracy. The process is sensitive to noise, and if overapplied can result in the appearance of false peaks, so it must be used with care. There are several ways of implementing deconvolution. For the deconvolution tool in BaseFinder, an iterative relaxation method (Jansson 1984; Barbee et al. 1995) was used because it is straightfor-

ward to implement. This approach requires an accurate model of the spread function σ as a function of time. Given such a model, the program attempts iteratively to estimate an ideal data set *I*, that when convolved with the spread function, results in the actual observed data set *S*.

In effect the deconvolution process is equivalent to performing a matrix inversion. The band spreading process can be described by

$$\vec{S} = \mathrm{M}\vec{I} \tag{1}$$

where the observed and ideal signals have been represented as vectors, and the matrix M is a matrix with each row *i* consisting of the spread function corresponding to sample *i* in the collected data. The deconvolution then works to essentially perform the following inversion:

$$\vec{I} = \mathrm{M}^{-1}\,\vec{S} \tag{2}$$

The key is the derivation of an accurate matrix M^{-1}. There are problems with doing it directly. It is a large, sparse matrix, so inversion is both time consuming and unstable. To get around this, the iterative approach is taken that was referred to above. This can be represented by

$$\vec{I}^{k+1} = \vec{I}^{k} + \lambda(\vec{S} - \mathrm{M}\vec{I}^{k}) \tag{3}$$

where

$$\vec{I}^{0} = \vec{S} \tag{4}$$

The term $(\vec{S} - \mathrm{M}\vec{I}^{k})$ is the difference between the observed data and the modeled data at iteration *k*. The value of *I* on successive iterations is determined by adding the error from the previous iteration to its value at the previous iteration. However, this can result in nonphysically possible negative numbers, and can also cause oscillation because of overshoot of the correct values. To avoid this, a relaxation function λ is introduced. This function guides the solution away from negative or excessively large values. The value of λ at point *i* in the data is given by

$$\lambda_i = 1 - e^{-\left(\frac{I_{ki}}{\alpha S_{\max}}\right)} \tag{5}$$

For each data point *i*, the error component of equation 3 is multiplied directly by each λ_i value. It is important to note that this is not a dot product. α is a parameter that controls the strength of the relaxation effect, and $S_{\max}$ is the maximal data value from *S*.

The zone-broadening function is not constant across a run—it changes with sample elution time. This means that the matrix M has an entry in each row representing the j^{th} zone-broadening function, and each column *i* corresponds to a particular sample number (elution time) in the input data. For DNA separations in polyacrylamide gel electrophoresis, a reasonable approximation of the zone-broadening function is a symmetrical Gaussian (Vohradský and Pánek 1993), in which the width of the Gaussian is the parameter that varies with sample number. In this case the matrix M has the elements

$$m_{ij} = A_j g_{ij} \tag{6}$$

where

$$g_{ij} = e^{\left(\frac{-(i-j)^2}{\sigma(j)}\right)} \tag{7}$$

and

$$A_j = \frac{1}{\sum_{i=1}^{n} g_{ij}} \tag{8}$$

Here g_{ij} represents the Gaussian zone-broadening function, and A_j normalizes each zone broadening function to an area of one, which causes area to be preserved in the data at each iteration (Barbee et al. 1995). The function σ(*j*) determines the amount of zone broadening that occurs as a function of sample elution time. Ideally it would be good to utilize a theoretical treatment to obtain this function. However, the theory of PAGE is very complex, making this a difficult task. Another approach is to model the function based on empirical measurements of peak spreading that occurs. We obtained data from several sequencing instruments and collected sets of Gaussian peak-fit data for these utilizing a nonlinear fitting function in *Mathematica* (Wolfram Research, Champaign, IL). Study of these data indicated that the zone-broadening function was nearly linear in the range from 0 to 1000 bases (~9500 data points at 6 sec/point) and could be modeled in a straightforward way with a polynomial function given by

$$\sigma(i) = a + bi + ci^2 \tag{9}$$

The constants *a*, *b*, and *c* are determined empirically, on the basis of a best fit to measured spread widths in an actual data set. To facilitate creation of the band-spreading function, a separate tool for BaseFinder was developed that provides a mechanism where peaks in the data can be selected by the user, and a Gaussian is fit to the selected peak to provide a σ value for the peak. Doing this multiple times, a table of values is created for position (time)

versus σ, and after a sufficiently representative sample has been selected, a least-squares fit is performed to determine the parameters *a*, *b*, and *c*. It appears that for a given instrument configuration this calibration does not change substantially from one run to the next.

The deconvolution tool has as its parameters the coefficients of the spreading function, as well as α and the number of iterations to be performed. Ideally it is best to choose a small value for α and a large number of iterations. However, each iteration involves a large number of computations, and therefore, is time consuming. By fine-tuning the parameters, a good balance can usually be achieved. Commonly used values for processing data from our in-house slab gel sequencer are $\alpha = 0.25$ and five iterations, which is often applied twice with an intervening noise filtering step. Figure 3C shows a section of trace data after typical deconvolution processing has been applied.

As a note, another approach to this process was proposed in Ives et al. (1994). Called Blind Homomorphic Deconvolution, this approach does not require knowledge of a specific zone-broadening function in advance of application, which is what the ''blind'' term refers to. We have not yet tested this technique in our laboratory, although it looks useful because it does not need any advance calibration.

Dye Mobility-Shift Correction

In multicolor fluorescence-based sequencing, the dyes used to mark the DNA fragments (either by primer or terminator linkage) affect the mobility of the DNA fragments as they migrate through the polyacrylamide gel. These mobility effects are dye specific. This results in a mobility shift, where two different fluorescently labeled DNA fragments of equal size migrate at slightly different rates and therefore, are shifted with respect to one another at detection by a small time Δt. On the basis of observational experience, this mobility shift is generally nonlinear in effect. The nonlinearity is particularly noticeable at early detection times, where the relative shift Δt between two dyes may change rapidly as a function of t. This is an effect of polyacrylamide gel electrophoresis that, like band spreading, suffers from lack of a straightforward theoretical treatment that could be used to correct the data.

If the effect is not removed, the base-calling step is made substantially more complicated because of varying degrees of overlap that occur because of the dye mobility differences. A tool was created to correct for this effect. Several shift functions were tested for efficacy, including quadratic, cubic, and linear with the addition of a $1/x$ term. From such empirical experiments, it appears that the cubic form provides the best fit based on residual values. However, the cubic form has the disadvantage that it is more sensitive to inaccuracies in the determination of the mobility-shift values to which the polynomial is fit, causing problems if the values are chosen without great care. In most cases the quadratic also provides a good fit. Because it is less sensitive to errors in the shift values collected it is the function most commonly used.

The mobility-shift tool provides a facility to allow the user to visually select mobility-shift values to which a polynomial is fitted by the program to create a mobility function. This is done by the user selecting a channel to be shifted by clicking on a button in the tool inspector, then moving the pointer to a selected location, typically corresponding with a peak, clicking on that location, and sliding that segment of data forward or back by the appropriate shift amount and releasing it. The amount of shift and the origin point for the shift is recorded (Δt versus t) in a table. Generally one data channel is chosen as the reference by the user and all shifts are performed against this channel. However, this choice does not matter to the program. Once the user has finished entering shift values, they are tabulated, and a polynomial is fitted to the shift values for each channel. Channels for which no shifts were performed are assigned a shift polynomial with zero coefficients.

As with other resource tools such as the multicomponent transform, mobility function sets (shift polynomial coefficients) can be named, stored, deleted, and selected from a menu for application to data. When applied within a script, the script will specify by name which mobility function is being used.

Signal Normalization

The scaling of intensity values varies greatly between different channels. It is useful for base-calling and visual analysis to normalize the data so that peak intensity values are more consistent between channels. Without normalization, base-calling may be impacted negatively. For example, a situation may arise where a real peak in one channel appears smaller than a noise artifact in another channel, thereby causing that location to be incorrectly base labeled. Originally a tool was developed that normalized the data according to the maximal response

value in each channel. However, this would fail to have the desired effect in many cases because a single outlying response value could cause an incorrect scaling of the rest of the data. Therefore, the tool was modified to normalize based on the average signal response values for each channel. Although this was an improvement, it still did not always produce the desired effect because of situations where base content was substantially different between channels. This is because a channel with a high base content would be scaled down more than one with a low base content, irrespective of the actual response values. To counter this, a further change was made to the tool base content adjustment of normalization. Before the normalization step, a preliminary base-calling step is applied. This is then used by the normalization step to adjust normalization according to base content (specifically, the normalization factor is divided by the base content percentage). If a previous base-calling step has not been performed, this tool performs the normalization assuming each channel has 25% of the total base content. The tool has no parameters.

Histogram Equalization

Another normalization that can be performed on the data is the removal of outlying point values, such as may occur at the top of an excessively high peak. As described in Giddings et al. (1993), the base-calling algorithm we are using attempts to correct for height variations within a channel and between channels. However, outlying points can throw the calibration off, making it less useful.

The histogram tool addresses this by sorting the data values according to intensity value and clipping the top and bottom x% of values, where x is a number specified by the user. The result is that a few large peaks will be chopped and have flat tops and the rest of the data will be scaled for improved analysis.

The parameters to the tool are the percent cutoff level, x, and a button that allows the choice of the cutoff being applied to maximal values, minimal values, and whether the effect should be performed per-channel or all-channels at once.

Base-Calling

The base-calling algorithms used in the current BaseCalling tool are described more fully in Giddings et al. (1993). It has not been changed substantially since the time that article was written, with the exception of two modifications. One of the modifications was separating the base-calling code from the main program into a distinct tool for modularity. The other modification consisted of reducing the confidence penalty for overlapping peaks, which occurred in the channels corresponding to G and C. This modification was made as part of the height assessment module as well as the spacing assessment module. It was done to allow for increased ability to read through GC compression regions, and appears to have had a slight positive effect on accuracy. However, it introduced a problem where false terminations occurring in one of the G or C channels and overlapping with a peak in the other of these channels results in an insertion error. Although this is a hindrance, experiments indicate there was still a net positive gain in accuracy from the modifications. We expect further adjustment to the code may remedy this trade-off.

All parameter choices for this tool remain as described in the aforementioned article. Figure 3D shows a section of trace data that has further undergone mobility-shift correction, normalization, histogram cutoff, and base-calling using typical parameters.

Trace Editing

One difficult issue encountered in our pilot sequencing projects has been trace editing. This consists of rejecting those portions of or entire traces where data quality is low. Degradations in trace quality include effects such as low resolution, low signal to noise, inconsistent peak spacing, false terminations, high noise level, baseline drift, and GC compressions. These effects result in lower accuracy in the base-calling process, which in turn can cause problems in the assembly process due to mis calls, insertions, or deletions. We use the Phred/Phrap software in assembly, which performs its own quality calculations on each base and accounts for base-specific quality problems in the assembly process. However, we have found that removal of low quality regions (or data sets) before entering the data into Phred/Phrap results in improved assembly results. This may be attributable to the sequencer platform and preprocessing used on this data, to which Phred is not specifically tuned. Using data from our instrument, Phred does a sufficient job of accounting for poor individual base-calls. However, it does not presently eliminate or account for entire regions where preprocessing and base-calling has been performed on poor quality data. This can happen in regions where true sequence-specific signal has dis-

sipated and the only remaining signal is attributable to chemical and instrumental noise. The preprocessing will affect these regions enough to present some small peaks that the base-calling algorithms will see as valid. Both Phred's and BaseFinder's algorithms will try to base-call them, producing incorrect sequence. Neither Phred nor our own base-calling algorithm adequately recognizes the situations where all peaks present are merely artifacts derived from noise. However, such regions can easily be recognized by eye because of their much decreased regularity in peak spacing, size, and shape. Currently such data regions are removed by hand in all of our sequencing projects.

Presently, we are working on a tool to address this automatically by eliminating data that are judged to be of low quality before the base-calling step occurs. The current approach is to do this by assessing the degree of variation present in different regions of the data for measures such as peak width and spacing. Preliminary results indicate these measures to be effective. Although not yet an existing tool, this is mentioned because it is an important part of the data analysis process that is undergoing active development.

RESULTS AND DISCUSSION

The program is now regularly being used for analysis of data from in-house production sequencing efforts. This real world use is the most important test of the program's functionality. Such usage had originally indicated a number of problems in the program, including bugs, insufficient preprocessing of the data, and lacking interface features. Over the past several years, most of these issues have been addressed.

Assessing the performance of the software in a meaningful way is a difficult issue. This is because the base-calling accuracy is affected by a great number of factors, including the preprocessing applied, the instrument configuration, the chemistries used, the type of DNA being sequenced and the base-calling parameters used. Further complicating this issue is the fact that most of the data collected in sequencing is from previously unknown sequence, and measuring the performance of algorithms without a known standard is difficult. Performance comparisons can be done using vector DNA; however, such tests are limited, as they often do not reflect accurately the performance of the software on real world sequencing data, which is generally more problematic and difficult at all stages of the sequencing process. In early experiments we discovered that software optimized for analyzing accurately vector sequence did not necessarily perform well in regular production sequencing. Some measure of performance can be obtained by comparing base-calls of the individual shotgun fragments to the assembled contig sequences from a sequencing project. However, any numbers resulting from such an effort are highly dependent on all of the parameters of the sequencing process, including the chemistries used, the DNA being sequenced, and the instrument used. This makes meaningful comparison to accuracy measures of the performance of other software packages difficult unless obtained under identical conditions.

To begin to address the accuracy issue, two metrics are presented here that in total provide an adequate picture of the software's performance, but should not be taken with too much weight individually because of the reasons mentioned above. One measure of accuracy is provided by experiments sequencing the M13mp18 vector. In these experiments accuracy of the base-calls produced by BaseFinder was compared to that of two other software packages. Experiments were performed with data collected on both our custom sequencing system (the ''UW Scanner'') as well as the ABI 377. A second measure was obtained by measuring the accuracy of 627 sequences from the shotgun phase of our pilot sequencing effort (described below) against the assembled contigs obtained in those efforts. These two accuracy measurements are reported along with some notes about our experience using the software in production to relay some of the benefits of the software as well as problems encountered that need to be addressed.

The M13mp18 vector-sequencing experiments were performed with the main goal of comparing the software's performance with that of the ABI and Phred packages. These were chosen because they are in widespread use, and were available to us. A more comprehensive comparison of other base-calling packages is beyond the scope of this paper. The results of these experiments indicate that with the definition of an appropriate processing script, BaseFinder performs competitively. The results also indicate the importance of the data processing that occurs before the base-calling step (preprocessing).

Accuracy results for processing and base-calling nine M13mp18 vector samples on the ABI 377 are shown in Figure 5. Figure 6 shows results using a similar methodology applied to 20 M13mp18 samples run on our in-house scanning sequencer system (M. Westphall, unpubl.). These are both cumulative plots of accuracy, and were collected as

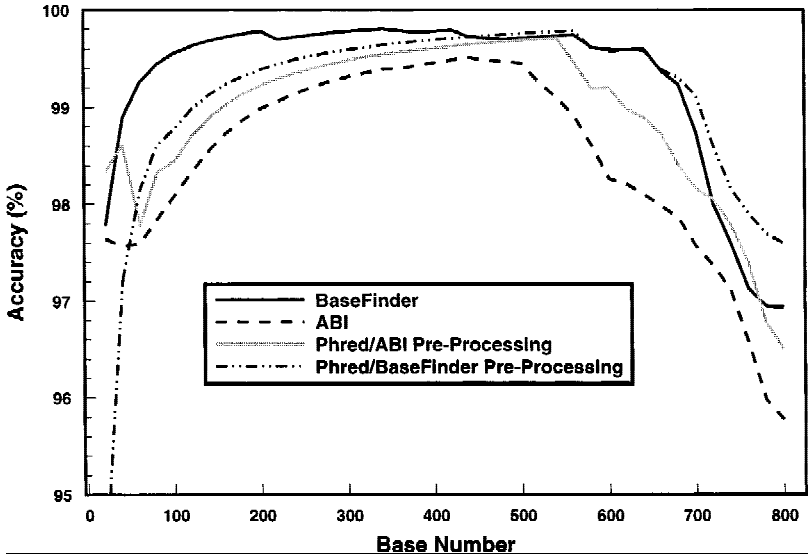


Figure 5 Plots of base-calling accuracy for four different combinations of trace processing and base-calling, each applied to a total of nine M13mp18 samples sequenced on an ABI 377 system. Results were calculated in 20-bp windows and averaged for each window over the samples. The plots represent cumulative accuracy to the base number shown on the *x*-axis.

described in the Methods section. Overall, the data illustrate that the software performs competitively in processing. There are several features worth noting. In Figure 5, the ABI base-called and the Phred Base Called data (using ABI preprocessing) have similar accuracy curve shapes, although the Phred curve is consistently higher. The BaseFinder curve has a somewhat different shape, exhibiting more linearity of performance across a broader region of the data. It is hypothesized that this has more to do with the different preprocessing applied to the data by BaseFinder than it has to do with the base-calling algorithm itself. This is corroborated by the curve for accuracy of Phred applied to the BaseFinder preprocessed (and base-called) ABI 377 data, also shown in the figure. Aside from changing the shape of the accuracy curve, BaseFinder's preprocessing increased the overall accuracy of Phred's base-calling by ~1% at a sequence length of 800 bases. Figure 6 also exhibits similarity between the Phred and BaseFinder accuracy curves, on which all data preprocessing was done by BaseFinder.

In these experiments, BaseFinder's current base-calling algorithm does better than both Phred and the ABI caller in regions earlier in the sequencing run where there are few compressions in the data sets analyzed, and continues to outperform the ABI base-caller in the later regions of a run. For the scanner data corresponding to Figure 6, the first difficult compression occurs in most of the data sets at about base position 545. From this point onward, there are a number of other compressions, each of which corresponds with the position on the plot where accuracy markedly drops. It is at about the point in the data where the compressions are observed at which Phred overtakes BaseFinder in performance (albeit only slightly), indicating that the better performance of the former may be attributable to smoother handling of difficult regions such as compressions. However, more important than these small differences in base-calling performance is the clear improvement in accuracy gained by using BaseFinder's preprocessing tools on the data, whether using BaseFinder or Phred for the fi-

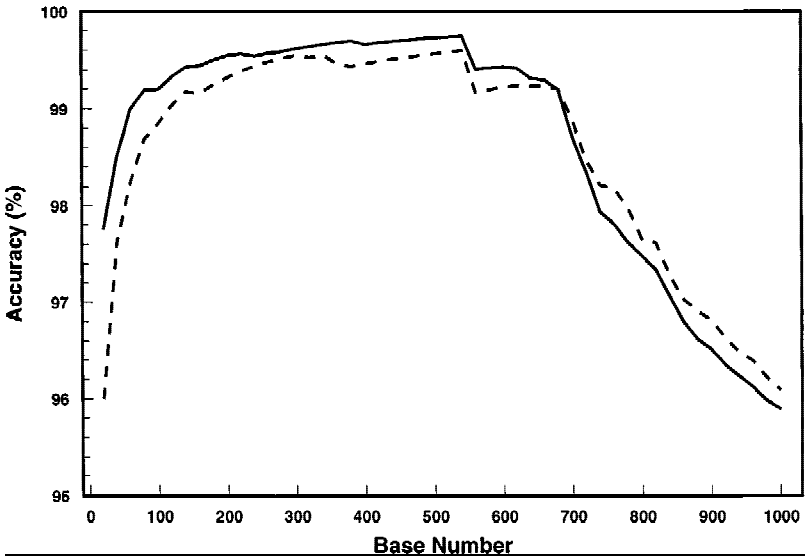


Figure 6 Plots of average base-calling accuracy for Phred's (broken line) and BaseFinder's (solid line) base-calling algorithms when applied to 20 data sets from sequencing M13mp18 on our laboratory's custom sequencing system. Both base-calling algorithms used preprocessing performed by BaseFinder. The plots represent cumulative accuracy, calculated in 20-bp windows, to the base number shown on the *x*-axis.

nal base-calling step. It can be seen from the figures that the curves for base-calling accuracy are strikingly similar for each preprocessing technique. This may indicate that each of these base-calling algorithms rely on similar features of the data, and that for performance to be improved, attention to the pre-base-calling signal processing steps is vital.

A more significant test of the software has been its application to a production-sequencing effort. This effort is aimed at sequencing a 500-kb region of human chromosome 19 in collaboration with the Department of Energy Joint Genome Institute. To date we have processed >2 million bases of raw sequence data that were collected on our custom built sequencer, using the software and the techniques described in the Methods section. These data consist of the shotgun sequence for five 38-kB cosmids. Of these, one has been completely finished and submitted to GenBank (accession no. AF025422) and another is in the final editing stages (no gaps remain in the assembled contig). The remaining three cosmids are in various stages of finishing. This software has also been used to analyze all the finishing reaction data collected, which includes a number of dye terminator reactions.

During this effort, the programs have performed very well. By the time the project was undertaken, earlier efforts had helped discover and eliminate bugs, and had helped the group understand what it takes to produce a script that performs effective data processing on a variety of data. Typical processing of a trace on a Pentium-Pro 200-Mh Z class computer takes 1 min per lane. A large portion of this time is spent in the deconvolution process. Although time consuming, the latter appears to account for an important portion of the gain in accuracy achieved by BaseFinder's preprocessing. It also appears that the deconvolution routines can be further optimized, and computers continue to get faster, therefore this is not perceived as a major stumbling block to the program's use in production sequencing. During these efforts, it was observed that typical useable read lengths are in the range of 700–750 bases. It should be noted that these long reads are not attributable to the software alone—the in-house scanning instrument has proved to be a very capable apparatus, as illustrated by the accuracy plots of Figure 6 compared to Figure 5.

Figure 7 shows the results of sequencing accuracy measurements obtained using sequences from the first two cosmids in the project, comprising a total of 627 individual sequences obtained from the shotgun phase of sequencing these cosmids. The individual sequences collected on the UW sequencer and processed by BaseFinder software were compared against the assembled and finished sequence data as further described in Methods. Phred base-calls of the same files were also compared against the contig sequences. Because all the trace files had been hand edited previously to remove the poorer quality regions of data at the end of each one, the sequences incorporated into the plot were of varying length. This is illustrated by the bar graph in the figure, showing the number of sequences that extend to each given read length. The average length of the BaseFinder sequences incorporated in the figure is 765 bases.

As expected, the results indicate a lower total accuracy in sequencing unknown stretches of DNA than is obtained in sequencing a well-known standard such as the vector region of M13. This indicates that tests of sequencing equipment and software using a standard such as M13 have only moderate utility in indicating real-world performance. There is some similarity in overall accuracy curve shape for the M13 sequences compared with the human chromosome 19 sequences; however, the M13 sequences have flatter, higher accuracy curves, which decline more steeply near the end of the useable read-length range. This difference in shape may be attributable to the hand editing performed on the human sequences, as the editing has culled the poor quality portions of sequence later in the runs, resulting in a more gradual sequence quality decline.

BaseFinder's and Phred's base-calling performances show similar trends to those observed above for the analysis of M13 vector sequence. BaseFinder's current base-calling algorithm yields greater accuracy than Phred for approximately the first half of the average read length, and then is overtaken by Phred. The maximum difference in performance, excluding the first 40 base region and the regions beyond 900 bases, is ~1%. Although not a large number, it is significant enough in the context of large-scale sequencing to warrant the continued use of Phred following BaseFinder's preprocessing for our sequencing efforts. It may be worth additional study to determine why BaseFinder's algorithm does better in earlier regions of a sequence and Phred in the latter portions, and work toward synthesizing a hybrid of both approaches. In any case, BaseFinder provided all of the preprocessing of the trace data for input to either base-calling algorithm, and based on the M13 sequencing accuracy experiments, it is a solid platform for these preprocessing steps.

In the process of getting the collected data from

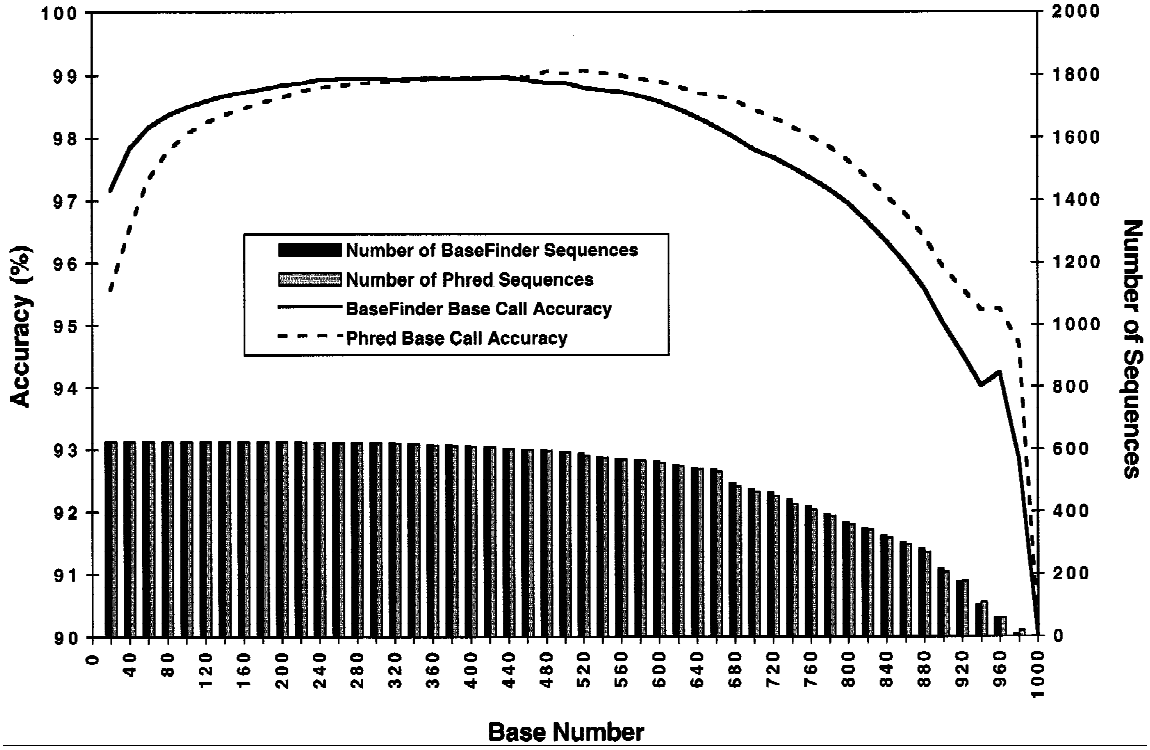


Figure 7 A plot for averaged accuracy for sequences obtained in sequencing two cosmids from human chromosome 19 using the UW scanner system. Accuracy comparisons for individual sequences were made against the assembled, finished sequence of the cosmids for BaseFinder's and Phred's base-calls (both of which used BaseFinder's preprocessing of the trace data). The line plots represent cumulative accuracy relative to read length for the two cosmids. The bar graphs show the number of sequences long enough to be incorporated into the accuracy calculation for each read length. The *y*-axis scale on the *left* corresponds to accuracy; the scale on the *right* to the number of sequences used in calculation of accuracy.

the UW electrophoresis instrument to the assembly project there remain two manually intensive time-consuming steps. These are lane verification and retracing on the gel image, and data quality verification combined with manual trace editing before inclusion of sequences in the assembly project. These issues are being addressed by ongoing work. To aid in lane tracking, internal markers are being placed in each sequenced sample to mark the starting location of each lane. In addition, enhanced image-processing steps (edge sharpening and multicomponent transformation of the data) are being incorporated into our version of the Getlanes program to further highlight each lane to improve lane tracking accuracy. To remove the manual steps involved in base-calling, a trace-editing tool is being developed (as discussed earlier). As well, a new base-calling algorithm is being developed, designed to produce bases with assigned quality values consistent with the Phred value, thus eliminating the need to run the data through Phred before assembly.

The area in which the program's base-calling exhibits the most noticeable deficit is handling GC compressions, polymorphisms, and false terminations. These situations are cases where special exception handling must occur, although even with special processing GC compressions may be difficult to handle accurately. As discussed, some effort has been made to tune base-calling to account for these effects, but further tuning is necessary to achieve optimal performance in these difficult regions. Another area of potential improvement for the base-calling is in the confidence assignments. These assignments are intended to provide a measure of the accuracy, on a range from 0 (least accurate) to 1 (most accurate) for base-calls. As described in the 1993 paper (Giddings et al. 1993), these measures are based on consistency of spacing, peak height, and width. Although they correspond generally with sequence quality, the values are not scaled in a particularly meaningful way, which has inhibited their use in assembly programs such as Phrap. This is one area that we hope to improve in the future.

CONCLUSION

The software presented has proven very valuable in efforts in our laboratory at both development of new sequencing technologies, as well as the application of those technologies to sequencing efforts. It continues to undergo improvement, and interest appears to be growing by outside parties in using the software for their own analysis tasks. Additional work is ongoing to improve the preprocessing capabilities and stability, as well as the performance of the base-calling algorithm with the goal of fully hands-off, automated processing.

METHODS

Data for the M13 vector sequencing experiments were collected as follows. Ten reaction mixtures were prepared with M13mp18 template, using Amersham FET primers and thermosequenase and following the cycle-sequencing protocols standard for those kits. Enough purified reaction product was produced to perform several analyses of each mix. One sequencing run was performed on our in-house slab-gel based system. In alternating lanes, 0.5 µl and 1 µl of each product were loaded, producing a total of 20 lanes. A second sequencing run was performed on an ABI 377, by a local for-contract sequencing service, using the 36-cm glass plate set and the standard protocol specified for that system, producing a total of 10 more lanes of data.

The data were then analyzed as follows. The ten samples collected on the ABI 377 were processed in four ways to produce sequences for comparison: (1) ABI software only; (2) ABI software followed by Phred; (3) BaseFinder, starting from the raw lane data produced by the ABI lane-finding software (meaning BaseFinder's preprocessing was used for these); and (4) Phred using the BaseFinder preprocessed and base-called data (saved in SCF format). A script was defined in BaseFinder for application to the 377 data, and was then applied unchanged to all 10 samples. All resulting sequences were compared to the GenBank M13mp18 sequence (accession no. X02513) using the GCG "best-fit" routine (Genetics Computer Group 1994). A custom program combined with a spreadsheet was used to analyze the output of the "best-fit" program and produce sequence accuracy numbers up to any arbitrary position N in the test sequence. This number is calculated as

$$\text{Accuracy} = 100 \times \left(1 - \frac{G + M + \frac{1}{4}C}{N}\right) \qquad (10)$$

where G is the number of gaps in either the test or the reference sequence up to position N (because of an insertion or deletion on the test strand), M is the number of mis-calls to position N, C is the number of Ns (no-calls) that do not align with a gap on the reference strand to position N (meaning an insertion, that is penalized fully), and N is the total number of aligned base-pair positions counted to (including gaps). These accuracy numbers were calculated in successive 20 base-pair windows and made into averaged, cumulative plots of accuracy (Fig. 5). Cumulative plots were used to both provide a better indication of overall performance at any particular sequence length, as well as to produce a smoother graph. Plots of local accuracy tend to vary greatly, particularly when the software runs into problematic areas such as compressions. It should be noted that one of the ABI 377 samples was very problematic because of low signal and very high noise, possibly attributable to a lane-tracking error, therefore this sample was rejected (as it would have been after visual inspection if it was a part of our production-sequencing effort). We did not have access to the gel image file to determine the reasons for the problem, but the sample was significantly skewing results in a negative direction for all base-callers. Therefore, the results reported are based on only 9 of the 10 samples loaded and processed on the ABI 377 system.

Analysis was performed similarly for the data from the custom sequencer, using a total of 20 data sets, which were preprocessed by BaseFinder, and then base-called by Phred (which requires processed trace data) and BaseFinder. ABI processing and base-calling was not tested, as the system provides no means for import of data generated on a non-ABI platform. Again, the accuracy was measured in 20-bp windows and averaged each window for all runs. Cumulative accuracy plots for these data are shown in Figure 6.

For the production sequencing efforts, the methods were as follows. DNA Separation and Detection in all of our sequencing projects is conducted solely on our own custom four-color, fluorescence-based electrophoresis instrument. The data (gel image) generated by this instrument is collected on a Pentium PC running Windows 3.1. The generated data file is transferred to the analysis platform (Intel processor running OpenStep/Mach) through FTP for processing. The gel image is first viewed with GelImager to ensure that the data was transferred successfully. Upon verification, the data file is processed by Getlanes (Cooper et al. 1996), to determine lane locations. The lane-marking information generated by Getlanes is read into GelImager and superimposed over the gel image for lane-tracking verification. Any mistracked lanes are manually corrected. Once all the lane tracking has been verified, the trace files for each lane are saved.

The extracted lane files are loaded into BaseFinder for preprocessing and base-calling. The first step is to load the script appropriate for the run conditions (dye terminator, dye primer, gel concentration, etc.) under which the data was collected. This script is applied to all lanes or trace files that have been selected for processing. Upon completion of the analysis, each trace is inspected manually. The traces that contain failed reactions, dropped bases, or too weak a signal to be properly processed are removed. At the same time any data points that extend beyond the visibly assessed reaction length are removed to prevent interference in the assembly processes. This is done because data collection conditions are such that read lengths of 1000+ bases may be obtained. However, the enzymology does not extend to >1000 bases on every clone, and in many cases will die out well before the data collection is terminated. In these early terminating reactions, the data will correspondingly drop to noise at a certain point, and data after this needs to be removed for optimal assembly as explained previously. The base-called data is saved out in SCF format, which can be read by most of the popular analysis/assembly programs.

Assembly of the shotgun data is performed using the Phrap assembly program (P. Green, unpubl.). To use fully the quality measures generated and utilized by Phrap in the assembly process, the BaseFinder-generated SCF files are processed with Phred for editing of the base-calls and assignment

of quality measures to each base. The Phred-assigned quality values and base-calls are then used by Phrap for assembly of all the data. The traces that go into the assembly of a contig can be viewed with Consed (Gordon et al. 1998). Upon completion of six to seven times redundancy in shotgun data, the assembled project is viewed with Consed to locate regions of ambiguities, as well as to examine contig ends. This information is then used to determine a finishing strategy for gap closure and removal of ambiguities. As a last step, each base in the final consensus is manually verified with the individual traces from which the consensus was determined. After this final editing, the data are then annotated and submitted to a public database.

Data from the two currently finished cosmids were analyzed to provide a measure of overall sequencing accuracy. Only the sequences from the initial shotgun phase of the project were used, as the finishing reactions used a different chemistry (dye terminator) and were of more variable quality (generally lower). In addition, those sequence files containing stretches of religated sequence from either the vector or the target DNA were removed before comparison. The set of sequence files used in collection of the statistics comprised a total of 627 sequences, 325 of which are from cosmid 1 and 302 from cosmid 2. The original SCF files containing the base-finder-processed traces and base-calls from the shotgun phase, which were saved as standard procedure during the sequencing project, were loaded back into BaseFinder to extract and save only the base sequences (as text files). A PERL script was developed to invoke the GCG best-fit routine to compare each sequence to the appropriate region of the correct finished cosmid sequence. To improve processing time, the assembly project file produced by Phrap for each cosmid was used to pinpoint the region in the cosmids against which to align each sequence. Similarly, the base-calls produced by Phred were compared against the finished cosmid sequences and statistics derived for the same set of sequences.

The resulting output files from best-fit were processed in the same way as for the M13mp18 samples using a program written for this purpose (source code provided on request). Because of the manual editing that was performed during the project on the trace files, the sequences are of varying length. Therefore, the cumulative accuracy was calculated for each 20-bp window using only those sequences that were long enough to reach the window. The number of sequences incorporated into the accuracy calculations at each point are shown in Figure 7 to give an idea of the statistical weight of the accuracy measurement based on how many sequences are being averaged.

ACKNOWLEDGMENTS

We thank Dr. David Rank for his assistance and patience in testing and debugging the programs. Funding for this work has been provided by National Human Genome Research Institute grant HG00321.

The publication costs of this article were defrayed in part by payment of page charges. This article must therefore be hereby marked ''advertisement'' in accordance with 18 USC section 1734 solely to indicate this fact.

REFERENCES

Barbee, K.A., J.A. Morrow, and S.C. Meridith. 1995. Deconvolution of gel filtration chromatographs of human plasma lipoprotiens. *Anal. Biochem.* 231: 301–308.

Berno, A. 1996. A graph theoretic approach to the analysis of DNA sequencing data. *Genome Res.* 6: 80–91.

Carrilho, E., M.C. Ruiz-Martinez, J. Berka, I. Smirnov, W. Goetzinger, A.W. Miller, D. Brady, and B.L. Karger. 1996. Rapid DNA sequencing of more than 1000 bases per run by capillary electrophoresis using replaceable linear polyacrylamide solutions. *Anal. Chem.* 68: 3305–3313.

Cooper, M.L., D.R. Maffitt, J.D. Parsons, L. Hillier, and D.J. States. 1996. Lane tracking software for four-color fluorescence-based electrophoretic gel images. *Genome Res.* 6: 1110–1117.

Dear, S. and R. Staden. 1992. A standard file format for data from DNA sequencing instruments. *J. DNA Sequenc. Map.* 3: 107–110.

Ewing, B., L. Hillier, M.C. Wendl, and P. Green. 1998. Base-calling of automated sequencer traces using Phred. I. Accuracy assessment. *Genome Res.* 8: 175–185.

Frans, S.D. and J.M. Harris. 1985. Selection of analytical wavelengths for multicomponent spectrophotometric determinations. *Anal. Chem.* 57: 2680–2684.

Fleischmann, R.D., M.D. Adams, O. White, R.A. Clayton, E.F. Kirkness, A.R. Kerlavage, C.J. Bult, J.-F. Tomb, B.A. Dougherty, J. M. Merrick et al. 1995. Whole-genome random sequencing and assembly of *Haemophilus influenzae* Rd. *Science* 269: 496–511.

Garner, H.R., B. Armstrong, and D.A. Kramarsky. 1992. High throughput DNA preparation system. *Genet. Anal. Tech. Appl.* 9: 134–139.

Genetics Computer Group. 1994. *Program manual for the Wisconsin package,* Version 8. Genetics Computer Group, Madison, WI.

Giddings, J.C. 1991. *Unified separation science,* pp. 193–195. John Wiley & Sons, New York, NY.

Giddings, M.C., R.L. Brumley Jr., M. Haker, and L.M. Smith. 1993. An adaptive, object oriented strategy for base calling in DNA sequence analysis. *Nucleic Acids Res.* 21: 4530–4540.

Golden, J.B., D. Torgersen, and C. Tibbetts. 1993. Pattern recognition for automated DNA sequencing: I. On-line signal conditioning and feature extraction for basecalling. In *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology* (ed. L. Hunter, D. Searls, and J. Shavlick), pp. 136–144. AAAI Press, Menlo Park, CA.

Gordon, D., C. Abajian, and P. Green. 1998. Consed: A graphical tool for sequence finishing. *Genome Res.* 8: 196–202.

Huang, W., Z. Yin, D.R. Fuhrmann, D.J. States, and L.J. Thomas Jr. 1997. A method to determine the matrix in

4-dye fluorescence-based DNA sequencing. *Electrophoresis* 18: 23–25.

Hunkapiller, T., R.J. Kaiser, B.F. Koop, and L. Hood. 1991. Large-scale and automated DNA sequence determination. *Science* 254: 59–67.

Ives, J.T., R.F. Gesteland, and T.G. Stockham Jr. 1994. An automated film reader for DNA sequencing based on homomorphic deconvolution. *IEEE Trans. Biomed. Eng.* 41: 509–519.

Jansson, P.A. 1984. Modern constrained nonlinear methods. In *Deconvolution with applications in spectroscopy* (ed. P.A. Jansson), chapter 4. Academic Press, New York, NY.

Kostichka, A.J., M. Marchbanks, R.L. Brumley Jr., H. Drossman, and L.M. Smith. 1992. High speed automated sequencing in ultrathin slab gels. *BioTechnology* 10: 78–81.

Luckey, J.A., H. Drossman, A.J. Kostichka, D.A. Mead, J. D'Cunha, T.B. Norris, and L.M. Smith. 1990. High speed DNA sequencing by capillary electrophoresis. *Nucleic Acids Res.* 18: 4417–4421.

Mathews, J. and R.L. Walker. 1970. Application of integral transforms and The bionomial, poisson and gaussian distribution. In *Mathematical methods of physics,* pp. 110–113 and 377–384, respectively. Addison Wesley, Reading, MA.

Mathies, R.A. and X.C. Huang. 1992. Capillary array electrophoresis: An approach to high-speed, high-throughput DNA sequencing. *Nature* 359: 167–168.

Savitsky, A. and M.J. Golay. 1964. Smoothing and differentiation of data by simplified least squares procedures. *Anal. Chem.* 36: 1627–1639.

Smith, L.M. 1993. Automated DNA sequencing: A look into the future. *Cancer Detect. Prevent.* 17: 283–288.

Smith, L.M. and L.E. Hood. 1987. Mapping and sequencing the human genome: How to proceed. *BioTechnology* 5: 933–939.

Smith, L.M., R.J. Kaiser, J.Z. Sanders, and L.E. Hood. 1987. The synthesis and use of fluorescent oligonucleotides in DNA sequence analysis. *Methods Enzymol.* 155: 260–301.

Swerdlow, H., B.J. Jones, and C.T. Wittwer. 1997. Fully automated DNA reaction and analysis in a fluidic capillary instrument. *Anal. Chem.* 69: 848–855.

Vohradský, J. and J. Pánek. 1993. Quantitative analysis of gel electrophoretograms by image analysis and least squares modeling. *Electrophoresis* 14: 601–612.

Wilson, R.K., C. Chen, N. Avdalovic, J. Burns, and L. Hood. 1990. Development of an automated procedure for fluorescent DNA sequencing. *Genomics* 6: 626–634.

Yin, Z., J. Severin, M.C. Giddings, W. Huang, M.S. Westphall, and L.M. Smith. 1996. Automatic matrix determination in four dye fluorescence-based DNA sequencing. *Electrophoresis* 17: 1143–1150.

Received December 17, 1997; accepted in revised form April 21, 1998.