An Essential Guide to the Basic Local Alignment Search Tool
BLAST
O'REILLY
Ian Korf, Mark Yandell & Joseph Bedell

BLAST

By Joseph Bedell, Ian Korf, Mark Yandell

Publisher: O'Reilly
Pub Date: July 2003
ISBN: 0-596-00299-8
Pages: 360

BLAST (Basic Local Alignment Search Tool) is a set of similarity search programs that explore all of the available sequence databases for protein or DNA. BLAST is the only book completely devoted to this popular and important technology and offers biologists, computational biology students, and bioinformatics professionals a clear understanding of this program. This book shows you how to get specific answers with BLAST and how to use the software to interpret results. If you have an interest in sequence analysis this is a book you should own.

- Table of Contents
- Index
- Reviews
- Examples
- Reader Reviews
- Errata

BLAST

By Joseph Bedell, Ian Korf, Mark Yandell

Publisher: O'Reilly
Pub Date: July 2003
ISBN: 0-596-00299-8
Pages: 360

Copyright

Published by O'Reilly & Associates, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Forward

Reading a book such as this brings home how much BLAST-now in its teenage years-has grown, and provides an occasion for fond reflection. BLAST was born in the first months of 1989 at the National Center for Biotechnology Information (NCBI). The Center had been created at the National Institutes of Health in November 1988, by an act of the U.S. Congress, to foster the development of a field that then had no widely accepted name, but which has since come to be known as "Bioinformatics." In early 1989, David Lipman, my post-doctoral advisor, who at the time was perhaps best known as a codeveloper of the FASTA program, was appointed director of NCBI. On the first of March we moved into new offices at the National Library of Medicine.The NCBI was small, but had large ambitions, and already a number of friends. Several of these well-wishers made it a point to drop by for a visit. Gene Myers, a computer scientist then at Arizona, arrived during a week in which Science was hyping a special-purpose computer chip for sequence comparison. He and David, software partisans both, were unimpressed and over dinner resolved to do better. Their original idea was to find not subtle sequence similarities, but fairly obvious ones, and to do it in a flash. Gene pursued a rigorous approach at first, but David, with a fine Darwinian wisdom, was willing to settle for imperfection. If one were to gamble, what kind of match could one expect a strong alignment to contain? Detailed algorithmic and code development on BLAST by Webb Miller-later to be joined by Warren Gish-had hardly begun before Sam Karlin, a Stanford mathematician, came calling. I had approached him a few months earlier with a conjecture concerning the asymptotic behavior of optimal ungapped local sequence alignments. Since then, he had spun this conjecture into a beautiful theory. Now, for the first time, rigorous statistics were available for alignment scoring systems of more than academic interest, and the essential nature of amino acid substitution matrices also began to come into clear focus. This theory dovetailed perfectly with the work that had just started on BLAST: both informing the selection of its algorithmic parameters, and yielding units for the alignment scores produced.

Although David chose BLAST's name as a bit of a pun on "FASTA" (it was only later that I realized "BLAST" to be an acronym), the new program was never intended to vie with the earlier one. Rather, the idea was to turn the "threshold parameter" way up, to find undoubted homologies before you take more than one sip of coffee. It surprised us all when BLAST started returning most weak similarities as well. Thus was born a sort of friendly competition with Bill Pearson's and David's earlier creation. From the start, BLAST had two major advantages to FASTA and one major disadvantage. In the plus column, BLAST was indeed much the faster, and it also boasted Sam's new statistics, which turned raw scores into E-values. However, BLAST could only produce ungapped local alignments, thereby often eliding large regions of similarity and sometimes completely missing weak alignments that FASTA, in its most sensitive but slowest mode, was able to find. These points of comparative advantage were healthy for both programs. In time, FASTA fit its scores to the extreme value distribution, yielding reliable statistical evaluations of its output. And by the mid '90s, Warren Gish's WU-BLAST from Washington University, and NCBI's BLAST releases, introduced gapped alignments, using differing algorithmic strategies. The result, at least for protein sequence comparisons, is that BLAST and FASTA have converged in many important ways, although there still remain significant differences.

The programs comprehended by the name "BLAST" have multiplied astonishingly in the nearly 15 years since the first one was conceived. Learning the best way to use these various programs for research can be a challenge, and this book is a significant aid.While BLAST's developers have done their best to make the programs' default behavior the most generally applicable, a sophisticated user still has many issues to consider.

To achieve speed, BLAST is a heuristic program. It isn't guaranteed to find every local alignment that passes its reporting criteria, and there are an array of parameters that control the shortcuts it takes.With the introduction of gapped alignments, the programs' complexity increased, as did the number of parameters that influence BLAST's tradeoff of speed and sensitivity. In a certain sense, however, these mechanics are the least important for a user to understand because, except for the occasional appearance or disappearance of a weak similarity, they don't greatly effect the programs' output. Perhaps of more importance is an understanding of attendant matters that are relevant to the effective use of any local alignment search method, such as the filtering of "low-complexity" sequence regions, the proper choice of scoring systems, and the correct interpretation of statistical significance. This book deals with these and many other matters, and nicely combines theoretical considerations with practical advice informed by these considerations.

The BLAST programs have been the fruit of much hard work by scores of talented programmers and scientists. This work continues, linking BLAST output to other databases, improving alignment formatting options, refining the types

Preface

The second half of the 20th century was witness to incredible advances in molecular biology and computer technology. Only 50 years after identifying the chemical structure of DNA (1953), the sequence of the human genome has been determined and can be downloaded to a computer small enough to fit in your hand. The pace of science can be truly dizzying. So what do you do when you literally have the book of life in the palm of your hand? Well, you read it of course. Unfortunately, it's much easier to read the book of life than to understand it, and one of the great quests of the 21st century will be unraveling its mysteries. One particularly fruitful approach to deciphering the book of life has been through comparative studies, such as those between mouse and human.

Comparisons between the human and mouse genomes show how little has changed since humans and mice last shared a common ancestor around 75 million years ago. Very few genes are unique to humans or mice, and in general the genes are more than 80% identical at the sequence level. However, genes account for a small fraction of these genomes and the majority of sequence is not recognizably similar. This is where BLAST, the Basic Local Alignment Search Tool, comes in. BLAST is useful for finding similarities between biological sequences, be they DNA, RNA, or protein. Sequence similarity is often an indication of conserved function, and you can use comparative sequence analysis to understand biological sequences in much the same way that ancient Greeks used comparative anatomy to understand the human body or that linguists used the Rosetta Stone to understand Egyptian hieroglyphs.

Audience for This Book

People interested in BLAST come from many disciplines including biology, chemistry, computer science, law, mathematics, medicine, physics, etc. One reason for this is that knowledge of genes and genomes is becoming increasingly useful in a variety of settings. Another reason is that bioinformatics is this century's rocket science. Researchers from many disciplines are being drawn into its fascinating and rapidly growing orbit. So if you've recently become interested in bioinformatics, understanding BLAST is a great place to start. And if you're already a bioinformatics student or professional, this book can help you get more out of BLAST.

Structure of This Book

This book is divided into six parts: An Introduction to BLAST, Theory, Practice, Industrial-Strength BLAST, Reference, and the Appendixes. The quick start guide in Chapter 1 is the best place to begin if you've never run BLAST before. You won't need sophisticated hardware or software, just a web browser connected to the Internet. In Part II, we begin by exploring the molecular biology, computer science, and statistics that form the foundation of BLAST searches. We then describe the BLAST algorithm in detail. You will find that a sound theoretical understanding is essential when you put BLAST into practice. In Part III, we present practical advice to help you design and interpret BLAST experiments intelligently and efficiently. Whether you're a complete novice or a seasoned pro, you'll find the tutorials and protocols a valuable resource. Part IV discusses using BLAST in a high-throughput setting where the goal is to get as much BLAST as possible for your buck. Here, we integrate the information usually found scattered among systems administrators, database administrators, and advanced BLAST users into a few sensible chapters. Part V contains reference chapters for NCBI-BLAST and WU-BLAST with detailed descriptions of each parameter.

Here's a chapter-by-chapter breakdown:

Part I

Chapter 1, gives a quick introduction to BLAST by exploring Internet search pages.

Part II

Chapter 2, gives some background molecular and evolutionary biology to help you understand why biological sequences are similar to one another.

Chapter 3, explains how global and local sequence alignment works and describes common algorithms for aligning sequences of letters.

Chapter 4, explains how scores are used to determine the best alignmentand discusses the statistical significance of sequence similarity in a database search.

Part III

Chapter 5, discusses BLAST itself. Understanding the theoretical framework of the BLAST suite of programs will help you design and interpret BLAST experiments and give you a foundation for troubleshooting when your search produces unexpected results.

Chapter 6, explores the standard format of the BLAST report.

Chapter 7, shows how to calculate the numbers in a BLAST report and use this knowledge to better understand the results of a BLAST search.

Chapter 8, is a summary of the previous seven chapters as well as the authors' expertise, and is designed to help you get the most from your BLAST searches.

Chapter 9, contains "recipes" for the most common BLAST searches; it describes what to do and why to do it.

Part IV

Chapter 10, shows how to install NCBI-BLAST and WU-BLAST software on your own computer. This is necessary if you want to use BLAST in a high-throughput setting or develop specialized applications.

Chapter 11, shows how to create and maintain BLAST databases—one of the most neglected yet important aspects of using BLAST.

A Little Math, a Little Perl

Certain parts of this book are mathematical or algorithmic in nature, so you will find various `equations and computer programs throughout the book. If these notations are unfamiliar to you, don't panic. To make this book accessible to a general audience, we have included graphical examples and descriptive text along with the equations. The programming examples are written in Perl, one of the most popular programming languages and one that has an especially strong following in bioinformatics. While we could have relied on pseudocode for our examples, using a real language means that you can run the programs on your own computer and edit them as you wish.

Conventions Used in This Book

The following conventions are used in this book:

Constant width

Used for Perl programs, parameters, and BLAST output

Italics

Used for program names, databases, for emphasis, and for new terms where they are defined

URLs Referenced in This Book

For more information about the URLs referenced in this book and for additional material about BLAST, see this book's web page, which is listed in the next section.

Comments and Questions

Please address comments and questions concerning this book to the publisher:
O'Reilly & Associates, Inc. 1005 Gravenstein Highway NorthSebastopol, CA 95472(800) 998-9938 (in the United States or Canada)(707) 829-0515 (international or local)(707) 829-0104 (fax)

There is a web page for this book, which lists errata, examples, or any additional information. You can access this page at:
http://www.oreilly.com/catalog/blast

To comment or ask technical questions about this book, send email to:
bookquestions@oreilly.com

For more information about books, conferences, Resource Centers, and the O'Reilly Network, see the O'Reilly web site at:
http://www.oreilly.com

Acknowledgments

As a group, the authors would like to thank O'Reilly & Associates for their patience and support, and especially their editor Lorrie LeJeune. The book owes a lot to its technical reviewers: Scott Markel, Tony Palombella, and staff of the NCBI. Special thanks go out to Scott McGinnis, Tom Madden, and Stephen Altschul for all their insightful comments.

Ian

I thank my wife Karen (whose critical comments improved the readability of the book) and daughter Zoe for putting up with the extra hours required to write this book. (Sorry, I had no idea it was going to take this much time.) I'd also like to thank my former mentors, especially Warren Gish and Susan Strome, for their scientific guidance and high standards. Writing a book in the wee hours can be arduous work, so I appreciate Apple Computer for making things simple and WeakLazyLiar and Trespassers William for musical companionship. My coauthors deserve a lot of credit for tolerating my tyranny and helping to make a dream come true. Lastly, I'd like to say a special thanks to Mom and Dad.

Mark

Thanks to my coauthors, Ian and Joey. Special thanks to Stephen Altschul for all his patience with my frequent telephone calls and emails, and to Tom Madden for help with the BLAST code. I'd also like to thank Karen Eilbeck for putting up with me; Suzi Lewis for her patience; and yes, Martin, it is finished now! Finally, I'd like to dedicate my portion of the book to Dr. Marc Perry for showing me my first BLAST report.

Joey

If you are reading this, it means that I'm an O'Reilly author—wow! I'd like to first and foremost thank O'Reilly for putting out a great line of books that have allowed me to make the transition from the bench to the keyboard and, ultimately, to the bookshelves! I also thank my coauthors, Ian and Mark. It is truly amazing that we were able to put this together without even being on the same continent for the last year and a half. This is a testament to Ian's great organizational skills, his grand (yet ever-changing) vision for the book, and his unrelenting quest for perfection. I thank my wife Alison and daughter Lauren for their love and support. Thanks for putting up with the late BLAST nights and early BLAST mornings. I owe you both a lot for your patience and understanding.

Finally, I'd like to thank the members of the Blueberry Hill dart league for their support and friendship.

I'd like to dedicate this book to the memory of David Jagor and to the BBBs, the best group of friends a guy could have!

Part I: Introduction

Chapter 1. Hello BLAST

Welcome to BLAST! This chapter offers a quick start guide to BLAST by exploring some Internet search pages. Throughout the chapter, you may encounter unfamiliar (or even frightening) terms. Don't panic. The terms are fully explained in later chapters or in the Glossary. You don't need to understand all the concepts to get the most out of this chapter. If you're already a seasoned BLAST user, feel free to skip this introduction and dive right into the later sections.

1.1 What Is BLAST?

BLAST is an acronym for Basic Local Alignment Search Tool. Despite the adjective "Basic" in its name, BLAST is a sophisticated software package that has become the single most important piece of software in the field of bioinformatics. There are several reasons for this. First, sequence similarity is a powerful tool for identifying the unknowns in the sequence world. Second, BLAST is fast. The sequence world is big and growing rapidly, so speed is important. Third, BLAST is reliable, from both a rigorous statistical standpoint and a software development point of view. Fourth, BLAST is flexible and can be adapted to many sequence analysis scenarios. Finally, BLAST is entrenched in the bioinformatics culture to the extent that the word "blast" is often used as a verb. There are other BLAST-like algorithms with some useful features, but the historical momentum of BLAST maintains its popularity above all others.

Although BLAST originated at the National Center for Biotechnology Information (NCBI), its development continues at various institutions, both academic and commercial. This can be a little confusing, especially because people often put prefixes or suffixes on the acronym to come up with names like XYZ-BLAST-PDQ. We have aimed to keep this book as simple as possible, and therefore we concentrate on the two most popular versions: NCBI-BLAST and WU-BLAST (pronounced "woo blast"). NCBI-BLAST, as the name suggests, is the version available from the NCBI. WU-BLAST comes from Washington University in St. Louis and is developed by Warren Gish, one of the original authors of BLAST.

1.2 Using NCBI-BLAST

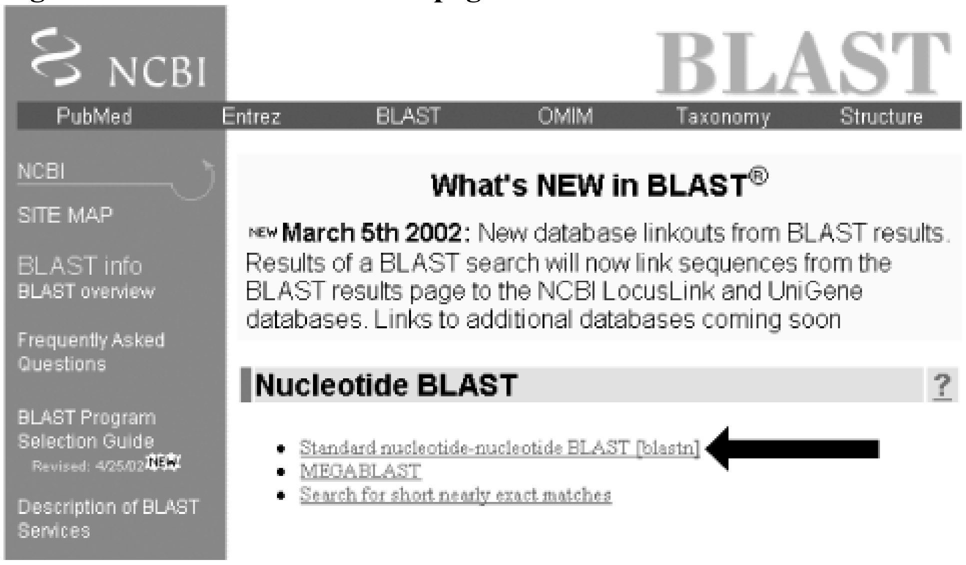
This book begins by exploring the BLAST pages on the NCBI web site. The NCBI, part of the National Institutes of Health, is a U.S. government-funded center for the curation and presentation of public biological knowledge. The NCBI is a public repository for DNA and protein sequences (GenBank), but it's far more than just a data storehouse. The NCBI also maintains a comprehensive medical publication archive (PubMed), distributes many tools for biological analyses (NCBI toolbox), and puts together its own tools for making the most use of the data that it stores (LocusLink, UniGene, RefSeq, Taxonomy browser). Most importantly, for our purposes, it's where the BLAST algorithm was first developed (Altschul et al., 1990) and where it can be obtained, distributed, and used for free without restrictions. Anyone with access to the Internet can run a BLAST search and explore the plethora of genetic resources that have been amassed and curated by the NCBI over the years.

You'll get the most out of this chapter if you follow along with a web browser. Begin by going to the BLAST homepage at http://www.ncbi.nlm.nih.gov/BLAST.

1.2.1 Choosing the BLAST Program

Without explaining all of the options presented on the homepage, let's get right into it with a default BLASTN search. Choose "Standard nucleotide-nucleotide BLAST [blastn]" as shown in Figure 1-1. BLASTN is a program that compares a nucleotide query sequence to a database of nucleotide sequences.

Figure 1-1. NCBI BLAST home page



What's NEW in BLAST®

NEW **March 5th 2002:** New database linkouts from BLAST results. Results of a BLAST search will now link sequences from the BLAST results page to the NCBI LocusLink and UniGene databases. Links to additional databases coming soon

Nucleotide BLAST

- Standard nucleotide-nucleotide BLAST [blastn]
- MEGABLAST
- Search for short nearly exact matches

1.2.2 Entering the Query Sequence

After choosing the kind of search you want to perform, the next step is to define the sequence with which to search. There are three options for this: paste in the bare sequence, paste in a file in FASTA format, or enter a valid NCBI identifier. You can just start typing a sequence in the search box; however, when the search is done, there will be no identifier to describe the sequence you entered. After several such searches, the lack of an identifier will make it difficult to keep track of which results go with which sequence. The second option allows you to define the sequence using the FASTA format. The FASTA format is described in detail in Chapter 11, but the basic specifications are that it's a text file beginning with a greater than sign (>) followed by an identifier and a definition line, which is then proceeded by the one-letter nucleotide or peptide sequence on subsequent lines. Let's use the following sequence:

```
>gi|11611818|gb|AF287139.1|AF287139 Latimeria chalumnae Hoxa-11 gene, partial cds

TACTTGCCAAGTTGCACCTACTACGTTTCGGGTCCCGATTTCTCCAGCCTCCCTTCTTTTTTGCCCCAGACCCCGTCTTCTCG
CCCCATGACATACTCCTATTCGTCTAATCTACCCCAAGTTCAACCTGTGAGAGAAGTTACCTTCAGGGACTATGCCATTGATA
CATCCAATAAATGGCATCCCAGAAGCAATTTACCCCATTGCTACTCAACAGAGGAGATTCTGCACAGGGACTGCCTAGCAACC
ACCACCGCTTCAAGCATAGGAGAAATCTTTGGGAAAGGCAACGCTAACGTCTACCATCCTGGCTCCAGCACCTCTTCTAATTT
CTATAACACAGTGGGTAGAAACGGGGTCCTACCGCAAGCCTTTGACCAGTTTTTCGAGACGGCTTATGGCACAACAGAAAACC
ACTCTTCTGACTACTCTGCAGACAAGAATTCCGACAAAATACCTTCGGCAGCAACTTCAAGGTCGGAGACTTGCAGGGAGACA
GACGAGAAGGAGAGACGGGAAGAAAGCAGTAGCCCAGAGTCTTCTTCCGGCAACAATGAGGAGAAATCAAGCAGTTCCAGTGG
```

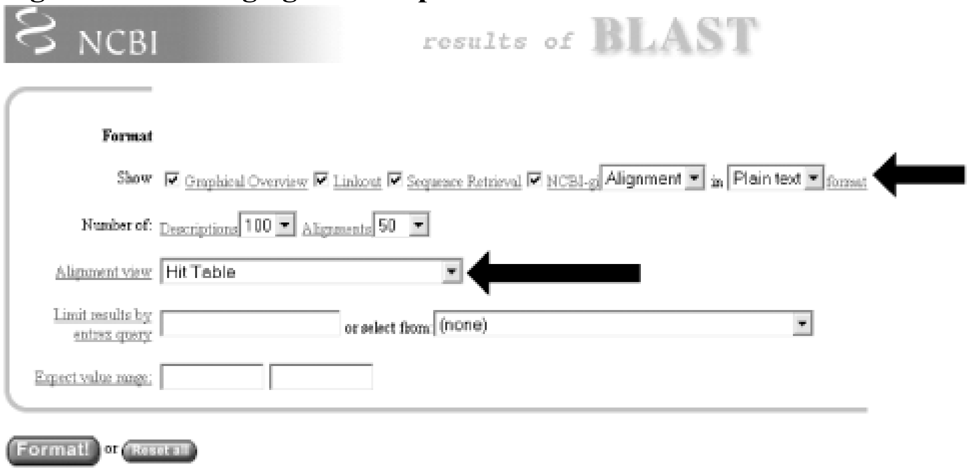
1.3 Alternate Output Formats

This chapter showed the default HTML format, which is obviously best for viewing in a web browser. But what if you wanted to parse the output or store it in a database? HTML is not the best format for these choices. The NCBI also supports Plain Text, eXtensible Markup Language (XML), and ASN.1 formats. To see these different formats, just scroll back to the top of the report, choose another format under the Format option, and then resubmit using the Format! button. You can try this for all the formats, and then just hit the browser Back button to return to the HTML formatted page.

1.4 Alternate Alignment Views

The default Pairwise view shown in Figure 1-10 is the classic BLAST output style, but other options are available for other purposes. These options, described in the NCBI reference section and in Appendix A, include pairwise, query-anchored with identities, query-anchored without identities, flat query-anchored with identities, flat query-anchored without identities, and Hit Table. The most friendly option for text parsers is the Hit Table, which is viewed in plaintext format. This displays all the results in a tab-delimited table, which can be parsed easily. You can select this at the top of the page by changing "Format" to "Plain text" and "Alignment view" to "Hit Table" (Figure 1-12).

Figure 1-12. Changing format options


NCBI
results of BLAST
Format
Show ☑ Graphical Overview ☑ Linkout ☑ Sequence Retrieval ☑ NCBI-gi Alignment in Plain text format
Number of: Descriptions 100 Alignments 50
Alignment view Hit Table
Limit results by entrez query or select from (none)
Expect value range:
Format!
or
Reset all

The Hit Table alignment view is shown in Figure 1-13. The first five lines start with # and are comments about the BLAST program, the query, and the database, followed by a description of the reported fields. The lines after the comments are the alignments in table format. The Hit Table contains all the necessary data to judge a hit without displaying the actual sequence being aligned.

Figure 1-13. Hit Table alignment

```
# BLASTN 2.2.5 [Nov-16-2002]
# Query: gi|11611818|gb|AF287139.1|AF287139 Latimeria chalumnae Hoxa-11 gene, partial cds.
# Database: nr
# Fields: Query id, Subject id, % identity, alignment length, mismatches, gap openings, q. start, q. end, s.
start, s. end, e-value, bit score
# Query: gi|11611818|gb|AF287139.1|AF287139 Latimeria chalumnae Hoxa-11 gene, partial cds.

gi|11611818|gb|AF287139.1|AF287139 gi|11611818|gb|AF287139.1|AF287139 100.00 606 0 0 1 606 1 606 0.0 1201.8
gi|11611818|gb|AF287139.1|AF287139 gi|21699164|gb|AC126321.1| 89.12 239 26 0 1 239 9736 9498 9.0e-69 268.1
gi|11611818|gb|AF287139.1|AF287139 gi|21699164|gb|AC126321.1| 92.00 50 4 0 349 398 9391 9342 1.7e-08 67.89
gi|11611818|gb|AF287139.1|AF287139 gi|21699164|gb|AC126321.1| 100.00 26 0 0 541 566 9199 9174 0.001 52.03
```

The other available alignment options allow a multiple sequence alignment view of the BLAST hits. One of these multiple alignment options, query-anchored with identities, is shown in Figure 1-14. In this view, the full sequence of the query is shown on the top line with a unique identifier (1_18852, in this case). Subsequently, each line shows the alignment for one database hit. Identical residues are represented with a dot (.), while nucleotide differences are shown explicitly. This alignment option is useful for quickly identifying changes common to a group of sequences. For example, you can see from the part of the alignment shown in Figure 1-14 that the bottom four sequences (6754225, 664837, 664835, and 664831) have common shared differences. A deeper look into these sequences reveals that they are actually different database entries for the same mouse Hoxa11 gene, which is homologous to the coelacanth Hoxa11 gene.

Figure 1-14. Query-anchored with identities view

```
ALIGNMENTS
1_18852  1      tacttgccaagttgcacctactacgtttcgggtcccgatttctccagcctcccttctttt 60
11611818 1      ............................................................ 60
21699164 9736   .....a..c.....t..t........g....c...t........................ 9677
211938   79        ...........t..g........c........t........................ 135
13249168 8516      ...........t..g........c........t........................ 8572
11611820 7          ..............t........c........t..............t.....a..c 63
20278974 30055                              ........g..............g......... 30087
6754225  94        ...........t..t........c........a........................ 150
664837   4753      ...........t..t........c........a........................ 4809
664835   93        ...........t..t........c........a........................ 149
664831   432       ...........t..t........c........a........................ 376

1_18852  61     ttgccccagaccccgtcttctcgccccatgacatactcctattcgtctaatctaccccaa 120
11611818 61     ............................................................ 120
```

1.5 The Next Step

This chapter has taken you through a simple BLASTN search at the NCBI database; however, more than two dozen specialized BLAST pages are available, and they let you do anything—from screening for vector sequence, to identifying protein family members, to mapping a sequence to the human genome. For a quick guide to these specialized pages, the NCBI presents a convenient reference to these tools at http://www.ncbi.nlm.nih.gov/BLAST/producttable.html.

1.6 Further Reading

Altschul, S.F., T.L. Madden, A.A. Schaeffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman (1997). "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs." Nucleic Acids Research, 25, pp. 3389-3402.

Part II: Theory

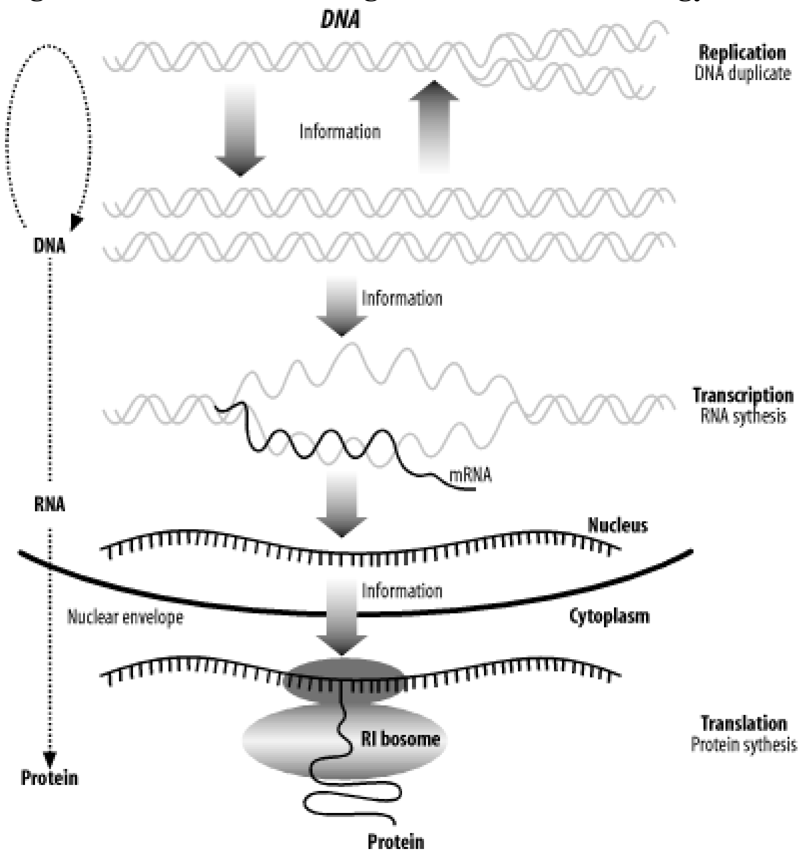
Chapter 2. Biological Sequences

Sequence similarity is a powerful tool for discovering biological function. Just as the ancient Greeks used comparative anatomy to understand the human body and linguists used the Rosetta stone to decipher Egyptian hieroglyphs, today we can use comparative sequence analysis to understand genomes, RNAs, and proteins. But why are biological sequences similar to one another in the first place? The answer to this question isn't simple and requires an understanding of molecular and evolutionary biology.

2.1 The Central Dogma of Molecular Biology

Most courses in molecular biology begin with the Central Dogma of Molecular Biology, which describes the path by which information contained in DNA is converted to protein molecules with specific functions. Stated simply, the Central Dogma is: "from DNA to RNA to protein." Figure 2-1 shows a more complete diagram of this process and will be referenced throughout this section.

Figure 2-1. The Central Dogma of Molecular Biology: DNA to RNA to protein



2.1.1 DNA

The hereditary material that carries the blueprint for an organism from one generation to the next is called deoxyribonucleic acid. It is much more commonly referred to by its acronym, DNA. Every time cells divide, the DNA is duplicated in a process called DNA replication. The entire DNA of an organism is called its genome, and genomes are sometimes called "the book of life" (especially with respect to the human genome). Reading and understanding the various books of life is one of the most important quests of the genomic age. Modern medicine, agriculture, and industry will increasingly depend on an intimate knowledge of genomes to develop individualized medicines, select and modify the most desirable traits in plants and animals, and understand the relationships among species.

The language of DNA is complicated. Over the last 50 years, scientists have begun to decipher it, but it is still largely a mystery. Although the language is elusive, the alphabet is simple, consisting of just four nucleotides: adenine, cytosine, guanine, and thymine. For simplicity in both speech and on the computer, they are usually abbreviated as A, C, G, and T. DNA usually exists as a double-stranded molecule, but we generally talk about just one strand at a time. Here's an example of a DNA sequence that is six nucleotides (nt) long:

```
GAATTC
```

DNA has polarity, like a battery, but its ends are referred to as 5-prime (5´) and 3-prime (3´) rather than plus and minus. This nomenclature comes from the chemical structure of DNA. While it isn't necessary to understand the chemical structure, the terminology is important. For example, when people say "the 5´ end of the gene," they mean the beginning of the gene. We usually display DNA sequence as we read text, left to right, and the convention is that the left side is the 5´ end and the right side is the 3´ end.

In addition to the 4-letter alphabet, there is also a 15-letter DNA alphabet used to describe nucleotide ambiguities (

2.2 Evolution

BLAST works because evolution is happening. Biological sequences show complex patterns of similarity to one another. In this regard, they mirror the external morphologies of the organisms in which they reside. You'll notice that birds, for example, show natural groupings. You don't have to be a biologist to see that ducks, geese, and swans comprise a reasonably natural group called the waterfowl, and that the similarities between ducks and geese seem too great to explain by mere coincidence. Biological sequences are no different. After all, the reason why ducks look like ducks and geese look like geese is because of their genes. Many molecular biologists are convinced that understanding sequence evolution is tantamount to understanding evolution itself.

Sequences change over time due to three forces: mutation, natural selection, and genetic drift. If you use BLAST, it's important to understand these forces because they form the biological foundation of similarity searches. The biological and mathematical foundations aren't the same, and are sometimes at odds. You need to understand both theories in order to knowledgeably interpret the sequence alignments in a BLAST report.

2.2.1 Mutation

A mutation is simply a change in a DNA sequence. What causes mutation? Many chemicals and conditions damage DNA, so its sequence either changes or ceases to be recognizable. Mutagenic agents are often called carcinogens because cancer is caused by the accumulation of mutations in genes that control cell division. But even in a world without carcinogens there would still be mutation because the process of DNA replication isn't perfect. Every time a cell divides, it must duplicate its DNA. The human genome is about three billion letters long, and the error rate of DNA replication is about one error in every 300 million letters, so you can expect about 10 mutations per genome duplication. Genome size varies, as does the replication error rate, so don't take the 10 mutations per genome replication as any kind of biological truth. Human beings are composed of about a trillion cells, and you might take a moment now and consider just how much mutation is going on in your own body. Whatever that large number is, it's infinitesimal compared to what's happening in the biosphere as a whole.

What happens when a mutation occurs in the protein-coding portion of a gene? Because the DNA is mutated, the mRNA is also mutated. This in turn may lead to a different protein, but not necessarily, because the genetic code is degenerate. Take a look at an example for which you mutate just one letter in a coding sequence. If the mutation changed a codon from TTA to TTG, for example, the protein would be unchanged because both codons translate to the amino acid leucine. Such mutations are called silent, synonymous, or same-sense because they don't affect the protein sequence in any way. If the mutation changed a TTA to a TTT, however, the codon would code for a different amino acid, phenylalanine. Such substitutions are called mis-sense mutations. Molecular biologists will often classify mis-sense mutations into either conservative or nonconservative substitutions, depending on whether the two amino acids are chemically similar to one another. Leucine and phenylalanine are both hydrophobic amino acids, and such a substitution would be considered conservative. Bioinformaticists, however, give a more rigorous and quantifiable definition of conservative (see Chapter 4). If the TTA codon is mutated to TAA, the codon becomes a stop codon, which causes the ribosome to stop translating the mRNA. This represents the most destructive kind of mutation, and is called a non-sense mutation. Non-sense mutations cause translation to terminate prematurely, and the result is a truncated protein that may function partially, not function at all, or be poisonous to the cell.

Not all mutations substitute one nucleotide for another. Some mutations may insert or remove nucleotides. In addition, there are duplications, inversions, and other large-scale rearrangements that destroy genes or even fuse them together. Insertions and deletions are often destructive because they change the reading frame of translation if they aren't additions/subtractions of a multiple of three (a whole codon). After such a frame-shift mutation, there are usually several mis-sense mutations caused by the out-of-frame codons, and then a premature stop codon that was not previously in frame. Insertions and deletions are therefore usually as disruptive as mis-sense mutations.

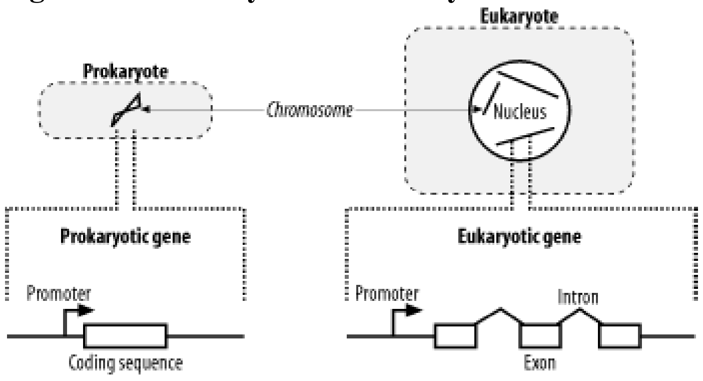
What happens to an organism with mutations? It depends on a lot of factors. A mutation may have disastrous consequences, it might prove beneficial, or it might have no effect at all. To understand the forces that govern sequence evolution, let's take a close look at natural selection and genetic drift.

2.2.2 Natural Selection

2.3 Genomes and Genes

In general, the genomic structure of prokaryotes is very different from that of eukaryotes (Figure 2-5). Genomes are organized into chromosomes. Prokaryotes often have a single circular chromosome, and eukaryotes usually have multiple linear chromosomes. People are sometimes surprised to find that genome size and chromosome number aren't reflected in organismal complexity. For example, the single-celled *Amoeba dubia* has a genome that is about 200 times larger than the human genome. Although dogs and cats have very similar genome sizes, dogs have twice as many chromosomes. One rule to keep in mind when thinking about genomic organization is that genomes of viruses and prokaryotic organisms generally contain little noncoding sequence, whereas the genomes of more complex organisms usually contain a much higher percentage of noncoding sequence.

Figure 2-5. Prokaryote and eukaryote cells



2.3.1 Prokaryotic Genes

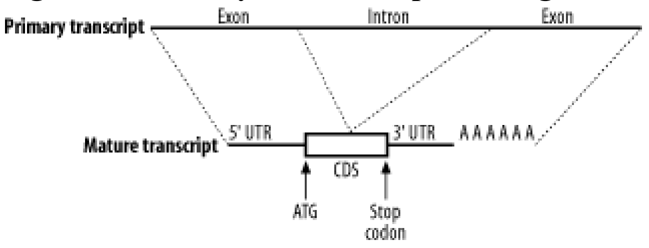
Prokaryotic genes are relatively simple. They contain a promoter that determines when the gene is transcribed and a coding region that contains the DNA sequence for a protein. It is relatively easy to find genes in prokaryotic genomes. Since stop codons are expected about every 21 triplets (there are three stop codons out of 64 total triplet combinations), long open reading frames (ORFs) should be very rare, at least from an unbiased random model. On average, proteins are 300 amino acids long, so finding an ORF that is 900 nucleotides long is really unexpected and a pretty clear signal that the ORF codes for a real protein. Of course, some genes encode small proteins, and finding these is a bit more difficult.

2.3.2 Eukaryotic Genes

Eukaryotic gene structure is more complicated than prokaryotic gene structure. Unlike prokaryotic genes, eukaryotic genes are often broken into pieces that are assembled before they are translated. Like prokaryotes, eukaryotes also have promoters to regulate when genes are turned on, but they are often much larger and may exist a great distance from the start of translation. In addition, many genes respond to additional sequences called enhancers and suppressors that aren't necessarily upstream of a gene and may be many kilobases away.

In eukaryotes, mRNAs are processed before they are translated (Figure 2-6). Two kinds of processing are common: splicing and poly-adenylation. Splicing brings together the coding sequences and throws out the intervening stuff. The sequences that end up in the mature mRNA are called exons, and the intervening stuff is called introns. The part of the mRNA that codes for protein is called the coding sequence (CDS), and the parts at either end are called untranslated regions (UTRs). The other common post-transcriptional modification is poly-adenylation. In this process, 50 or more adenine nucleotides are added to the end of the mRNA, which is called a poly-A tail.

Figure 2-6. Eukaryotic mRNA processing



2.4 Biological Sequences and Similarity

The beginning of this chapter asked why biological sequences are similar to one another. Let's answer that question now. You've seen that biological sequences like proteins may have important functions necessary for the survival of an organism. You've also seen that DNA sequence can mutate randomly, and this may change how a sequence functions. Over time, both functional constraints and random processes impact the course of sequence evolution. The degree to which a sequence follows a functional or random path depends on natural selection and neutral evolution. So the reason why sequences are similar to one another is because they start out similar to one another and follow different paths. When you read a BLAST report, you will find that your knowledge of molecular and evolutionary biology will help you interpret the similarities and differences you see.

2.5 Further Reading

Genetics, molecular biology, and evolution aren't especially difficult topics, but they are filled with many potentially unfamiliar terms. The following books are recommended for those just getting started in these fields. They are informative and entertaining, and can help more experienced readers communicate effectively with novices.

Clark, David P. and Lonnie D. Russell, Molecular Biology Made Simple and Fun (Cache River Press). Gonick, Larry and Mark Wheelis, The Cartoon Guide to Genetics (Perennial). Tagliaferro, Linda and Mark Vincent Bloom, The Complete Idiot's Guide to Decoding Your Genes (Alpha Books).

The following are typical textbooks for college-level courses in molecular biology, genetics, and evolution:

Alberts, Brooks et al., Molecular Biology of the Cell (Garland). Futuyma, Douglas J., Evolutionary Biology (Sinauer Associates, Inc.). Graur, Dan and Wen-Hsiung Li, Molecular Evolution (Sinauer). Hartl, Daniel L. and Elizabeth W. Jones, Genetics: Analysis of Genes and Genomes, (Jones & Bartlett). Lewin, Benjamin, Genes VII (Oxford University Press). Lodish, Harvey et. al., Molecular Cell Biology (W.H. Freeman & Co.). Page, Roderic D. M. and Edward C. Holmes, Molecular Evolution: A Phylogenetic Approach (Blackwell Science). Watson, James D. and Joan Steitz. Molecular Biology of the Gene (Addison-Wesley).

Chapter 3. Sequence Alignment

BLAST finds statistically significant similarities between sequences by evaluating alignments, but how are sequences aligned? In principle, there are many ways to align two sequences, but in practice, one method is used more often than any other. This chapter explains this technique with the biologist in mind, without using the mathematical notation and jargon that is usually employed to describe such algorithm. Divested of unfamiliar language and notation, these algorithms are quite simple.

Finding the optimal alignment between two sequences can be a computationally complex task. Fortunately, a technique called *dynamic programming* (DP) makes sequence alignment tractable as long as you follow a few rules. Rather than have you struggle with a confusing definition of DP, this chapter demonstrates how the technique works for sequence alignment and then gets back to the generalities. There are fundamentally two kinds of alignment: global and local. In global alignment, both sequences are aligned along their entire lengths and the best alignment is found. In local alignment, the best subsequence alignment is found. For example, if you want to find the two most similar sentences between two books, you use local alignment. If you want to compare the sentences end to end, use global alignment. This chapter describes global alignment, then local alignment. The example uses English words instead of biological sequences and the algorithms are quite general.

3.1 Global Alignment: Needleman-Wunsch

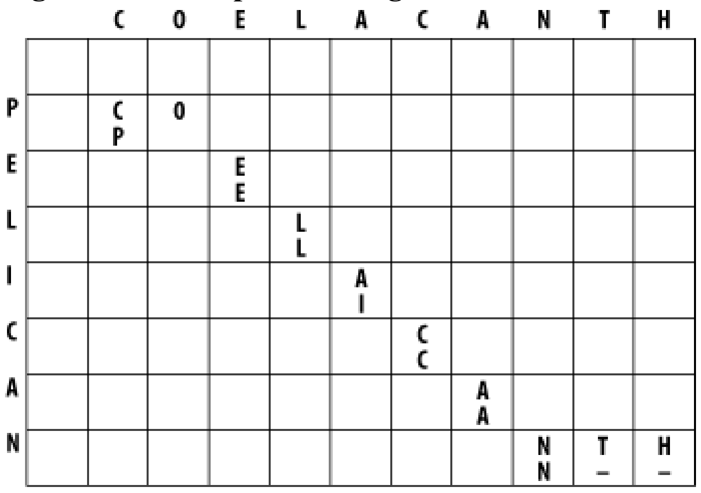
The global alignment algorithm described here is called the Needleman-Wunsch algorithm. We will explain it in a way that seems natural to biologists, that is, it tells the end of the story first, and then fills in the details. (This is why biologists make terrible comedians; they always tell the punch line first.) We will align the words COELANCANTH and PELICAN using a simple scoring scheme: +1 for letters that match, -1 for mismatches, and -1 for gaps. The alignment will eventually look like one of the following, which are equivalent given our scoring scheme:

```
COELACANTH      COELACANTH
P-ELICAN--      -PELICAN--
```

Note that every letter of each sequence is aligned to a letter or a gap. In local alignments, discussed later, this isn't the case.

The alignment takes place in a two-dimensional matrix in which each cell corresponds to a pairing of one letter from each sequence. To get an intuitive understanding of the alignment algorithm, look at Figure 3-1, which shows where the maximum scoring alignment lies in the matrix. The alignment starts at the upper left and follows a mostly diagonal path down and to the right. When two letters are aligned, the path follows a diagonal trajectory. There are several places in which the letters from COELACANTH are paired to gap characters. In this case, the graph is followed horizontally. Although not shown here, the path may be also be followed vertically when the letters from PELICAN are paired with gap characters. Gap characters can never be paired to each other. Note that the first row and column are blank. The reason for this will become clear shortly.

Figure 3-1. Example of an alignment matrix



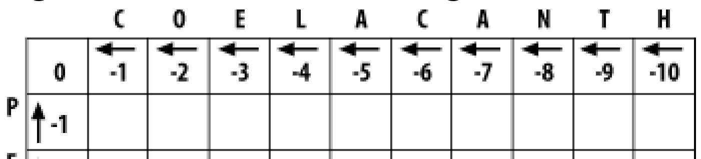
		C	O	E	L	A	C	A	N	T	H
P		C P	O								
E				E E							
L					L L						
I						A I					
C							C C				
A								A A			
N									N N	T -	H -

In reality, you don't store letters in the matrix as shown in Figure 3-1. Each cell of the matrix actually contains two values: a score and a pointer. The score is derived from the scoring scheme. Here, this means +1 or -1, but when aligning biological sequences, the values come from a scoring matrix (a topic of the next chapter). The pointer is a directional indicator (an arrow) that points up, left, or diagonally up and left. The pointer navigates the matrix, and its use will become clearer later in the chapter. Now, let's look at the algorithm in detail. There are three major phases: initialization, fill, and trace-back.

3.1.1 Initialization

In the initialization phase, you assign values for the first row and column (Figure 3-2). The next stage of the algorithm depends on this. The score of each cell is set to the gap score multiplied by the distance from the origin. Gaps may be present at the beginning of either sequence, and their cost is the same as anywhere else. The arrows all point back to the origin, which ensures that alignments go all the way back to the origin (a requirement for global alignment).

Figure 3-2. Initialization of the alignment matrix



		C	O	E	L	A	C	A	N	T	H
	0	← -1	← -2	← -3	← -4	← -5	← -6	← -7	← -8	← -9	← -10
P	↑ -1										
E											

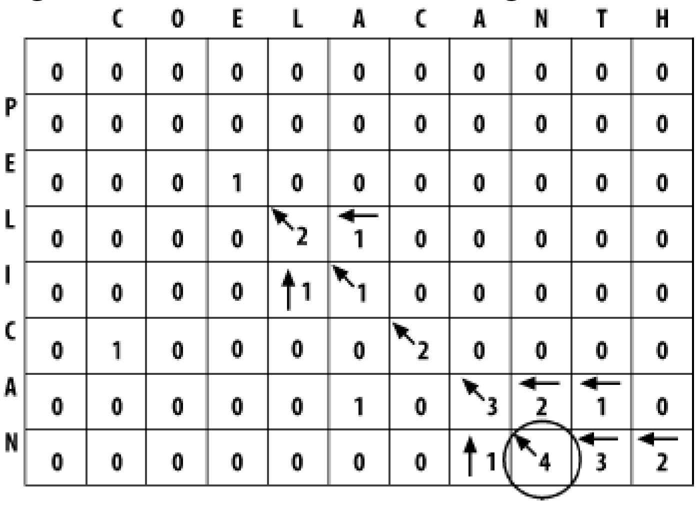
3.2 Local Alignment: Smith-Waterman

The local alignment algorithm we describe here, the Smith-Waterman algorithm, is a very simple modification of Needleman-Wunsch. There are only three changes:

- The edges of the matrix are initialized to 0 instead of increasing gap penalties.
- The maximum score is never less than 0, and no pointer is recorded unless the score is greater than 0.
- The trace-back starts from the highest score in the matrix (rather than at the end of the matrix) and ends at a score of 0 (rather than the start of the matrix).

These simple changes have a profound effect on the behavior of algorithm. Using the same words and scoring scheme as you did in global alignment, look at the filled matrix in Figure 3-6.

Figure 3-6. Filled Smith-Waterman alignment matrix



		C	O	E	L	A	C	A	N	T	H
	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	1	0	0	0	0	0	0	0
L	0	0	0	0	↖2	←1	0	0	0	0	0
I	0	0	0	0	↑1	↖1	0	0	0	0	0
C	0	1	0	0	0	0	↖2	0	0	0	0
A	0	0	0	0	0	1	0	↖3	←2	←1	0
N	0	0	0	0	0	0	0	↑1	↖4	←3	←2

As you can see, there are a lot of zeroes. That's because there are a lot of places where you can't get a positive score. Also note that one cell is circled. This is the cell with the maximum score in the matrix. There may be more than one place in the matrix with the same maximum score, and in this case, some kind of arbitrary decision must be made. Start your trace-back from the maximum score and follow it to a score of zero, creating your alignment as you step backward, just as you did with global alignment. At the end, you have an alignment with a score of 4 that looks like this:

```
ELACAN
ELICAN
```

Example 3-2 shows the Perl code.

Example 3-2. Local alignment with the Smith-Waterman algorithm

```
# Smith-Waterman  Algorithm

# usage statement
die "usage: $0 <sequence 1> <sequence 2>\n" unless @ARGV == 2;

# get sequences from command line
my ($seq1, $seq2) = @ARGV;

# scoring scheme
my $MATCH    =  1; # +1 for letters that match
my $MISMATCH = -1; # -1 for letters that mismatch
my $GAP      = -1; # -1 for any gap
```

3.3 Dynamic Programming

Now that you've seen the typical approach to global and local alignment, consider the generality of dynamic programming. The advantage of DP can be seen in the fill phase. Each cell represents the maximum scoring alignment between the two sequences up to that point. When you calculate the next cell, you use previously stored values. In other words, DP is an optimizing function whose definition is extended as the algorithm proceeds.

3.4 Algorithmic Complexity

The complexity of algorithms is often described in big-O notation, a shorthand for the asymptotic behavior of the algorithm. For example, searching for a name in a phone book by starting at the beginning and going name-by-name takes on average, *n/2* operations, where *n* is the number of names in the phone book. Such a search has *O(n)* time complexity (constants are dropped from the notation). It scales linearly in time; a phone book twice as long takes twice as long to search. An approach that scales more efficiently successively splits the phone book in half based on the alphabetical order. This is called a *binary search* and has O(log2n) complexity. For example, a phone book eight times longer takes only three times longer to search. The alignment algorithms as described have *O(nm)* complexity in both time and memory, where *n* and *m* are the lengths of the sequences. It's also common to label such algorithms as *O(n2)* and speak of them as scaling quadratically.

3.5 Global Versus Local

Although global and local alignment are mechanistically similar, they have very different properties. Consider the alignment between the genomic sequence of two eukaryotic genes from distantly related organisms. You'd expect the exons to remain the same because their coding sequences are evolutionarily constrained, but the introns may no longer be recognizably similar, especially if they have acquired many insertions or deletions. The problem is that exons may account for only 1 to 2 percent of the sequence. As a result, a global alignment between these sequences is an alignment of mostly random letters. In such a scenario, it's very likely (especially if introns change size, as they often do) that the exons will not align to one another because their score contribution is very small compared to the rest of the sequence. In contrast, local alignment can pick out conserved exons, but unfortunately, just the maximum scoring one. The shortcomings of the standard alignment algorithms have been addressed by numerous variants.

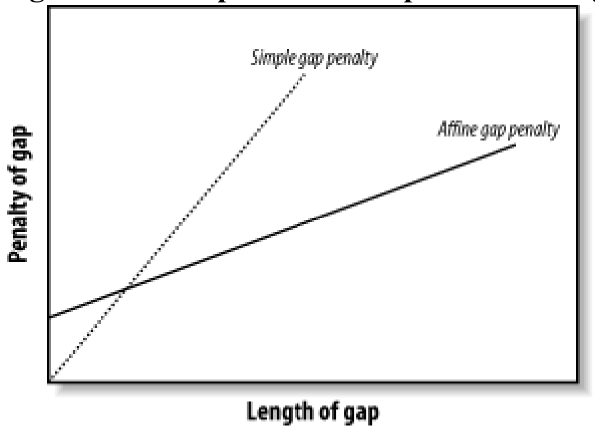
3.6 Variations

The algorithms as described and implemented earlier are rarely used. Over the years, important innovations have made the general algorithms more applicable to aligning biological sequences and running efficiently in a computer.

3.6.1 Gap Modifications

Most alignment algorithms employ affine gaps. The previous description used a simple gap-scoring scheme in which every letter inserted or deleted cost the same amount. This isn't a very good model of biology because large gaps tend to occur fairly often. To better model this phenomenon, affine gaps are used where there is a greater penalty for opening a gap than extending the gap. Figure 3-7 shows a graphical view.

Figure 3-7. Comparison of simple and affine gap scoring schemes



Affine gap penalties are a simple modification to either algorithm. All you need to do is to record a third value in each cell of the matrix that keeps track of whether a gap has already been opened or not and then assign the appropriate gap penalty. Some algorithms even allow multiple gap scoring schemes so that very long gaps are not penalized as much. You can visualize how this works by following the minimal penalty in Figure 3-7. These algorithms are slightly more complicated because scores for each affine gap must be tracked. Usually two affine penalties are employed, and the algorithms are labeled double affine.

3.6.2 Reduced Memory

You shouldn't attempt to align long sequences (e.g., genomic DNA) with the versions of Needleman-Wunsch and Smith-Waterman described in Examples Example 3-1 and Example 3-2. The number of cells in the alignment matrix is the product of the sequence lengths, and each cell may take 8 bytes or more of memory. Therefore, aligning two 100,000-bp DNA sequences requires approximately 80 gigabytes (GB) of RAM. Fortunately, there are linear-memory versions of the algorithms.

You may have noticed during the fill phase of the algorithms that you use only two rows at a time. The other rows just sit there, either waiting to be calculated or waiting for the trace-back. Why not just use two rows at a time and not allocate the whole matrix? If you do this, you can calculate the score of the maximum scoring alignment and record its ending coordinates. By reversing direction, you can then compute the starting coordinates. Unfortunately, you lose the ability to perform a trace-back. But the memory required is now just two times the length of one of the sequences rather than the product of the two sequences. Using this approach, you can compare two 100,000-bp DNA sequences in just 1.6 megabytes (MB) of RAM. The alignment algorithm is now *O(n)* in memory, but still *O(nm)* in time.

How do you get the actual alignment? Given the coordinates of the maximum scoring alignment you can restrict the search space to a region between the two endpoints (Figure 3-8). With this approach, a diagonal line is drawn between the endpoints, and the search space is restricted to a certain number of cells on either side of the diagonal. The width of the search space is called the bandwidth. If the bandwidth is too small, the alignment will walk along the edge of the band and may no longer follow the path of the maximum score. If the bandwidth is needlessly large, both memory and computation are wasted. Now that the search space is sufficiently small, you can perform a complete fill

3.7 Final Thoughts

When you use the Needleman-Wunsch or Smith-Waterman algorithms to find the maximum scoring alignment, you're playing by computational, not biological, rules. As a result, the maximum scoring alignment only approximates the truth. However, even if all the nuances of biology were clear and you could code this in a computer algorithm, you might still favor the approximation because the computational cost of the correct algorithm can be excessive. In any case, the fact that you can align the unrelated words pelican and coelacanth merits consideration. It's possible to align any sequence; finding proper meaning in alignments is up to the user, not the algorithm.

3.8 Further Reading

For more information on the Perl programming language, consider these books:
Christiansen, Tom and Nathan Torkington, Perl Cookbook (O'Reilly & Associates). Schwartz, Randal L. and Tom Phoenix, Learning Perl (O'Reilly). Tisdall, James, Beginning Perl for Bioinformatics (O'Reilly). Wall, Larry, Tom Christiansen, and Jon Orwant, Programming Perl (O'Reilly).

The following texts are indispensable resources for information on sequence alignment and algorithms in general:
Cormen, Thomas H. et al., Introduction to Algorithms (MIT Press). Gusfield, Dan, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology (Cambridge University Press).

Chapter 4. Sequence Similarity

The fact that the human genome is often referred to as the Book of Life is an apt description because nucleic acids and proteins are often represented (and manipulated) as text files. Chapter 3 described common algorithms for aligning sequences of letters, and score is the metric used to determine the best alignment. This chapter shows what scores really are. Some of the introduced terms come from information theory, so the chapter begins with a brief introduction to this branch of mathematics. It then explores the typical ways to measure sequence similarity. You'll see that this approach fits well with the sequence-alignment algorithms described in Chapter 3. The last part of the chapter focuses on the statistical significance of sequence similarity in a database search. The theories discussed in this chapter apply only to local alignment. There is currently no theory for global alignment.

4.1 Introduction to Information Theory

In common usage, the word information conveys many things. Forget everything you know about this word because you're going to learn the most precise definition. Information is a decrease in uncertainty. You can also think of information as a degree of surprise.

Suppose you're taking care of a child and the response to every question you ask is "no." The child is very predictable, and you are pretty certain of the answer the next time you ask a question. There's no surprise, no information, and no communication. If another child answers "yes" or "no" to some questions, you can communicate a little, but you can communicate more if her vocabulary was greater. If you ask "do you like ice cream," which most children do, you would be informed by either answer, but more surprised if the answer was "no." Qualitatively, you expect more information to be conveyed by a greater vocabulary and from surprising answers. Thus, the information or surprise of an answer is inversely proportional to its probability. Quantitatively, information is represented by either one of the following equivalent formulations shown in Figure 4-1.

Figure 4-1. Equation 4-1

$$H(p) = \log_2 \frac{1}{p} \qquad H(p) = -\log_2 p$$

The information, *H*, associated with some probability *p*, is by convention the base 2 logarithm of the inverse of *p*. Values converted to base 2 logarithms are given the unit bits, which is a contraction of the words binary and digit (it is also common to use base e, and the corresponding unit is nats). For example, if the probability that a child doesn't like ice cream is 0.25, this answer has 2 bits of information (liking ice cream has 0.41 bits).

It is typical to describe information as a message of symbols emitted from a source. For example, tossing a coin is a source of head and tail symbols, and a message of such symbols might be:

```
tththttt
```

Similarly, the numbers 1, 2, 3, 4, 5, and 6 are symbols emitted from a six-sided die source, and the letters A, C, G, and T are emitted from a DNA source. The symbols emitted by a source have a frequency distribution. If there are *n* symbols and the frequency distribution is flat, as it is for a fair coin or die, the probability of any particular symbol is simply 1/*n*. It follows that the information of any symbol is log2(n), and this value is also the average. The formal name for the average information per symbol is entropy.

But what if all symbols aren't equally probable? To compute the entropy, you need to weigh the information of each symbol by its probability of occurring. This formulation, known as Shannon's Entropy (named after Claude Shannon), is shown in Figure 4-2.

Figure 4-2. Equation 4-2

$$H = -\sum_{i}^{n} p_i \log_2 p_i$$

Entropy (*H*) is the negative sum over all the symbols (n) of the probability of a symbol (pi) multiplied by the log base 2 of the probability of a symbol (log2pi). Let's work through a couple of examples to make this clear. Start with the flip of a coin and assume that h and t each have a probability 0.5 and therefore a log2 probability of -1. The entropy of a coin is therefore:

```
 - ( (0.5)(-1) + (0.5)(-1) ) = 1 bit
```

Suppose you have a trick coin that comes up heads 3/4 of the time. Since you're a little more certain of the outcome, you expect the entropy to decrease. The entropy of your trick coin is:

```
 - ( (0.75)(-0.415) + (0.25)(-2) ) = 0.81 bits
```

A random DNA source has an entropy of:

```
- ( (0.25)(-2) + (0.25)(-2) + (0.25)(-2) + (0.25)(-2) ) = 2 bits
```

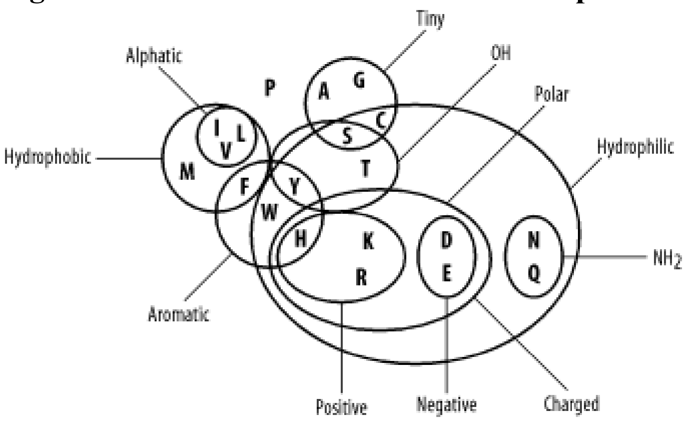
However, a DNA source that emits 90 percent A or T and 10 percent G or C has an entropy of:

```
- ( 2(0.45)(-1.15) + 2(0.05)(-4.32) ) = 1.47 bits
```

4.2 Amino Acid Similarity

Molecular biologists usually think of amino acid similarity in terms of chemical similarity (see Table 2-1). Figure 4-3 depicts a rough qualitative categorization. From an evolutionary standpoint, you expect mutations that radically change chemical properties to be rare because they may end up destroying the protein's three-dimensional structure. Conversely, changes between similar amino acids should happen relatively frequently.

Figure 4-3. Amino acid chemical relationships



In the late '60s and early '70s, Margaret Dayhoff pioneered quantitative techniques for measuring amino acid similarity. Using sequences that were available at the time, she constructed multiple alignments of related proteins and compared the frequencies of amino acid substitutions. As expected, there is quite a bit of variation in amino acid substitution frequency, and the patterns are generally what you'd expect from the chemical properties. For example, phenylalanine (F) is most frequently paired to itself. It is also found relatively frequently with tyrosine (Y) and tryptophan (W), which share similar aromatic ring structures (see Table 2-1), and to a lesser extent with the other hydrophobic amino acids (M, V, I, and L). Phenylalanine is infrequently paired with hydrophilic amino acids (R, K, D, E, and others). You can see some of these patterns in the following multiple alignment, which corresponds to a portion of the *cytochrome b* protein from various organisms.

```
 PGNPFATPLEILPEWYLYPVFQILRVLPNKLLGIACQGAIPLGLMMVPFIE
PANPFATPLEILPEWYFYPVFQILRTVPNKLLGVLAMAAVPVGLLTVPFIE
PANPMSTPAHIVPEWYFLPVYAILRSIPNKLGGVAAIGLVFVSLLALPFIN
PANPLVTPPHIKPEWYFLFAYAILRSIPNKLGGVLALLFSILMLLLVPFLH
PANPLSTPAHIKPEWYFLFAYAILRSIPNKLGGVLALLLSILVLIFIPMLQ
PANPLSTPPHIKPEWYFLFAYAILRSIPNKLGGVLALLLSILILIFIPMLQ
IANPMNTPTHIKPEWYFLFAYSILRAIPNKLGGVIGLVMSILIL..YIMIF
ESDPMMSPVHIVPEWYFLFAYAILRAIPNKVLGVVSLFASILVL..VVFVL
IVDTLKTSDKILPEWFFLYLFGFLKAIPDKFMGLFLMVILLFSL..FLFIL
```

Dayhoff represented the similarity between amino acids as a log2 odds ratio, also known as a *lod score*. To derive the lod score of an amino acid, take the log2 of the ratio of a pairing's observed frequency divided by the pairing's random expected frequency. If the observed and expected frequencies are equal, the lod score is zero. A positive score indicates that a pair of letters is common, while a negative score indicates an unlikely pairing. The general formula for any pair of amino acids is shown in Figure 4-4.

Figure 4-4. Equation 4-3

$$S_{ij} = \log(q_{ij}/p_i p_j)$$

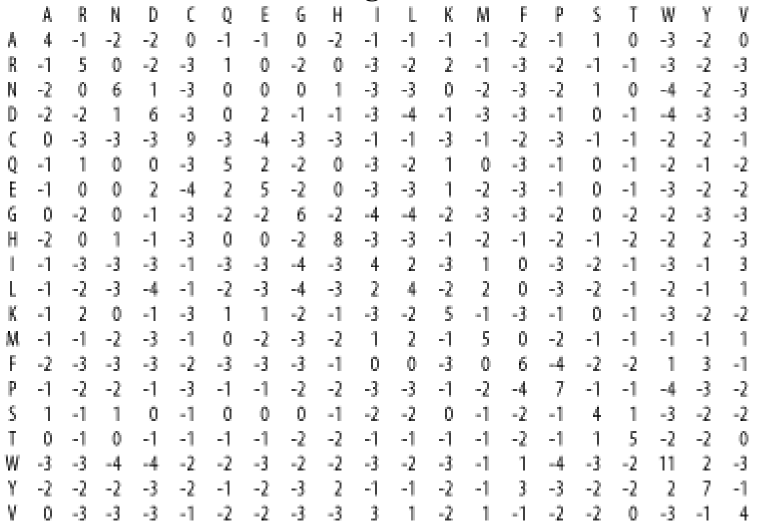
The score of two amino acids *i* and *j*, is *s*ij, their individual probabilities are *p*i and *p*j, and their frequency of pairing is *q*ij. For example, suppose the frequencies of methionine (M) and leucine (L) in your data set are 0.01 and 0.1, respectively. By random pairing, you expect 1/1000 amino acid pairs to be M-L. If the observed frequency of pairing is 1/500, the odds ratio is 2/1. Converting this to a base 2 logarithm gives a lod score of +1, or 1 bit. Similarly, if the frequency of arginine (R) is 0.1 and its frequency of pairing with L is 1/500, the lod score of an R-L pair is -2.322 bits. In computers, using base e rather than base 2 is more convenient. The values of +1 and -2.322 bits are 0.6931 and -1.609 nats, respectively.

If you know the direction of change from an evolutionary tree, the pair-wise scores can be asymmetric. That is, the

4.3 Scoring Matrices

A two-dimensional matrix containing all possible pair-wise amino acid scores is called a *scoring matrix*. Scoring matrices are also called substitution matrices because the scores represent relative rates of evolutionary substitutions. Scoring matrices are evolution in a nutshell. Take a moment now to peruse the scoring matrix in Figure 4-5 and compare it to the chemical groupings in Figure 4-3.

Figure 4-5. BLOSUM62 scoring matrix



	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Lod scores are real numbers but are usually represented as integers in text files and computer programs. To retain precision, the scores are generally multiplied by some scaling factor before converting them to integers. For example, a lod score of -1.609 nats may be scaled by a factor of two and then rounded off to an integer value of -3. Scores that have been scaled and converted to integers have a unitless quantity and are called raw scores.

4.3.1 PAM and BLOSUM Matrices

Two different kinds of amino acid scoring matrices, PAM (Percent Accepted Mutation) and BLOSUM (BLOcks SUbstitution Matrix), are in wide use. The PAM matrices were created by Margaret Dayhoff and coworkers and are thus sometimes referred to as the Dayhoff matrices. These scoring matrices have a strong theoretical component and make a few evolutionary assumptions. The BLOSUM matrices, on the other hand, are more empirical and derive from a larger data set. Most researchers today prefer to use BLOSUM matrices because in silico experiments indicate that searches employing BLOSUM matrices have higher sensitivity.

There are several PAM matrices, each one with a numeric suffix. The PAM1 matrix was constructed with a set of proteins that were all 85 percent or more identical to one another. The other matrices in the PAM set were then constructed by multiplying the PAM1 matrix by itself: 100 times for the PAM100; 160 times for the PAM160; and so on, in an attempt to model the course of sequence evolution. Though highly theoretical (and somewhat suspect), it is certainly a reasonable approach. There was little protein sequence data in the 1970s when these matrices were created, so this approach was a good way to extrapolate to larger distances.

Protein databases contained many more sequences by the 1990s so a more empirical approach was possible. The BLOSUM matrices were constructed by extracting ungapped segments, or *blocks,* from a set of multiply aligned protein families, and then further clustering these blocks on the basis of their percent identity. The blocks used to derive the BLOSUM62 matrix, for example, all have at least 62 percent identity to some other member of the block.

Why, then, are the BLOSUM matrices better than the PAM matrices with respect to BLAST? One possible answer is that the extrapolation employed in PAM matrices magnifies small errors in the mutation probabilities for short evolutionary time periods. Another possibility is that the forces governing sequence evolution over short evolutionary times are different from those shaping sequences over longer intervals, and you can't estimate distant substitution frequencies without alignments from distantly related proteins.

4.4 Target Frequencies, lambda, and H

The most important property of a scoring matrix is its target frequencies and the expected frequencies of the individual amino acid pairs. Target frequencies represent the underlying evolutionary model. While scoring matrices don't actually contain the target frequencies, they are implicit in the scores.

The Karlin-Altschul statistical theory on which BLAST is based (discussed in the next section) states that all scoring schemes for which a positive score is possible (and the expected score is negative) have implicit target frequencies. Thus they are lod-odds scoring schemes; even a simple "+1 match -1 mismatch" scheme is implicitly a log-odds scoring scheme and has target frequencies. You'll learn how to calculate those frequencies in just a bit, but you first need to understand two additional concepts associated with scoring schemes: lambda and relative entropy.

4.4.1 Lambda

Raw score can be a misleading quantity because scaling factors are arbitrary. A normalized score, corresponding to the original lod score, is therefore a more useful measure. Converting a raw score to a normalized score requires a matrix-specific constant called lambda (or λ). Lambda is approximately the inverse of the original scaling factor, but its value may be slightly different due to integer rounding errors. Let's now derive lambda.

When calculating target frequencies from multiple alignments, the sum of all target frequencies naturally sums to 1 (see Figure 4-6).

Figure 4-6. Equation 4-4

$$\sum_{i=1}^{n} \sum_{j=1}^{i} q_{ij} = 1$$

Recall from Figure 4-4 that the score of two amino acids is the log-odds ratio of the observed and expected frequencies. The same equation is presented in Figure 4-7, but the lod score is replaced by the product of lambda and the raw score (in other words, lambda had a value of 1 in Figure 4-4).

Figure 4-7. Equation 4-5

$$\lambda S_{ij} = \log_e(q_{ij}/p_i p_j)$$

Figure 4-8 rearranges Figure 4-7 to solve for pair-wise frequency.

Figure 4-8. Equation 4-6

$$q_{ij} = p_i p_j e^{\lambda s_{ij}}$$

From Figure 4-8, you can see that a pair-wise frequency (*q*ij) is implied from individual amino acid frequencies (*p*i and *p*j) and a normalized score (λ*S*ij). The key to solving for lambda is to provide the individual amino acid frequencies (*p*i and *p*j) and find a value for lambda where the sum of the implied target frequencies equals one. The formulation is given in Figure 4-9 and later in Figure 4-1.

Figure 4-9. Equation 4-7

$$\sum_{i=1}^{n} \sum_{j=1}^{i} q_{ij} = \sum_{i=1}^{n} \sum_{j=1}^{i} p_i p_j e^{\lambda s_{ij}} = 1$$

Normally, once lambda is estimated, it is used to calculate the Expect of every HSP in the BLAST report. Unfortunately, the residue frequencies of some proteins deviate widely from the residue frequencies used to construct the original scoring matrix. Recently, some versions of PSI-BLAST and BLASTP have therefore begun to use the query and subject sequence amino acid compositions to calculate a composition-based lambda. These "hit-specific" lambdas have been shown to improve BLAST sensitivity, so this approach may see wider use in the near future.

4.4.2 Relative Entropy

4.5 Sequence Similarity

Sequence similarity is a simple extension of amino acid or nucleotide similarity. To determine it, sum up the individual pair-wise scores in an alignment. For example, the raw score of the following BLAST alignment under the BLOSUM62 matrix is 72. Converting 72 to a normalized score is as simple as multiplying by lambda. (Note that for BLAST statistical calculations, the normalized score is λS - ln$k$.)

```
Query:   885 QCPVCHKKYSNALVLQQHIRLHTGE 909
             +C VC K ++     L++H RLHTGE
Sbjct:   267 ECDVCSKSFTTKYFLKKHKRLHTGE 291
```

Recall from Chapter 3 that the score of each pair of letters is considered independently from the rest of the alignment. This is the same idea. There is a convenient synergy between alignment algorithms and alignment scores. However, when treating the letters independently of one another, you lose contextual information. Can you assume that the probability of A followed by G is the same as the probability of G followed by A? In a natural language such as English, you know that this doesn't make sense. In English, Q is always followed by U. If you treat these letters independently, you lose this restriction. The context rules for biological sequences aren't as strict as for English, but there are tendencies. For example, low entropy sequences appear by chance much more frequently than expected. To avoid becoming sidetracked by the details, accept that you're using an approximation, and note that in practice, it works well.

4.6 Karlin-Altschul Statistics

In 1990, Samuel Karlin and Stephen Altschul published a theory of local alignment statistics. Karlin-Altschul statistics make five central assumptions:

- A positive score must be possible.
- The expected score must be negative.
- The letters of the sequences are independent and identically distributed (IID).
- The sequences are infinitely long.
- Alignments don't contain gaps.

The first two assumptions are true for any scoring matrix estimated from real data. The last three assumptions are problematic because biological sequences have context dependencies, aren't infinitely long, and are frequently aligned with gaps. You now know that both alignment and sequence similarity assume independence, and that this is a necessary convenience. You will soon see how sequence length and gaps can be accounted for. For now, though, let's turn to the Karlin-Altschul equation (see Figure 4-12):

Figure 4-12. Equation 4-10

$$E = kmne^{-\lambda S}$$

This equation states that the number of alignments expected by chance (E) during a sequence database search is a function of the size of the search space ($m*n$), the normalized score (λS), and a minor constant (k).

In a database search, the size of the search space is simply the product of the number of letters in the query (m) and the number of letters in the database (n). A small adjustment (k) takes into account the fact that optimal local alignment scores for alignments that start at different places in the two sequences may be highly correlated. For example, a high-scoring alignment starting at residues 1, 1 implies a pretty high alignment score for an alignment starting at residues 2, 2 as well. The value of k is usually around 0.1, and its impact on the statistics of alignment scores is relatively minor, so don't bother with its derivation. According to Figure 4-12 the relationship between the expected number of alignments and the search space (mn) is linear. If the size of the search space is doubled, the expected number of alignments with a particular score also doubles. The relationship between the expected number of alignments and score is exponential. This means that small changes in score can lead to large differences in E.

4.6.1 Gapped Alignments

In practice, gaps reduce the stringency of a scoring scheme. In other words, an alignment score of 30 occurs more often in collection of gapped alignments than it does in a similar collection of ungapped alignments. How much more often depends on the cost of the gaps relative to the scoring matrix values. To determine the statistical significance of gapped alignments with Karlin-Altschul statistics (Figure 4-12), you must find values for lambda, k, and H for a particular scoring matrix and its associated gap initiation and extension costs. Unfortunately, you can't do this analytically, and the values must be estimated empirically. The procedure involves aligning random sequences with a specific scoring scheme and observing the alignment properties (scores, target frequencies, and lengths). The ungapped scoring matrix whose behavior is most similar to the gapped scoring scheme provides values for lambda, k, and H.

In the Karlin-Altschul theory, the distribution of alignment scores follows an extreme value distribution, a distribution that looks similar to a normal (Gaussian) distribution but falls off more quickly on one side and more slowly on the

4.7 Sum Statistics and Sum Scores

BLAST uses Figure 4-16 to calculate the normalized score of an individual HSP, but it uses a different function to calculate the normalized score of group of HSPs (see Chapter 7 for more information about sum statistics).

Figure 4-16. Equation 4-14

$$S'_{nats} = \lambda S - \ln k$$

Before tackling the actual method used by BLAST to calculate a sum score, it's helpful to consider the problem from a general perspective. One simple and intuitive approach for calculating a sum score might be to sum the raw scores, Sr for a set of HSPs, and then convert that sum to a normalized score by multiplying by λ, or in mathematical terms (see Figure 4-17).

Figure 4-17. Equation 4-15

$$S'_{sum} = \lambda \sum_{i=1}^{r} S_r$$

The problem with such an approach is that summing the scores for a collection of r HSPs, always results in a higher score, even if none or those HSPs has a significant score on its own. In practice, BLAST controls for this by penalizing the sum score by a factor proportional to the product of the number of HSPs, r, and the search space as shown in Figure 4-18.

Figure 4-18. Equation 4-16

$$S'_{sum} = \lambda \sum_{i=1}^{r} S_r - r\ln(kmn)$$

Figure 4-18 is sometimes referred to as an unordered-sum score and is suitable for calculating the sum score for a collection of noncollinear HSPs. Ideally, though, you should use a sum score formulation that rewards a collection of HSPs if they are collinear with regards to their query and subject coordinates because the HSPs comprising real BLAST hits often have this property. BLASTX hits for example often consist of collinear HSPs corresponding to the sequential exons of a gene. Figure 4-19 is a sum score formulation that does just that.

Figure 4-19. Equation 4-17

$$S'_{sum} = \lambda \sum_{i=1}^{r} S_r - r\ln(kmn) + \ln(r!)$$

Figure 4-20 is sometimes referred to as a pair-wise ordered sum score. Note the additional term lnr!, which can be thought of as a bonus added to the sum score when the HSPs under consideration are all consistently ordered.

One shortcoming of Figure 4-18 and Figure 4-19 is that they invoke a sizable penalty for adding an additional HSP raw score to the sum score. To improve the sensitivity of its sum statistics, NCBI-BLASTX employs a modified version of the pair-wise ordered sum score (Figure 4-19) that is influenced less by the search space and contains a term for the size of the gaps between the HSPs (Figure 4-20). The advantage of this formulation is that the gap size, g, rather than the search space, mn, is multiplied by r. For short gaps and big search spaces, this formulation results in larger sum scores.

Figure 4-20. Equation 4-18

$$S'_{sum} = \lambda \sum_{i=1}^{r} S_r - \ln(kmn) - (r-1)\cdot(\ln(k) + 2\ln(g)) - \log(r!)$$

4.7.1 Converting a Sum Score to a Sum Probability

The aggregate pair-wise P-value for a sum score can be approximated using Figure 4-21.

Figure 4-21. Equation 4-19

4.8 Further Reading

Altschul, S.F. (1991). "Amino acid substitution matrices from an information theoretic perspective." Journal of Molecular Biology, Vol. 219, pp. 555-565. Altschul, S.F. (1997). "Evaluating the statistical significance of multiple distinct local alignments." Theoretical and Computational Methods in Genome Research, S. Suhai (ed.), pp. 1-14. Altschul, S.F. (1993). "A protein alignment scoring system sensitive at all evolutionary distances." Journal of Molecular Evolution, Vol. 36, pp. 290-300. Altschul, S.F., M.S. Boguski, W. Gish, and J.C. Wootton (1994). "Issues in searching molecular sequence databases." Nature Genet., Vol. 6, pp. 119-129. Altschul S.F., R. Bundschuh, R. Olsen, T. Hwa (2001) "The estimation of statistical parameters for local alignment score distributions." Nucleic Acids Research, January 15;29(2), pp. 351-361. Altschul, S.F. and W. Gish (1996). "Local alignment statistics." Meth. Enzymol., Vol. 266, pp. 460-480. Dayhoff, M.O., R.M. Schwartz, and B.C. Orcutt (1978). "A model of evolutionary change in proteins." Atlas of Protein Sequence and Structure, Vol. 5, Suppl. 3. M.O. Dayhoff (ed.), National Biomedical Research Foundation, pp. 345-352. Henikoff, S. and J.G. Henikoff (1992). "Amino acid substitution matrices from protein blocks." Proceedings of the National Academy of Sciences, Vol. 89, pp. 10915-10919. Karlin, S. and S.F. Altschul (1993). "Applications and statistics for multiple high-scoring segments in molecular sequences." Proceedings of the National Academy of Sciences, Vol. 90, pp. 5873-5877. Karlin, S. and S. F. Altschul (1990). "Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes." Proceedings of the National Academy of Sciences, Vol. 87, pp. 2264-2268. Schaffer A.A., Aravind L., Madden TL., Shavirin S., Spouge JL., Wolf YI., Koonin EV., Altschul SF. (2001). "Improving the accuracy of PSI-BLAST protein database searches with composition-based statistics and other refinements." Nucleic Acids Research, July 15;29(14), pp. 2994-3005. Schwartz, R.M. and M.O. Dayhoff (1978). "Matrices for detecting distant relationships." Atlas of Protein Sequence and Structure, Vol. 5, Suppl. 3. M.O. Dayhoff (ed.), National Biomedical Research Foundation, pp. 353-358.

Part III: Practice

Chapter 5. BLAST

Previous chapters explored what biological sequences are, how they are aligned, and how similarity is measured. This chapter discusses BLAST itself. What is BLAST? The simple answer is that it is a set of programs that search sequence databases for statistically significant similarities. The details of how BLAST searches for similarities aren't so easily answered. Searching requires multiple steps and many controlling parameters. Understanding the theoretical framework will help you design and interpret BLAST experiments, and give you a foundation for troubleshooting when your search produces unexpected results.

5.1 The Five BLAST Programs

The five traditional BLAST programs are: BLASTN, BLASTP, BLASTX, TBLASTN, and TBLASTX. BLASTN compares nucleotide sequences to one another (hence the N). All other programs compare protein sequences (see Table 5-1).

Table 5-1. Traditional BLAST programs

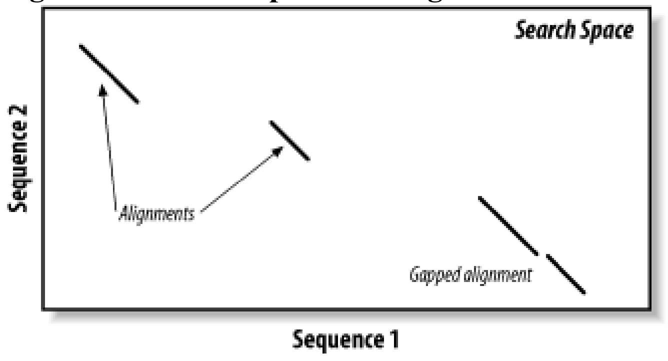
Program	**Database**	**Query**	**Typical uses**
BLASTN	Nucleotide	Nucleotide	Mapping oligonucleotides, cDNAs, and PCR products to a genome; screening repetitive elements; cross-species sequence exploration; annotating genomic DNA; clustering sequencing reads; vector clipping
BLASTP	Protein	Protein	Identifying common regions between proteins; collecting related proteins for phylogenetic analyses
BLASTX	Protein	Nucleotide translated into protein	Finding protein-coding genes in genomic DNA; determining if a cDNA corresponds to a known protein
TBLASTN	Nucleotide translated into protein	Protein	Identifying transcripts, potentially from multiple organisms, similar to a given protein; mapping a protein to genomic DNA
TBLASTX	Nucleotide translated into protein	Nucleotide translated into protein	Cross-species gene prediction at the genome or transcript level; searching for genes missed by traditional methods or not yet in protein databases

You'll also find several BLAST derivatives and BLAST wrappers (scripts that run BLAST in a specialized way) with names such as PSI-BLAST, PHI-BLAST, MegaBLAST, BLASTZ, XBLAST, MPBLAST, HT-BLAST, and GENE-BLAST. This book—and especially this chapter—deals primarily with the five traditional programs. If you are familiar with the details of these algorithms, you will have a solid foundation for understanding the variants.

5.2 The BLAST Algorithm

The search space between two sequences can be visualized as a graph with one sequence along the X-axis and the other along the Y-axis (Figure 5-1). Each point in this space represents a pairing of two letters, one from each sequence. Each pair of letters has a score that is determined by a scoring matrix whose values were determined empirically. (See Chapter 4 for more about scoring matrices.) An alignment is a sequence of paired letters that may contain gaps (See Chapter 3 and Chapter 4 for more about gaps). Ungapped alignments appear as diagonal lines in the search space, and the score of an ungapped alignment is simply the sum of the scores of the individual letter pairs. Alignments containing gaps appear as broken diagonals in the search space, and their score is the sum of the letter pairs minus the gap costs, which usually penalize more score points for initiating a gap than extending a gap. How can you tell the difference between two ungapped alignments and a single gapped alignment? In a BLAST report, unaligned regions aren't displayed, and gaps are represented by dashes. However, a simple change in parameters can change one into the other. The diagrams in this chapter show only one gapped alignment, which is indicated in Figure 5-1.

Figure 5-1. Search space and alignment

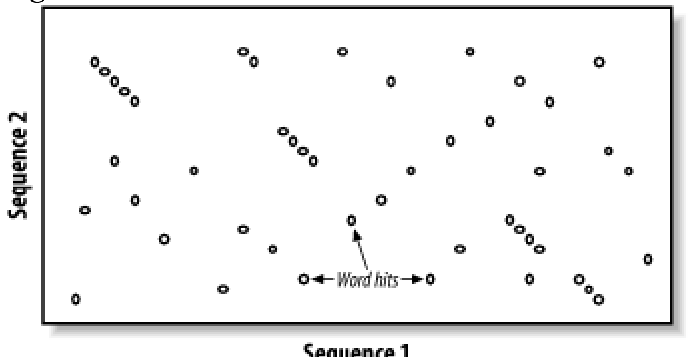


As you saw in Chapter 3, the Smith-Waterman algorithm will find the maximum scoring alignment between two sequences. Some people feel that this ability makes Smith-Waterman the gold standard of alignment algorithms, but this is true only in theory. When comparing real sequences, you may have several good alignments or none. What you really want reported is all of the statistically significant alignments; this is what BLAST does. However, unlike Smith-Waterman, BLAST doesn't explore the entire search space between two sequences. Minimizing the search space is the key to its speed but at the cost of a loss in sensitivity. You will find that the speed/sensitivity trade-off is a key concept when designing BLAST experiments. How exactly does BLAST find similarities without exploring the entire search space? It uses three layers of rules to sequentially refine potential high scoring pairs (HSPs). These heuristic layers, known as seeding, extension, and evaluation, form a stepwise refinement procedure that allows BLAST to sample the entire search space without wasting time on dissimilar regions.

5.2.1 Seeding

BLAST assumes that significant alignments have words in common. A word is simply some defined number of letters. For example, if you define a word as three letters, the sequence MGQLV has words MGQ, GQL, and QLV. When comparing two sequences, BLAST first determines the locations of all the common words, which are called word hits (Figure 5-2). Only those regions with word hits will be used as alignment seeds. This way, BLAST can ignore a lot of the search space.

Figure 5-2. Word hits



5.3 Further Reading

The following papers describe BLAST algorithms in more detail:

Altschul, Stephen F., Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman (1990). "Basic local alignment search tool," Journal of Molecular Biology, 215:403-10. Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997), "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," Nucleic Acids Research, 25:3389-3402. Gish, Warren and David J. States (1993). "Identification of protein coding regions by database similarity search," Nature Genet., 3:266-72.

For the latest information on NCBI-BLAST and other helpful documents, see the official web site at http://ncbi.nlm.nih.gov/BLAST.

The WU-BLAST web site (http://blast.wustl.edu) has several useful documents on BLAST in general, and includes pages specifically for WU-BLAST users. One of its most interesting documents is a thorough manual for the classic, ungapped version of BLAST 1.4 (http://blast.wustl.edu/doc/blast1.pdf).

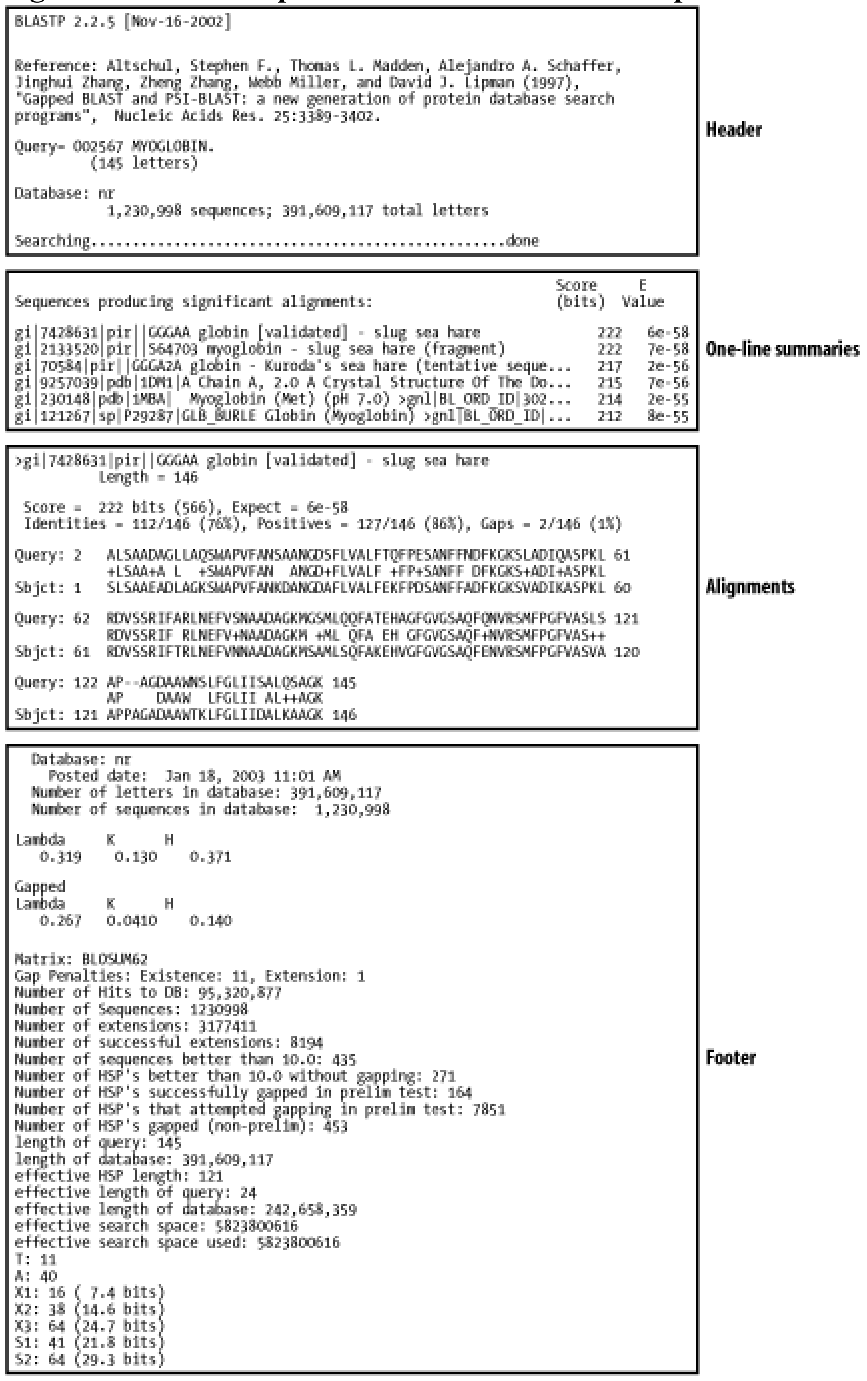
Chapter 6. Anatomy of a BLAST Report

This chapter explores the standard BLAST format. NCBI-BLAST and WU-BLAST are very similar, and their few important differences are described below. NCBI-BLAST offers several additional output options described in Appendix A.

6.1 Basic Structure

A standard BLAST report has four parts (see Figure 6-1).

Figure 6-1. The four parts of a standard BLAST report



```
BLASTP 2.2.5 [Nov-16-2002]


Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs",  Nucleic Acids Res. 25:3389-3402.

Query= 002567 MYOGLOBIN.
         (145 letters)

Database: nr
           1,230,998 sequences; 391,609,117 total letters

Searching..................................................done
```

```
                                                                 Score     E
Sequences producing significant alignments:                      (bits)  Value

gi|7428631|pir||GGGAA globin [validated] - slug sea hare            222   6e-58
gi|2133520|pir||S64703 myoglobin - slug sea hare (fragment)         222   7e-58
gi|70584|pir||GGGA2A globin - Kuroda's sea hare (tentative seque...  217   2e-56
gi|9257039|pdb|1DM1|A Chain A, 2.0 A Crystal Structure Of The Do...  215   7e-56
gi|230148|pdb|1MBA|  Myoglobin (Met) (pH 7.0) >gnl|BL_ORD_ID|302...  214   2e-55
gi|121267|sp|P29287|GLB_BURLE Globin (Myoglobin) >gnl|BL_ORD_ID|...  212   8e-55
```

```
>gi|7428631|pir||GGGAA globin [validated] - slug sea hare
          Length = 146

 Score =  222 bits (566), Expect = 6e-58
 Identities = 112/146 (76%), Positives = 127/146 (86%), Gaps = 2/146 (1%)

Query: 2   ALSAADAGLLAQSWAPVFANSAANGDSFLVALFTQFPESANFFNDFKGKSLADIQASPKL 61
           +LSAA+A L  +SWAPVFAN  ANGD+FLVALF +FP+SANFF DFKGKS+ADI+ASPKL
Sbjct: 1   SLSAAEADLAGKSWAPVFANKDANGDAFLVALFEKFPDSANFFADFKGKSVADIKASPKL 60

Query: 62  RDVSSRIFARLNEFVSNAADAGKMGSMLQQFATEHAGFGVGSAQFQNVRSMFPGFVASLS 121
           RDVSSRIF RLNEFV+NAADAGKM +ML QFA EH GFGVGSAQF+NVRSMFPGFVAS++
Sbjct: 61  RDVSSRIFTRLNEFVNNAADAGKMSAMLSQFAKEHVGFGVGSAQFENVRSMFPGFVASVA 120

Query: 122 AP--AGDAAWNSLFGLIISALQSAGK 145
           AP    DAAW  LFGLII AL++AGK
Sbjct: 121 APPAGADAAWTKLFGLIIDALKAAGK 146
```

```
  Database: nr
    Posted date:  Jan 18, 2003 11:01 AM
  Number of letters in database: 391,609,117
  Number of sequences in database:  1,230,998

Lambda     K      H
   0.319    0.130    0.371

Gapped
Lambda     K      H
   0.267   0.0410    0.140


Matrix: BLOSUM62
Gap Penalties: Existence: 11, Extension: 1
Number of Hits to DB: 95,320,877
Number of Sequences: 1230998
Number of extensions: 3177411
Number of successful extensions: 8194
Number of sequences better than 10.0: 435
Number of HSP's better than 10.0 without gapping: 271
Number of HSP's successfully gapped in prelim test: 164
Number of HSP's that attempted gapping in prelim test: 7851
Number of HSP's gapped (non-prelim): 453
length of query: 145
length of database: 391,609,117
effective HSP length: 121
effective length of query: 24
effective length of database: 242,658,359
effective search space: 5823800616
effective search space used: 5823800616
T: 11
A: 40
X1: 16 ( 7.4 bits)
X2: 38 (14.6 bits)
X3: 64 (24.7 bits)
S1: 41 (21.8 bits)
S2: 64 (29.3 bits)
```

Header

The first line contains the name of the program, its version, and its build date. If BLAST crashes or has some kind of unexpected behavior, include this information when you report the problem to the authors. The next piece of information is a reference to the scientific literature, which should be cited if you publish research that employed BLAST. The most important information in the header, the names of the query sequence and the database, appear next. The last line is a progress meter that is updated during the search.

One-line summaries

Each line indicates the name of the sequence, the highest scoring alignment found and the lowest E-value for any HSP or group of HSPs. The one-line summaries are often hyperlinked to the alignments farther below when the output comes from a web page. If you just want to know for example, the names of the top matches, these one-line summaries are convenient.

Alignments

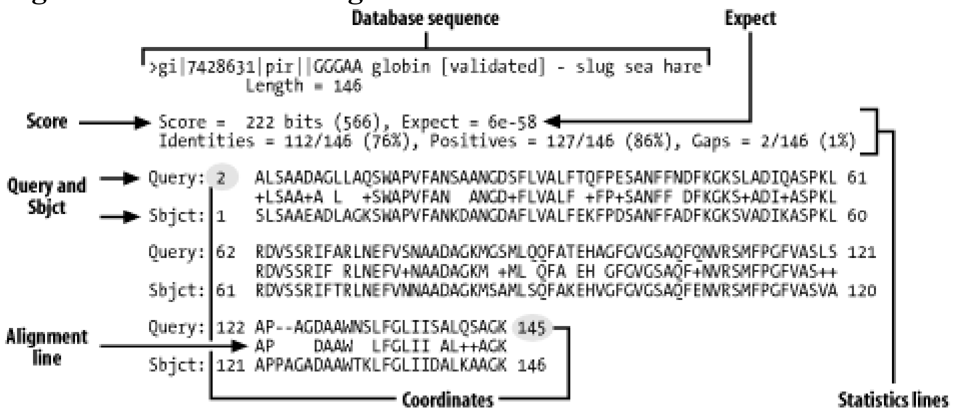
6.2 Alignments

The alignments and alignment statistics reported by BLAST differ slightly from program to program. The rest of this chapter describes the details of BLASTP, BLASTN, BLASTX, TBLASTN, and TBLASTX alignments and shows how to recognize alignment groups.

6.2.1 BLASTP

BLASTP alignments are the simplest to understand. Figure 6-2 shows the anatomy of a typical BLASTP alignment.

Figure 6-2. A BLASTP alignment



```
>gi|7428631|pir||GGGAA globin [validated] - slug sea hare
          Length = 146

 Score =  222 bits (566), Expect = 6e-58
 Identities = 112/146 (76%), Positives = 127/146 (86%), Gaps = 2/146 (1%)

Query: 2   ALSAADAGLLAQSWAPVFANSAANGDSFLVALFTQFPESANFFNDFKGKSLADIQASPKL 61
            +LSAA+A L  +SWAPVFAN  ANGD+FLVALF +FP+SANFF DFKGKS+ADI+ASPKL
Sbjct: 1   SLSAAEADLAGKSWAPVFANKDANGDAFLVALFEKFPDSANFFADFKGKSVADIKASPKL 60

Query: 62  RDVSSRIFARLNEFVSNAADAGKMGSMLQQFATEHAGFGVGSAQFQNVRSMFPGFVASLS 121
            RDVSSRIF RLNEFV+NAADAGKM +ML QFA EH GFGVGSAQF+NVRSMFPGFVAS++
Sbjct: 61  RDVSSRIFTRLNEFVNNAADAGKMSAMLSQFAKEHVGFGVGSAQFENVRSMFPGFVASVA 120

Query: 122 AP--AGDAAWNSLFGLIISALQSAGK 145
            AP    DAAW  LFGLII AL++AGK
Sbjct: 121 APPAGADAAWTKLFGLIIDALKAAGK 146
```

Here are the parts you should pay attention to:

Score

This value is computed from the scoring matrix and gap penalties. A higher score indicates greater similarity. The raw score is shown without units, and the normalized score is followed by "bits."

Database sequence

The complete FASTA definition line is reported here along with the length of the sequence. All the alignments between the query and a specific database sequence are collectively called a hit. The database in Figure 6-2 has one alignment.

Expect

The number of alignments expected at random given the size of the search space, the scoring matrix, and the gap penalties. The lower the E-value, the less likely this is a random similarity.

Statistics lines

Score, E-value, and percent identity always appear here. Depending on the program, percent positive scoring, P-value, group, gaps, strand, and reading frame may also be reported.

Coordinates

The coordinates of each sequence are indicated at the beginning and ending of each line. The single alignment in Figure 6-2 is long enough that it is reported on three separate lines.

Alignment line

Letters that are identical between two sequences are reported here. Those that have positive scores in the scoring matrix are displayed with a plus sign. Gaps and nonpositive scores are blank.

Query and Sbjct

The query sequence is always listed first. The database sequence is abbreviated as Sbjct (subject).

The database sequence may be several lines long if the BLAST database is a nonredundant database with concatenated definition lines. For more on this topic, see Chapter 11. The WU-BLAST format differs slightly from the NCBI format: gaps aren't reported on the statistics line, and the P-value (displayed as P or Sum P) is always

Chapter 7. A BLAST Statistics Tutorial

BLAST statistics are everywhere in biology today. In fact, it's hard to find a molecular-biology paper, grant proposal, patent application, or biotech business plan that doesn't refer to the Expect or P-value of a BLAST result. The BLAST Expect has so permeated biological thinking in recent years that for many scientists it has become synonymous with biological truth. Tell a colleague that you've just cloned a gene that's homologous to something trendy, and odds are that he will ask what the Expect of its alignment was in a BLAST search.

Of course, what some see as a sweeping change, others label a fad. While some researchers consider BLAST statistics a welcome injection of mathematical rigor into the biological world, others lament the abandonment of biological insight for faith in a number. No matter where you stand on this issue, there is no avoiding the reality of BLAST statistics in today's bioinformatics workplace. Understanding what the numbers in a BLAST report mean and how they are derived isn't just for mathematicians; it's a real-world survival skill for biologists and bioinformatics professionals in academia and industry alike.

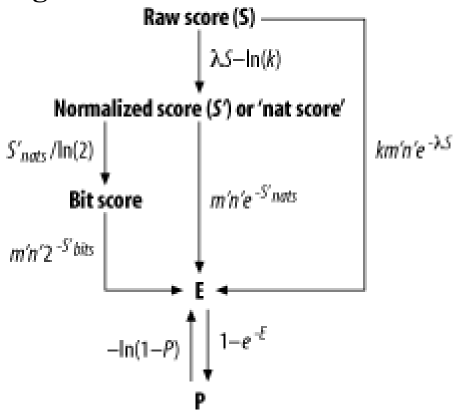
The material covered in this chapter is practical rather than theoretical in nature. Chapter 4 summarized some of the theory behind local alignment statistics. Read that chapter to learn more about the basic parameters of BLAST: λ, k, and H. This chapter shows how to calculate the numbers in a BLAST report and use this knowledge to better understand the results of a BLAST search.

7.1 Basic BLAST Statistics

Primary BLAST literature doesn't focus on the arithmetic involved in calculating an Expect. Understandably, the papers discuss the theoretical underpinnings of Karlin-Altschul statistics, leaving BLAST users without the kinds of examples found in a text book. This chapter provides this missing information.

Figure 7-1 summarizes the basic operations involved in converting the raw score of a high-scoring pair into an Expect, or P-value. The following discussion explains how to perform each calculation and any accessory operations required to derive an Expect.

Figure 7-1. The essential calculations involved in converting an HSP's raw score into a probability



Calculating Karlin-Altschul statistics isn't easily done with a pencil, paper, and calculator. Applying Karlin-Altschul statistics to an HSP and avoiding a migraine requires code. We use simple Perl subroutines to take the grind out of the calculations and help clarify the equations. Hopefully these subroutines will also provide a basis for further investigations of your own BLAST results. If you'd like to add these subroutines to your codebase, you can download them from this book's web site; the module name is BlastStats.pm.

Example 7-1 shows a sample HSP from a BLASTP search. Use this alignment as a reference to check your calculations.[1] For our present purposes, the details of the search aren't important, but the values given in the header and footer of the report are. The header and footer are omitted from the search that generated this alignment.

[1] If you plan to perform the calculations described in this chapter and have searched the NCBI web site (http://www.ncbi.nlm.nih.gov/BLAST), make sure you don't select the composition-based statistics in the Advanced Options checkbox because the values of the composition-based lambdas aren't given in the report. Currently, it's selected by default for BLASTP, and you unselect it by clicking on it.

Example 7-1. A sample HSP from a BLASTP search

```
>CT12121 translation from_gene[CG3603 CG3603 FBgn0029648]
          seq_release:2  mol_weight=18568
          cds_boundaries:(X:2,699,992..2,700,667[+]) aa_length:176
          transcript_info:[CT12121  seq_release:2] gene_info:[gene
          symbol:CG3603 FBgn0029648
          gene_boundaries:(X:2,699,935..2,700,670[+]) (GO:0016614
          "oxidoreductase, acting on CH-OH group of donors")]
         Length = 176

 Score = 70.9 bits (172), Expect = 8e-13
 Identities = 49/170 (28%), Positives = 85/170 (49%), Gaps = 6/170 (3%)

Query: 50   IAGEVAVVTGAGHGLGRAISLELAKKGCHIAVVDINVSGAEDTVKQIQDIYKVRAKAYKA 109
          +AG+VA+VTGAG G+GRA    LA+ G   +  VD N+  A++TV    Q++    R+ A +
Sbjct: 6    LAGKVALVTGAGSGIGRATCRLLARDGAKVIAVDRNLKAAQETV---QELGSERSAALEV 62

Query: 110  NVTNYDDLVELNSKVVEEMGPV-TVLVNNAGVMMHRNMFNPDPADVQLMINVNLTSHFWT 168
          +V++    +    ++ +++      T++VN+AG+      +        D    +   VNL    F
Sbjct: 63   DVSSAQSVQFSVAEALKKFQQAPTIVVNSAGITRDGYLLKMPERDYDDVYGVNLKGTFLV 122
```

7.2 Using Statistics to Understand BLAST Results

Karlin-Altschul statistics is much more than a way to determine the statistical significance of a sequence alignment in the context of a database search. It also provides a framework with which to probe the complex relationships that exist between BLAST parameters and results. Using Karlin-Altschul statistics to ask and answer questions about a BLAST search is much like using stoichiometry at the lab bench; it doesn't require theoretical savvy, just a little algebra. It's also useful; you no longer need to be frustrated when confronted with an inexplicable BLAST result.

Now let's look at a practical application of Karlin-Altschul statistics: using BLASTN to map a PCR primer to a genome. The application is a simple but striking example of how to use Karlin-Altschul statistics to understand the way parameter choice determines BLAST results. Finally, Karlin-Altschul statistics reveal much about BLASTN's strengths and weaknesses and its potential as a tool to detect the conserved, cis-regulatory regions of genes.

7.3 Where Did My Oligo Go?

First, try to identify the position of the following oligo-nucleotide in the Drosophila melanogaster genome using WU-BLASTN with its default parameters:

```
TACATCCGGCACTTAGCCGGGCTCG
```

Example 7-4 shows that the oligo isn't found in the Drosophila melanogaster genome that uses WU-BLASTN with default parameters.

Example 7-4. The oligo isn't found

```
Reference:  Gish, W. (1996-2000) http://blast.wustl.edu

Notice:  this program and its default parameter settings are optimized to find
nearly identical sequences rapidly.  To identify weak similarities encoded in
nucleic acid, use BLASTX, TBLASTN or TBLASTX.

Query=  oligo
        (25 letters)

Database:  na_whole-genome_genomic_dmel_RELEASE3.FASTA
           7 sequences; 124,181,667 total letters.
Searching....10....20....30....40....50....60....70....80....90....100% done

                                                                     Smallest
                                                                       Sum
                                                              High  Probability
Sequences producing High-scoring Segment Pairs:              Score  P(N)      N

      *** NONE ***
```

There are, of course, many reasons why you might not be able to identify an oligo in the Drosophila melanogaster genome. First, the oligo might contain repetitive sequence and thus be masked out. However, because WU-BLAST doesn't mask by default, that can't be the reason. Second, the assembled genome may be incomplete. Every sequenced genome to date is incomplete to some degree. In fact, a 99 percent complete 124mb genome is still missing 1.24 mega-bases of a euchromatic (nonrepetitive DNA) sequence, leaving plenty of space for an oligo to go missing in. The incompleteness of the genome is a possible explanation for our WU-BLAST result, but is it the correct one? Before concluding that the oligo falls into a sequencing gap, let's try to run NCBI-BLASTN with its default parameters. Aha! The NCBI-BLASTN results in Example 7-5 show that the oligo is present in the Drosophila melanogaster genome and the HSP is assigned a significant Expect.

Example 7-5. Using NCBI-BLASTN to find the oligo

```
Sequences producing significant alignments:                    (bits)  Value

2R 2R.3 assembled 23-11-2001                                       50   1e-06
X X release:2 length:21666217bp Assembled X chromosome arm seque...    32   0.25
3R 3R.3                                                            32   0.25
U GenomicInterval:U                                                30   0.99
3L 3L.3 v.3e  23351213bp BCM HGSC guide:3l-mtp-eval.08apr02        28   3.9
2L 2L release:3 length:22217931bp Assembled 2L chromosome arm se...    28   3.9

>2R 2R.3 assembled 23-11-2001
          Length = 20302755

 Score = 50.1 bits (25), Expect = 1e-06
 Identities = 25/25 (100%)
 Strand = Plus / Plus

Query: 1        tacatccggcacttagccgggctcg 25
                |||||||||||||||||||||||||
Sbjct: 16190927 tacatccggcacttagccgggctcg 16190951
```

Results like these frustrate a lot of BLAST users. Why does NCBI-BLAST find the oligo when WU-BLAST

Chapter 8. 20 Tips to Improve Your BLAST Searches

8.1 Don't Use the Default Parameters

You shouldn't use BLAST the way you use an Internet search engine such as Google. BLAST results are very sensitive to parameters, and the defaults aren't suitable for all searches. Historically, BLAST parameters have changed periodically. In addition, the NCBI-BLAST and WU-BLAST defaults have very different properties, and your results may differ depending on where you perform your BLAST search. Thus, you should get out of the habit of using default parameters. There are situations in which the default parameters are just fine, but using them knowledgeably and accepting them out of ignorance are two very different motivations. If you don't know which settings are appropriate for a particular search, you're not alone; most BLAST users don't know how to set up a search. That's why we wrote this book, so keep reading.

8.2 Treat BLAST Searches as Scientific Experiments

Scientists are often taught to structure their experiments into four parts: the question (hypothesis), the approach (experimental design), the results (data), and the interpretations (beliefs). This approach shows that beliefs depend on the experiment's data. Whether or not an experiment is capable of answering the question is one way to separate good science from bad.

When setting up a BLAST experiment, the most important thing to remember is "you get what you look for." In other words, search parameters determine what you find. For example, the BLASTN program from NCBI with the default settings assumes that the alignments you are seeking are nearly identical because the parameters (match +1, mismatch -3) have a target frequency of 99 percent identity. If your experimental question is "How many worm genes are related to my favorite human gene," using the default parameters would be foolish because the approach (looking for nearly identical sequences) isn't expected to answer the question; too many sequences have changed in the 500 million years that separate worms and humans.

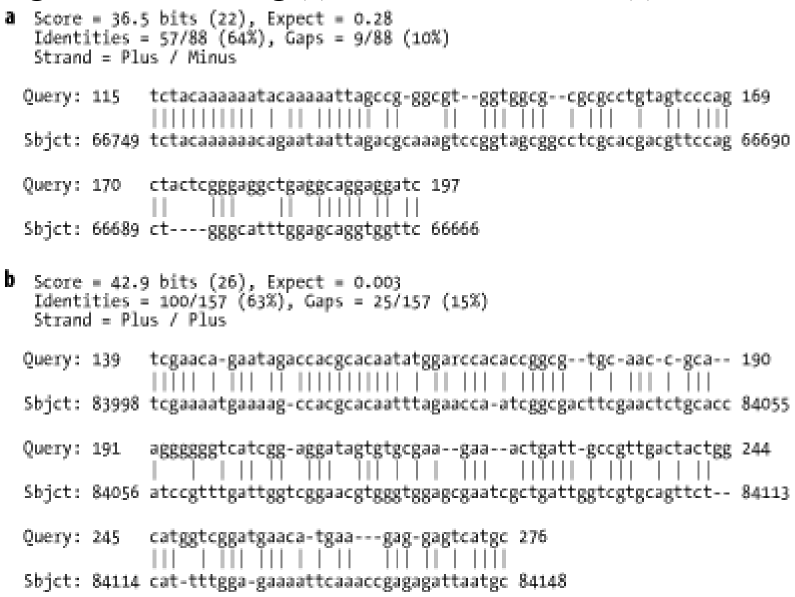
8.3 Perform Controls, Especially in the Twilight Zone

Controls are crucial to any scientific experiment. The random model underlying BLAST statistics provides one kind of control, but performing an explicit control can give you greater confidence in your results. This is especially true when looking for weak similarities, commonly called the twilight zone. One of the simplest and most effective ways to determine if an alignment is believable is to shuffle your query sequence and repeat the search. If the shuffled sequence returns similar results, the alignment is based on compositional biases or the search parameters aren't specific enough. The following Perl script shuffles a FASTA file:

```
#!/usr/bin/perl -w
use strict;

my ($def, @seq) = <>;
print $def;
chomp @seq;
@seq = split(//, join("", @seq));
my $count = 0;
while (@seq) {
    my $index = rand(@seq);
    my $base = splice(@seq, $index, 1);
    print $base;
    print "\n" if ++$count % 60 == 0;
}
```

Now let's put this script into action. Let's make the dubious hypothesis that ALU repeats aren't specific to primates but are present in all genomes. They haven't been found because people just haven't looked hard enough. Your search parameters use +1/-1 match/mismatch scores and a gap opening cost of 1 and extension cost of 1. (WU-BLAST users would understand this as a cost of 2 for the first gap and 1 for each additional gap). Figure 8-1a shows an alignment between a human ALU (a variety of repeats are available from ftp://ftp.ncbi.nih.gov/repository/repbase) and the *Caenorhabditis elegans* genome (see http://www.wormbase.org). Without a control, you might be able to convince yourself that you found a match to a *C. elegans* ALU. However, because a shuffled control (Figure 8-1b) produces an alignment that is approximately 100 times more significant, this conclusion isn't very likely.

Figure 8-1. Searching (a) an ALU element and (b) a shuffled version against the C. elegans genome



```
a  Score = 36.5 bits (22), Expect = 0.28
   Identities = 57/88 (64%), Gaps = 9/88 (10%)
   Strand = Plus / Minus

  Query: 115   tctacaaaaaatacaaaaattagccg-ggcgt--ggtggcg--cgcgcctgtagtcccag 169
               ||||||||||| | || |||||| ||    ||  ||| |||  | |||  |  || ||||
  Sbjct: 66749 tctacaaaaaacagaataattagacgcaaagtccggtagcggcctcgcacgacgttccag 66690

  Query: 170   ctactcgggaggctgaggcaggaggatc 197
               ||    |||    ||  ||||| || ||
  Sbjct: 66689 ct----gggcatttggagcaggtggttc 66666

b  Score = 42.9 bits (26), Expect = 0.003
   Identities = 100/157 (63%), Gaps = 25/157 (15%)
   Strand = Plus / Plus

  Query: 139   tcgaaca-gaatagaccacgcacaatatggarccacaccggcg--tgc-aac-c-gca-- 190
               ||||| | ||| || ||||||||||| | || ||| | |||||  | | ||| | |||
  Sbjct: 83998 tcgaaaatgaaaag-ccacgcacaatttagaacca-atcggcgacttcgaactctgcacc 84055

  Query: 191   aggggggtcatcgg-aggatagtgtgcgaa--gaa--actgatt-gccgttgactactgg 244
               |   |  | || ||  |||  ||| | | |  |||   |||||| | |||  | | ||
  Sbjct: 84056 atccgtttgattggtcggaacgtgggtggagcgaatcgctgattggtcgtgcagttct-- 84113

  Query: 245   catggtcggatgaaca-tgaa---gag-gagtcatgc 276
               |||  | ||| ||| | | ||   ||| || | ||||
  Sbjct: 84114 cat-tttgga-gaaaattcaaaccgagagattaatgc 84148
```

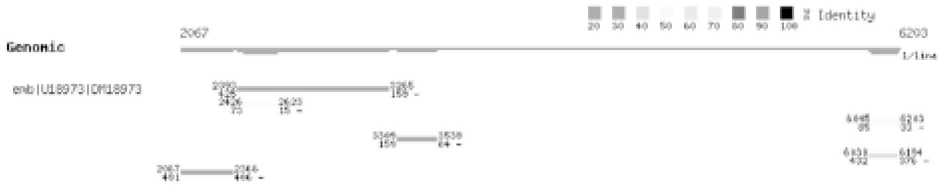
You might wonder why the alignments in Figure 8-1 seem to have significant E-values. The search employed low gap penalties and ungapped alignment statistics. When using gapped alignment statistics, these alignments are expected at random.

8.4 View BLAST Reports Graphically

BLAST reports can be complicated. Viewing them graphically can help you understand them better, especially when the reports are very long. Appendix D includes a simple Perl program that converts tabular output from NCBI-BLAST reports into a GIF, PNG, or JPEG image. Figure 8-2, Figure 8-4, Figure 8-8, and Figure 8-9 were created with this program. Appendix E contains a program that converts the standard BLAST reports to the NCBI tabular format. The programs are available at this book's O'Reilly web site.

Figure 8-2 is an example of a BLASTX search. Appendix D contains more information on the display.

Figure 8-2. A graphical view of a BLASTX report



8.5 Use the Karlin-Altschul Equation to Design Experiments

The Karlin-Altschul equation is very useful for predicting the outcome of a BLAST experiment, especially in large search spaces. Suppose you want to find exons in the human genome by looking for similarities in the pufferfish genome. These genomes last shared a common ancestor about 450 million years ago. You might assume that any similarities at this distance must be due to evolutionary conservation.

Recall from Chapter 4 that the number of alignments expected by chance (E) is a function of the search space (M, N), the normalized score (λS), and a minor constant (K).

$$E = KMNe^{-\lambda S}$$

The typical cross-species parameters +1/-1 match/mismatch have a target frequency of 75 percent identity and 0.55 nats per aligned letter on average (*H*). A 50-bp alignment therefore contains about 27.5 nats. Substituting this normalized score into the Karlin-Altschul equation with K=0.334, M=1.5 GB (assuming half of the human genome contains repeats), and N=450 MB (the size of the repeat-poor pufferfish genome), you expect about 230,000 alignments by chance. That's roughly the same as the number of exons in the human genome. If you want to look for 50-bp exons, you'll have to sift through a lot of false positives.

To change the Karlin-Altschul expectation to something more manageable, either look for larger exons or reduce your search space. A 72-nucleotide alignment is expected only once by chance, and an alignment the size of a typical exon (110 bp) has a probability of about 1 in 1 billion of occurring. An even better approach is to restrict the search to orthologous regions of the size of a typical gene. Here 50-bp alignments have a probability of approximately 1 in 10,000.

8.6 When Troubleshooting, Read the Footer First

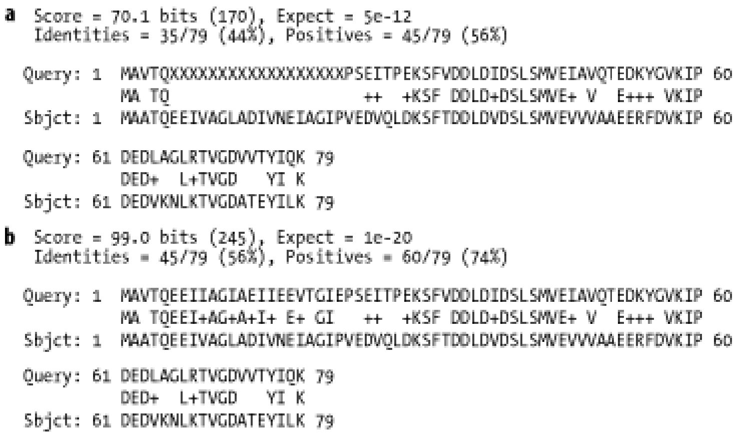
Novices usually focus on the one-line summaries, regular users concentrate on the alignments and their statistics, and professionals first read the footer. When it comes to solving the two most common problems, no hits and too many hits, the one-line summaries aren't much help. Regular users can often look at alignments and diagnose compositional biases and unidentified repeats, but determining the cause of no hits isn't easy. Examining the footer to determine what the search was actually looking for is the best way to determine what happened. Always answer the following questions first:

- What are the values for the seeding parameters W, T, and two-hit distance? If the seeding parameters are too stringent, divergent alignments may not be seeded. In NCBI-BLAST, W is unfortunately not displayed in the footer. The value for T and two-hit distance are given as T: and A:, respectively.
- What is the scoring scheme expecting to find (i.e., target frequency)? If the scoring matrix expects nearly identical sequences, highly divergent sequences may be missed.
- What is the alignment threshold? If the alignment threshold is too high, low scoring alignments will be thrown away. The gapped and ungapped values are given after S1: and S2: in NCBI-BLAST. In WU-BLAST, they are on the rows beneath S2.
- What are B and V set to? If they are set too low, the number of one-line summaries and database hits may be truncated.
- What is the score and expected length of a significant alignment? Use the Karlin-Altschul equation to solve for the normalized score and then divide by *H* to calculate the length.
- Was complexity filtering employed, and if so, was it hard or soft? Complexity filtering is generally a good idea, but may prevent some sequences from generating significant alignments. NCBI-BLAST doesn't not currently report which filters were employed.

8.7 Know When to Use Complexity Filters

Low-complexity sequence occurs much more frequently than expected by chance in both proteins and nucleic acids. When a BLAST search takes longer than expected, it is almost always due to low complexity sequence or repeats. Low-complexity filters can sometimes be destructive. Figure 8-3a shows what happens when a query sequence is filtered: the low complexity region is replaced with Xs (or Ns for nucleotide sequences). This operation always reduces the score and can terminate an alignment extension. For this reason, it is almost always better to use soft-masking (see Figure 8-3b). This technique masks low-complexity sequence in the seeding phase but allows the extension phase to see the sequence normally. See -F in Chapter 13 and wordmask in Chapter 14.

Figure 8-3. Complexity filters (a) hard-masking and (b) soft-masking



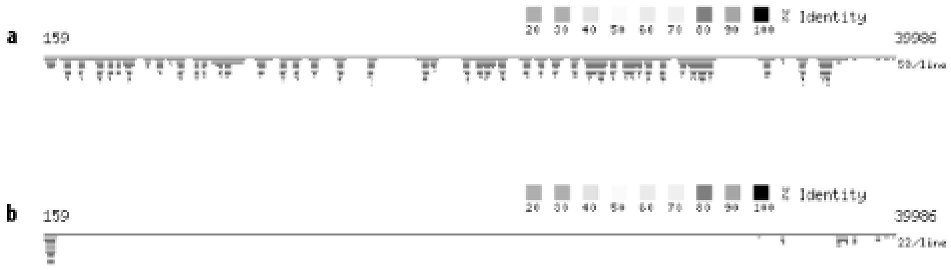
```
a  Score = 70.1 bits (170), Expect = 5e-12
   Identities = 35/79 (44%), Positives = 45/79 (56%)

  Query: 1  MAVTQXXXXXXXXXXXXXXXXXXPSEITPEKSFVDDLDIDSLSMVEIAVQTEDKYGVKIP 60
            MA TQ                    ++  +KSF DDLD+DSLSMVE+ V  E+++ VKIP
  Sbjct: 1  MAATQEEIVAGLADIVNEIAGIPVEDVQLDKSFTDDLDVDSLSMVEVVAAEERFDVKIP 60

  Query: 61 DEDLAGLRTVGDVVTYIQK 79
            DED+  L+TVGD   YI K
  Sbjct: 61 DEDVKNLKTVGDATEYILK 79

b  Score = 99.0 bits (245), Expect = 1e-20
   Identities = 45/79 (56%), Positives = 60/79 (74%)

  Query: 1  MAVTQEEIIAGIAEIIEEVTGIEPSEITPEKSFVDDLDIDSLSMVEIAVQTEDKYGVKIP 60
            MA TQEEI+AG+A+I+ E+ GI   ++  +KSF DDLD+DSLSMVE+ V  E+++ VKIP
  Sbjct: 1  MAATQEEIVAGLADIVNEIAGIPVEDVQLDKSFTDDLDVDSLSMVEVVAAEERFDVKIP 60

  Query: 61 DEDLAGLRTVGDVVTYIQK 79
            DED+  L+TVGD   YI K
  Sbjct: 61 DEDVKNLKTVGDATEYILK 79
```

What if your query is almost entirely low-complexity? If soft-masking doesn't work, you may have to perform the search without complexity filters. In this case, expect many false-positive alignments and a slow search. Setting a lower E-value to remove low-scoring alignments can help reduce the size of the output.

8.8 Mask Repeats in Genomic DNA

As mentioned in Chapter 2, genomes may be full of repetitive elements and low-complexity sequence. They can be very problematic in BLAST searches, and if not masked prior to a BLAST search, will waste computer time and inflate BLAST reports with meaningless, redundant information (Figure 8-4).

Figure 8-4. BLASTX search with (a) repeats intact and (b) repeats masked (the alignments were removed from the display)



8.9 Segment Large Genomic Sequences

Nucleotide sequences can be very, very long. For example, the shortest human chromosome, number 22, is over 47 million bp. BLAST wasn't designed for large sequences and runs poorly in such an environment. You can easily run out of memory with chromosome-sized sequences. Even if you have a computer with sufficient memory, searching large sequences is inefficient because the procedure for assessing combined statistical significance scales quadratically with the number of alignments.

The simplest way to deal with large sequences is to split them into overlapping fragments. For genomes with high gene density, each fragment should be 100 Kb or less. For the human genome and others with low gene density, fragments can be larger, but try not to exceed 1 Mb.

The following Perl script splits a FASTA file into overlapping fragments. Each sequence fragment is given a unique identifier, and the definition contains the original coordinates and complete definition.

```
#!/usr/bin/perl -w
use strict;
die "usage: $0 <fasta file> <size> <overlap>\n" unless @ARGV == 3;
my ($file, $size, $overlap) = @ARGV;

my $def = "";
my $dna = "";
my $sequence = 0;
my $fragment = 0;

open(IN, $file) or die;
while (<IN>) {
    chomp;
    if (/^>(.+)/) {
        segment(  );
        $def = $1;
        $sequence++;
        $fragment = 1;
        $dna = "";
    }
    else {
        $dna .= $_;
    }
    while (length($dna) > $size) {segment(  )}
}
segment(  );

sub segment {
    return unless $dna;
    my $output = substr($dna, 0, $size);
    if (length($output) == $size) {
        $dna = substr($dna, $size - $overlap);
    }
    else {
        $dna = "";
    }
    my $start = ($fragment -1) * ($size - $overlap) + 1;
    my $end = $start + length($output) -1;
    print ">$sequence-$fragment {$start..$end} $def\n";
    for (my $i = 0; $i < length($output); $i+= 80) {
        print substr($output, $i, 80), "\n";
    }
    $fragment++;
}
```

8.10 Be Skeptical of Hypothetical Proteins

Amino acid sequencing is more difficult than nucleic acid sequencing, and therefore, sequences of most proteins are inferred from DNA translations. Some inferences come from gene predictions and others come from transcript translations. Finding the correct structure of genes in genomic DNA is very difficult; algorithms are incomplete approximations, and people make mistakes. Some research groups are conservative and only report proteins when there is good evidence. Others submit hypothetical proteins and hope that they will be useful (and they often are). As a result, many proteins in the public database are slightly incorrect or even fictitious. Unfortunately, hypothetical gene structures aren't always clearly labeled.

The most accurate protein sequences come from translating full-length cDNAs. But determining the protein encoded by a transcript isn't as simple as it sounds. While there is usually only one long open reading frame (ORF), the longest ORF won't necessarily correspond to a real protein. Be suspicious of all short proteins. Even in a full-length cDNA with a very large ORF, determining the start of translation isn't straightforward. The first methionine in the longest ORF is usually picked as the start of translation, but as a rule of convenience, not a biological truth. Many protein sequences have erroneous N-terminal extensions.

8.11 Expect Contaminants in EST Databases

A simple view is that ESTs are sequencing reads from cDNAs, cDNAs are derived from mRNAs, and mRNAs are derived from genes. Theoretically, this is true, but in practice ESTs frequently don't correspond to genes (e.g., rather than match an exon or UTR, they overlap part of a repeat on the wrong strand within an intron). The fraction of nontranscript sequence depends on the way the library was created. Some libraries are nearly devoid of extra-genic material, while others are essentially random shotgun sequence. How can you tell the difference? It's difficult to determine directly from the EST sequences.

Before the human genome was completed, the number of genes was estimated at 100,000 to 200,000. Current estimates are 25,000 to 30,000. One of the reasons for the initial high figure was that EST clustering experiments found many clusters, and people believed each cluster was a gene. One of the best ways to sort out real transcripts from pollutants is to align ESTs back to their genome. See Section 9.1.5 for more details.

8.12 Use Caution When Searching Raw Sequencing Reads

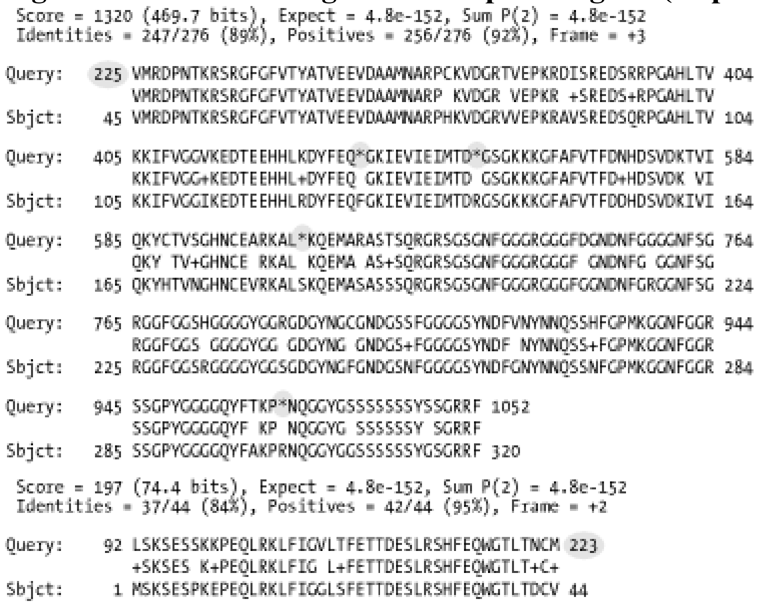
The largest source of raw sequencing reads comes from the early stages of genome projects and from EST sequencing. Most sequencing reads have an error rate of about 1 percent. This rate isn't uniform; there is a spike near the beginning and a gradual increase towards the end of the read. In addition, some regions have intrinsically high error rates due to compositional properties such as high GC content. DNA sequencing involves several steps, and there are abundant opportunities for mechanical and human error. Thus, you will need to be careful when using large word sizes. For redundant sequence collections, such as 3x shotgun coverage of a genome, large word sizes are fine, but if the absence of a single alignment is troublesome, scale down the word size to keep sequencing errors from preventing seeding.

Raw sequencing reads may be contaminated from a variety of sources. Cloning vectors are one expected source. Depending on the sequencing center, the vectors may or may not have been clipped from the sequence. Other kinds of contamination are also possible. Nuclear DNA is sometimes contaminated with mitochondrial or viral DNA, and any collection of sequence can be contaminated from another organism (genome centers usually sequence more than one entity at a time, and sometimes there's a mix up of who did what and when). ESTs sometimes have their poly-A tail intact, and whether or not this is contamination is a matter of perspective. Taken together, there are many opportunities for contamination, and it's a good idea to be cautious when using raw sequencing reads.

8.13 Look for Stop Codons and Frame-Shifts to find Pseudo-Genes

Stop codons generally aren't found in protein-coding genes. They are common, however, in pseudo-genes. It's important to recognize pseudo-genes early in a sequence-analysis pipeline because they may confound downstream analyses. Pseudo-genes usually have stop codons and insertions or deletions (Figure 8-5). Stop codons are easy to spot in alignments, but insertions and deletions must be inferred from alignment coordinates. Look for HSPs that are in different frames and appear too close to be separated by an intron (< 25 bp).

Figure 8-5. BLASTX alignment of a pseudo-gene (stop codons are circled)



```
 Score = 1320 (469.7 bits), Expect = 4.8e-152, Sum P(2) = 4.8e-152
 Identities = 247/276 (89%), Positives = 256/276 (92%), Frame = +3

Query:   225 VMRDPNTKRSRGFGFVTYATVEEVDAAMNARPCKVDGRTVEPKRDISREDSRRPGAHLTV 404
             VMRDPNTKRSRGFGFVTYATVEEVDAAMNARP KVDGR VEPKR +SREDS+RPGAHLTV
Sbjct:    45 VMRDPNTKRSRGFGFVTYATVEEVDAAMNARPHKVDGRVVEPKRAVSREDSQRPGAHLTV 104

Query:   405 KKIFVGGVKEDTEEHHLKDYFEQ*GKIEVIEIMTD*GSGKKKGFAFVTFDNHDSVDKTVI 584
             KKIFVGG+KEDTEEHHL+DYFEQ GKIEVIEIMTD GSGKKKGFAFVTFD+HDSVDK VI
Sbjct:   105 KKIFVGGIKEDTEEHHLRDYFEQFGKIEVIEIMTDRGSGKKKGFAFVTFDDHDSVDKIVI 164

Query:   585 QKYCTVSGHNCEARKAL*KQEMARASTSQRGRSGSGNFGGGRGGGFDGNDNFGGGGNFSG 764
             QKY TV+GHNCE RKAL KQEMA AS+SQRGRSGSGNFGGGRGGGF GNDNFG GGNFSG
Sbjct:   165 QKYHTVNGHNCEVRKALSKQEMASASSSQRGRSGSGNFGGGRGGGFGGNDNFGRGGNFSG 224

Query:   765 RGGFGGSHGGGGYGGRGDGYNGCGNDGSSFGGGGSYNDFVNYNNQSSHFGPMKGGNFGGR 944
             RGGFGGS GGGGYGG GDGYNG GNDGS+FGGGGSYNDF NYNNQSS+FGPMKGGNFGGR
Sbjct:   225 RGGFGGSRGGGGYGGSGDGYNGFGNDGSNFGGGGSYNDFGNYNNQSSNFGPMKGGNFGGR 284

Query:   945 SSGPYGGGGQYFTKP*NQGGYGSSSSSSSYSSGRRF 1052
             SSGPYGGGGQYF KP NQGGYG SSSSSSY SGRRF
Sbjct:   285 SSGPYGGGGQYFAKPRNQGGYGGSSSSSSYGSGRRF 320

 Score = 197 (74.4 bits), Expect = 4.8e-152, Sum P(2) = 4.8e-152
 Identities = 37/44 (84%), Positives = 42/44 (95%), Frame = +2

Query:    92 LSKSESSKKPEQLRKLFIGVLTFETTDESLRSHFEQWGTLTNCM 223
             +SKSES K+PEQLRKLFIG L+FETTDESLRSHFEQWGTLT+C+
Sbjct:     1 MSKSESPKEPEQLRKLFIGGLSFETTDESLRSHFEQWGTLTDCV 44
```

8.14 Consider Using Ungapped Alignment for BLASTX, TBLASTN, and TBLASTX

The first versions of BLAST produced strictly ungapped alignments but were still very useful. Although gapped alignment has some advantages, it may produce surprising results. When running the translating BLAST programs (BLASTX, TBLASTN, and TBLASTX), you generally look for protein coding regions and therefore don't expect to see stop codons. Stop codons are very frequent in alignments from these programs, and it isn't possible to eliminate stop codons by simply making their scores highly negative. In standard alignment algorithms (see Chapter 3), no match score can be more negative than the cost of two gaps. In Figure 8-6, all stop codon scores are given a value of -999 (for more details, see Chapter 10). Notice how two alternating gaps skip over the stops in this TBLASTX alignment between two noncoding sequences (this is a WU-BLAST alignment; NCBI-BLAST is always ungapped for TBLASTX). You can avoid stop codons only by using ungapped alignment in addition to highly negative stop scores. Doing so segments the alignment in Figure 8-6 into three short alignments with insignificant E-values.

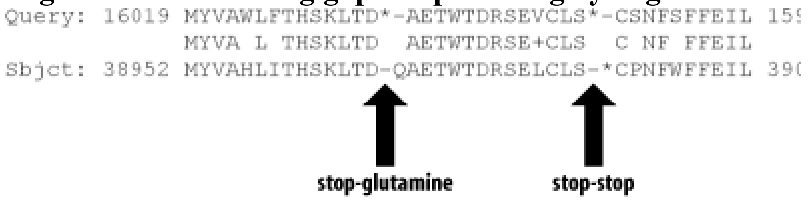


Figure 8-6. Alternating gaps skip over highly negative scores

```
Query: 16019 MYVAWLFTHSKLTD*-AETWTDRSEVCLS*-CSNFSFFEIL 15
             MYVA L THSKLTD  AETWTDRSE+CLS  C NF FFEIL
Sbjct: 38952 MYVAHLITHSKLTD-QAETWTDRSELCLS-*CPNFWFFEIL 39
```

Figure 8-7 demonstrates another feature of gapped alignment: alignments may extend far beyond the end of an exon because gapped extension is generally less specific. This is especially annoying in genomes with short introns in which gapped alignments can extend between nonadjacent exons and obscure intervening introns and exons. To reduce these lengthy extensions, decrease *X*, increase the gap extension cost, select a more stringent scoring matrix, or use ungapped alignment.

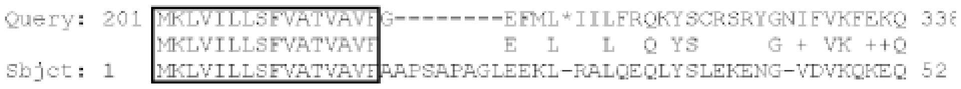


Figure 8-7. Extension is sometimes excessive: the real exon region is boxed in this BLASTX alignment

```
Query: 201 MKLVILLSFVATVAVFG---------EFML*IILFRQKYSCRSRYGNIFVKFEKQ 33
           MKLVILLSFVATVAVF          E  L   L  Q YS      G + VK ++Q
Sbjct: 1   MKLVILLSFVATVAVFAAPSAPAGLEEKL-RALQEQLYSLEKENG-VDVKQKEQ 52
```

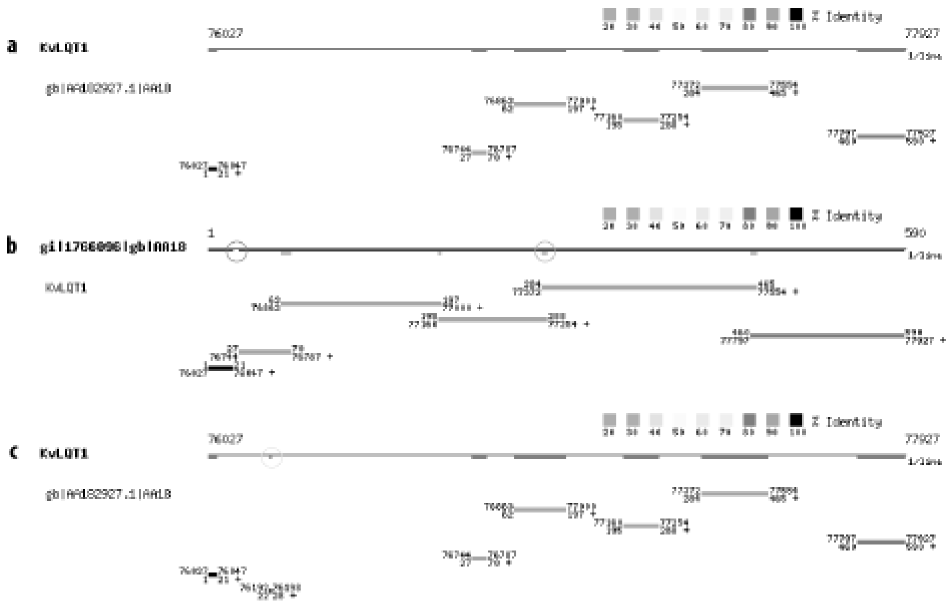
8.15 Look for Gaps in Coverage as a Sign of Missed Exons

The seeding parameters and alignment thresholds may prevent short or highly divergent exons from appearing in BLAST reports. Figure 8-8a shows an alignment between a genomic query and an EST. Most alignments overlap by a few bp, except for the 2 at the 5′ end (left side). Gaps and overlaps in coverage are easier to see by using the reciprocal search shown in Figure 8-8b. To find the missing 7-bp exon in Figure 8-8c, use *bl2seq* (see Chapter 13) with the following command line:

```
bl2seq -i est -I 21,29 -j genomic -J 76047,76744 -pblastn -W 7
```

The -I and -J parameters let you select a specific region of each sequence. What you've done is a BLASTN search between the missing part of the EST and the region between the alignments.

Figure 8-8. Finding missed exons: (a) an alignment between a genomic query and EST, (b) the reciprocal alignment showing a gap (d) and overlap (e) in coverage, (c) the tiny missed exon can be found (f) by changing the word size to 7



8.16 Parse BLAST Reports with Bioperl

The traditional BLAST output format is meant to be human readable, but when your BLAST report is 1,000 pages long, it isn't much fun to read. Sometimes all you want is the names of all sequences that have alignments above 90 percent identity. Such tasks require a BLAST parser that lets you select only the information you want. Many freely available BLAST parsers can be downloaded from the Internet, but the ones in most common use come from the Bioperl project. Bioperl is an open-source community of bioinformatics professionals that develops and maintains code libraries and applications written in the Perl programming language. If your daily routine finds you running BLAST or other sequence analysis applications, learning to use the Bioperl system can save you many hours of work and frustration.

Let's see how Bioperl can help solve the problem posed earlier: to report the names of all sequences that are more than 90 percent identical to your query.

```
#!/usr/bin/perl -w
use strict;
use Bio::SearchIO;

my $blast = new Bio::SearchIO(
    -format => 'blast',
    -file   => $ARGV[0]);

my %Name;
my $result = $blast->next_result;
while(my $sbjct = $result->next_hit) {
    while(my $hsp = $sbjct->next_hsp) {
        $Name{$sbjct->name} = 1 if $hsp->frac_identical >= 0.9;
    }
}

print join("\n", sort keys %Name), "\n";
```

Pretty simple, huh? With BLAST and Bioperl, it's possible to create all kinds of useful applications.

8.17 Perform Pilot Experiments

Before embarking on a large BLAST experiment, first try some pilot experiments. For example, if you want to compare all human proteins to all nonhuman proteins, try 100 proteins first. Or, if you want to annotate a 5 mb chromosomal region with BLASTX similarities, search 100 Kb first. If you're unsure of which parameters to use, try several and see which ones give you the kinds of results you're looking for. It may seem like a waste of time, but performing pilot experiments will actually save you time in the end.

PREVIOUS NEXT

8.18 Examine Statistical Outliers

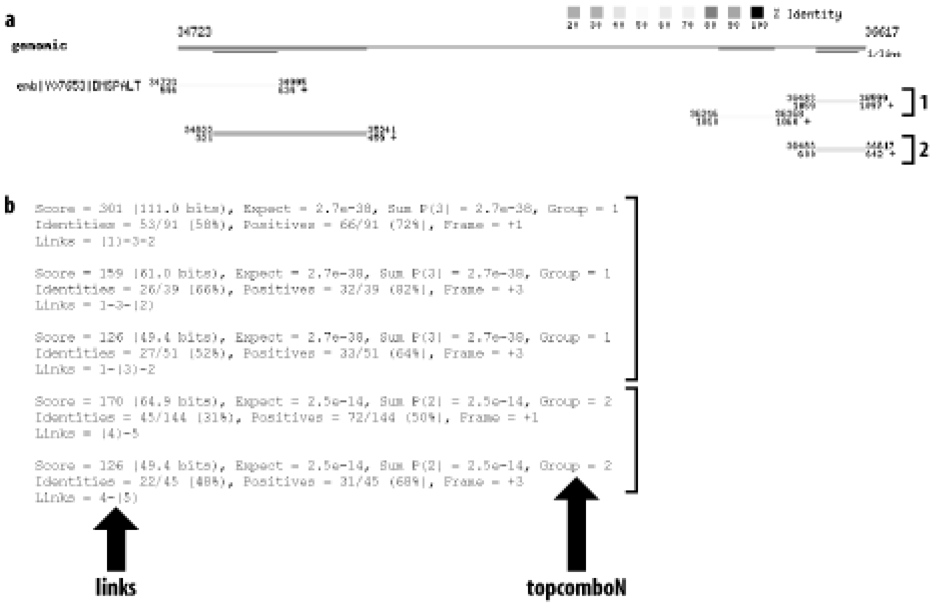
In a high-throughput setting, BLAST reports may be huge and number in the thousands. There's no way you can look at all of them, but for quality control, you should examine some of them. Keep global statistics on BLAST reports, such as number of hits per Kb. Statistical outliers may point to general problems that become more apparent in certain sequences.

PREVIOUS NEXT

8.19 Use links and topcomboN to Make Sense of Alignment Groups

WU-BLAST has two very useful parameters for displaying alignment groupings. topcomboN sorts alignments into groups and labels them. The links parameter shows the order of alignments in a group, which is much like the order of a gene's exons. Figure 8-9 displays these features.

Figure 8-9. WU-BLAST topcomboN and links (the top-to-bottom order of alignments in the graphic (a) are the same as the statistics lines from the BLASTX report (b))



```
Score = 301 (111.0 bits), Expect = 2.7e-38, Sum P(3) = 2.7e-38, Group = 1
Identities = 53/91 (58%), Positives = 66/91 (72%), Frame = +1
Links = (1)-3-2

Score = 159 (61.0 bits), Expect = 2.7e-38, Sum P(3) = 2.7e-38, Group = 1
Identities = 26/39 (66%), Positives = 32/39 (82%), Frame = +3
Links = 1-3-(2)

Score = 126 (49.4 bits), Expect = 2.7e-38, Sum P(3) = 2.7e-38, Group = 1
Identities = 27/51 (52%), Positives = 33/51 (64%), Frame = +3
Links = 1-(3)-2

Score = 170 (64.9 bits), Expect = 2.5e-14, Sum P(2) = 2.5e-14, Group = 2
Identities = 45/144 (31%), Positives = 72/144 (50%), Frame = +1
Links = (4)-5

Score = 126 (49.4 bits), Expect = 2.5e-14, Sum P(2) = 2.5e-14, Group = 2
Identities = 22/45 (48%), Positives = 31/45 (68%), Frame = +3
Links = 4-(5)
```

8.20 How to Lie with BLAST Statistics

Several techniques can help you massage BLAST statistics to either hide significant alignments or make meaningless alignments appear highly significant. Why would you want to do this? If you have to ask, you're not the intended audience. Dishonest evil doers read on.

The easiest method to adjust the significance of all scores is to set the effective size of the search space either higher or lower. Command-line parameters in both NCBI-BLAST (-Y) and WU-BLAST (Y and Z) are available. You can also alter the scoring scheme by editing the scoring matrices. A more involved approach involves hacking the source code to set your own values for λ, k, and H. WU-BLAST makes it all too easy because you can alter scores or set Karlin-Altschul parameters on the command line. Whatever approach you take, you will, of course, want to edit the footer to cover your tracks. The easiest way to do this is to run the search twice and *diff* the footers to determine what needs fixing.

With low gap penalties, you can make alignments between just about anything. For BLASTN, NCBI-BLAST always uses ungapped statistics, so you don't have to do much work to lie. Just hope that nobody notices all the gaps. This works best if you have a supervisor who is either too busy to look at alignments or wouldn't know a decent alignment if it bit him. NCBI-BLAST is very restrictive about what gap penalties you can employ for the protein-based BLAST programs. Your only choice here is to hack and recompile. WU-BLAST is very easy; set your gap costs low and include warnings on the command line to suppress messages about ungapped statistics.

Another way to trick the unobservant is to remove complexity filters. This works especially well when claiming that some anonymous low-complexity region or transcript is a cool gene. You can almost always find a small ORF that has a poor match to something with an interesting definition line. A poor match is only poor if you don't know how to fix the statistics. This approach even works when fooling scientific journals. (It really does. We've seen it happen.)

Chapter 9. BLAST Protocols

This chapter contains protocols for the most common BLAST searches. Because every BLAST experiment is unique, you should treat the protocols as a starting point and use your own knowledge about BLAST to modify the procedures. The discussions include what to do, as well as why. Although this approach makes the descriptions more verbose, explaining the logic behind these choices will help you make intelligent choices when creating your own protocols.

Most BLAST experiments fall into one of two categories: mapping and exploring. Mapping is the process of finding the position of one sequence within another—for example, finding a gene within a genome. When mapping, you can expect the alignments to be nearly identical, and the coordinates are generally the focus of the results. When exploring, the goal is usually to find functionally related sequences. When exploring, your alignment statistics (score, expectation, percent identity, etc.) are often of greatest importance, at least initially. Making functional and phylogenetic inferences, especially between distantly related sequences, often requires inspecting the alignments from a biological rather than a statistical perspective. There is, of course, a continuum between mapping and exploring, but keeping this dichotomy in mind can help you zero in on the fundamental aspects of a search strategy.

The notation used here and in the reference chapters in Section 5 is the command-line interface. If you're unfamiliar with shells and terminals, the command line is where you type in program names and options. It may seem a little odd at first, but it is analogous to filling in a web form and then clicking the submit button. While most people use BLAST via some web interface, not all pages look the same or support the same parameters. Behind the scenes, though, they all interact with a command-line version of BLAST. BLAST pages frequently let you set advanced options; usually it's a text box or boxes in which you can enter the command-line options.

9.1 BLASTN Protocols

As we said earlier, most searches can be categorized as either mapping or exploring searches. When sequences are expected to be nearly identical, you should use the +1/-3 match-mismatch parameters, which have a target frequency of 99 percent identity. Cross-species exploration requires a change in the scoring parameters and word size. We like +1/-1 for both its simplicity and its 75 percent identity target frequency. The choice of word size depends on balancing sensitivity and specificity. The default word size of 11 is too risky; use 9, which corresponds to a little more stringency than three identical amino acids because there's no allowance for degenerate codons. The choice of gap costs depends on the size of the expected gap. For simulating sequencing errors, the gap costs should be uniform and relatively high, but for modeling amino acid gaps or nucleotide hybridization bubbles, the cost of extension should be lower.

9.1.1 Mapping Oligos to a Genome

Many kinds of experiments, both molecular and computational, employ short nucleotide sequences called oligonucleotides, or just oligos (oligo is Greek for few). For example, the polymerase chain reaction (PCR) is a routine laboratory procedure for amplifying a specific nucleotide sequence from DNA or indirectly from RNA. In PCR, oligos are used as templates for DNA replication, and the subsequence between the oligos is amplified. The most important feature of oligos is they may be short enough to give rise to many false-positive matches. In a test tube, we would say the oligo hybridizes nonspecifically, and in a BLAST experiment, we would say the alignments have high expectations.

9.1.1.1 Approach

Our goal here is to simulate the interaction between an oligo and a genome in a test tube. The thermodynamics of annealing are complex and depending on the conditions of the experiment (temperature, salt concentration, length, and composition of oligo), some mismatches between the sequences and even gaps may be possible. Still, the sequences are expected to be nearly identical, so we use corresponding match-mismatch parameters. The default word size is fine here; we don't increase it because a fortuitous mismatch can prevent seeding for a short oligo. Complexity filtering is turned off because we want the entire oligo to match, and low complexity isn't expected to be a problem with such a short query sequence. Because there is quite a bit of variation from one oligo to the next, we can't set a specific *E* value. Instead, we use the default and visually inspect the report after the search.

9.1.1.2 NCBI-BLAST parameters

```
 blastall -p blastn -d <genome> -i <oligo> -G 2 -E 1 -F F
megablast -d <genome> -i <oligo> -W 11 -F F -D 2
```

9.1.1.3 WU-BLAST parameters

```
 blastn <genome> <oligo> M=1 N=-3 Q=3 R=1
```

9.1.1.4 Expected results

There may be several alignments between the oligo and genome, and not all of them may align end to end. If you are simulating PCR, mismatches at the 3′ end of the oligo are of particular interest because they may prevent priming.

If you don't find any hits, the oligo may be too short for its alignments to achieve statistical significance. For short oligos, even a 100 percent matching alignment may have a score that is expected at random in a large search space. Try raising *E*. Also, make sure that the scoring scheme favors near identity. Otherwise, lambda may transform the score to a very low amount of information, and you may not be able to set *E* high enough to recover the alignment. Other possibilities include too large a value for *W* or the use of complexity filters.

If you find too many hits, increase the stringency of the search by decreasing *E*. The suggested scoring scheme is already pretty strict, but you may want to set the gap penalties higher or turn off gapping entirely if you find too many gapped alignments. It may be that the query is just found in many places. If you don't care about the details of the alignments, tabular format is convenient to parse and takes up much less space. See Appendix A and Appendix E to learn how to report in tabular format.

9.2 BLASTP Protocols

Most BLASTP searches fall under the exploring category, which means you're trying to learn about your query sequence by comparing it to other proteins. You might also want to determine if particular regions are highly or not so highly conserved. Or you may want to gather proteins to build a phylogenetic tree. In any case, your main concern is how deeply you want to explore. The following protocols offer three levels of sensitivity.

9.2.1 The Standard BLASTP Search

Probably the most common BLAST search is BLASTP with default parameters. It is used in various settings because it balances speed and sensitivity. For example, if you want to compare all the proteins between two organisms, this is a good place to start. If the proteomes are very distant, the default parameters may not be ideal because alignments containing less than 35 percent identity aren't as easily detected. If the proteomes are very close, the standard search is still a good strategy because not all proteins evolve at the same rate, and some may diverge rather quickly.

9.2.1.1 Approach

We'll make only one adjustment to the default NCBI parameters. We use soft masking instead of normal complexity filtering so the entire alignment is scored. The WU-BLAST parameters are approximately the same as those of NCBI-BLAST.

9.2.1.2 NCBI-BLAST parameters

```
blastall -p blastp -d <db> -i <query> -F "m S"
```

9.2.1.3 WU-BLAST parameters

```
blastp <db> <query> hitdist=40 wordmask=seg postsw
```

9.2.1.4 Expected results

If you don't find any database hits, your query sequence may correspond to a novel protein. On the other hand, it may be that the parameters of the search are obscuring the similarity. If your query is very short, it may be difficult for it to achieve statistical significance. In this case, first try raising *E*. However, this step alone may not be enough, and you may have to change to a scoring matrix with a higher value of *H* (bits per aligned letter), such as BLOSUM80.

If you want to find remote homologies with short query sequences, be prepared for many false-positive alignments. If your sequence has a long, low-complexity region, be sure to have soft masking turned on. It's difficult to find collagens, for example, if complexity filters are destroying most of the alignment. Finally, try the slower, more sensitive search described later.

If you find that you have hundreds of database hits, you may be overrunning the output reporting parameters (-b and -v in NCBI-BLAST and V and B in WU-BLAST). If this is a concern, simply increase these values. However, if you're interested in only the top hits, you can either set *E* higher or use a search strategy designed for more similar sequences (below).

9.2.1.5 Optimizations and variations

The two protocols below offer speed-sensitivity tradeoffs. For more subtle changes, try altering *T*. If you use WU-BLAST, set W=4 and scale up *T* appropriately.

9.2.2 Fast, Insensitive Search

Increased speed is one reason to use an insensitive search. This is particularly true when performing multiple searches. Another reason is to increase the information content in the alignments, which is helpful for short query sequences whose alignments might otherwise fall below the significance threshold. As a rule, the insensitive search shouldn't be used for sequences that are expected to have less than 50 percent identity.

9.3 BLASTX Protocols

BLASTX is generally used to find protein coding genes in genomic DNA or to identify proteins encoded by transcripts. BLASTX runs relatively slowly, and can be the bottleneck in a sequence annotation pipeline. Most BLASTX searches are of the exploring variety. However, it is sometimes necessary to identify nearly identical sequences quickly. The last protocol gives some advice.

9.3.1 Gene Finding in Genomic DNA

Most proteins are related to other proteins. This makes BLASTX a very powerful gene-finding tool. As protein databases become larger and more diverse, BLASTX becomes even more useful because it can identify more and more genes.

9.3.1.1 Approach

As with any search involving genomic DNA, the sequence must have its repeats masked, and lowercase is preferred to Ns. If possible, low-complexity sequences shouldn't be masked with Ns to avoid terminating extension if a coding region contains a region of low complexity.

Since we want to capture a range of protein similarities, we'll use the default BLOSUM62 scoring matrix, which is a good all-around matrix. We increase the threshold score from its default of 12 to 14, which increases the speed more than twofold and is still quite sensitive.

We use a higher value for *E* because we don't want to miss low-scoring alignments that may be real genes. We set the output report options very high so that no matches are missed simply by truncation. Some protein families have many members and may fill up a BLASTX report by themselves.

For WU-BLAST, we offer two command lines. The first is similar to the NCBI parameter set, and the second uses a single large word rather than two small words. In our tests, the second set is slightly faster and more sensitive.

9.3.1.2 NCBI-BLAST parameters

```
blastall -p blastx -d <db> -i <genomic> -F "m S" -U -f 14 -b 10000 -v 10000 -e 100
```

9.3.1.3 WU-BLAST parameters

```
blastx <db> <genomic> wordmask=seg lcmask hitdist=40 T=14 B=10000 V=10000 E=100
blastx <db> <genomic> wordmask=seg lcmask W=4 T=20 B=10000 V=10000 E=100
```

9.3.1.4 Expected results

Percent identity is often a good indicator of the reliability of a protein match. Lengthy alignments above 50 percent identity don't occur by chance unless the sequence has some compositional bias. Alignments below 35 percent identity should be met with skepticism, and those below 30 percent aren't very reliable.

The ends of alignments often overrun exon boundaries. If the extensions are too long, they may force an exon into a separate alignment group. If the exon is short, it may not be able to achieve statistical significance when separated. To counteract this phenomenon, you can set the *X* parameters lower to prevent the extensions from going too far, but this can destroy weak alignments. Another option in WU-BLAST is to use olf and olmax (and their gapped counterparts golf and golmax) to change the overlap rules. These solutions aren't foolproof, so the best approach is to realize that this scenario could happen, and be on the lookout for gaps in coverage.

If your report is very long, you may have some unmasked repeats or low-complexity regions. A graphical report (Appendix D) can help determine where such repeats occur. Look for regions in which the alignment depth is very high. You may have to mask them by hand.

A less obvious problem is associated with GC-rich regions of DNA. These regions tend to have long open reading frames and can match various proteins. If the alignment threshold (Chapter 5) is low, these compositionally biased

9.4 TBLASTN Protocols

TBLASTN and BLASTX are very similar in that one sequence is protein and the other is nucleotide. But their usage is different. TBLASTN commonly maps a protein to a genome or searches EST databases for related proteins not yet in the protein databases.

9.4.1 Mapping a Protein to a Genome

Many avenues of investigation focus on a specific protein—for example, medical research on a genetic disease. For many proteins, there exist several closely related homologs, and understanding the role of a particular protein often means studying the near neighbors because they sometimes have interesting properties. The genomic environment of an encoded protein is often of great interest because the genomic sequence contains regulatory elements that determine where and when proteins are expressed. So, in addition to the typical BLASTP search for homologous proteins, it is also useful to do a TBLASTN search against your favorite genomes.

9.4.1.1 Approach

Even though this is conceptually a mapping experiment, we don't choose extremely insensitive parameters because we also want to identify closely related proteins that may be of interest. The seeding parameters, which require two matching words in a 40 aa window, capture a surprising amount of variability. We'll provide an additional WU-BLAST command line that uses a single, large neighborhood. It's both faster and more sensitive than the two-hit version but takes substantially more memory. We set *E* to a low value to cut down on the number of low scoring hits that may be prevalent in a large search space, but if your query is especially small, this value should be increased.

9.4.1.2 NCBI-BLAST parameters

```
 blastall -p tblastn -d <genome> -i <protein> -f 999 -e 1e-5
```

9.4.1.3 WU-BLAST parameters

```
 tblastn <genome> <protein> filter=seg T=999 E=1e-5
tblastn <genome> <protein> filter=seg W=5 T=25 E=1e-5
```

9.4.1.4 Expected results

If all goes well, you'll find your gene in the genome. You may also find several related proteins. If the genome is small, you may not find more than one, but if your source is larger, and more complex, you may find several copies. Genomic sequences in BLAST databases are sometimes not masked, so if your search takes a long time to complete, or if you find hundreds of similar genes, you may be hitting a repeat.

Some of the hits may be to pseudogenes. High stop codon penalties with ungapped extension will not remove all pseudogenes, so in addition to inspecting alignments for the presence of stop codons, also look for overlapping HSPs (from frame shifts) and single HSPs (when multiple exons are expected). Nearby repeats and poly-A tails in the genomic sequence are other useful indicators.

9.4.1.5 Optimizations and variations

We recommend using the serial search strategy described in Chapter 12 for all translating BLAST searches that employ long sequences. If you can't do this automatically, you can follow up each of the hits found here with *bl2seq*.

For more sensitivity, reduce the value of *T* to allow neighborhood words. For a less sensitivity and a lot more speed, W=5 T=999 is a useful WU-BLAST setting. It also has the added benefit of using much less memory than W=5 T=25. If you need to do a quick lookup and are only interested in identical matches, you can adapt the protocol found in Section 9.3.3 to TBLASTN.

One way to speed up this procedure is to start with a BLASTP search to identify similar proteins and then follow up each hit in its own genome with near-identity parameters. One disadvantage to this strategy is that you will have more searches to perform and a lot of sequence handling. The assumption that the protein database you're using for your

9.5 TBLASTX Protocols

As discussed in Chapter 2, coding sequences evolve slowly compared to surrounding DNA. This makes TBLASTX a powerful gene-prediction tool for genomes that are appropriately diverged. What is the proper evolutionary distance? Because genes and organisms change at varying rates, there is no simple answer like "100 million years." If the distance is too great, the similarities may no longer be visible, but if the distance is too small, sequence similarity loses discriminatory power. For example, there is little sense in performing TBLASTX searches between humans and *E. coli* or humans and chimpanzees.

Historically, TBLASTX has not been as popular as the other BLAST programs for several reasons. First, TBLASTX is computationally intensive. Second, until recently, there were not many completely sequenced genomes. Third, when you get a match, you will rarely find a useful description for what was found—just an alignment between two potential coding sequences. As more genomes are sequenced and computer performance continues to rise, TBLASTX will become more useful.

9.5.1 Preventing Stop Codons

The scoring matrices distributed with the BLAST programs give positive scores for aligning stop codons to one another. This is unacceptable for discriminating between coding and noncoding regions. Chapter 10 covers installation of BLAST software and describes how to create derivative scoring matrices with highly negative stop codon scores. If you don't have permission to make these changes, you can create the derivatives in your home directory. In this case, you need to specify the explicit path to your matrix rather than use just the name. WU-BLAST operates a little differently, and it is more convenient to specify alternate scores on the command line. Each protocol gives an example of this. As discussed in Chapter 8, gapped alignment can skip over stop codons. For this reason, consider using ungapped alignment for your TBLASTX searches.

9.5.2 Finding Undocumented Genes in Genomic DNA

Gene prediction is difficult. There are no genomes for which all protein coding genes are completely known. One of the most highly investigated genomes is the human genome, but despite what you read in the news, the number of genes can't be stated with much confidence. Counting genes is easier than determining their exact structure, and as a result, there are many proteins for which the true sequence is in doubt. Many genes are still waiting to be discovered (and many documented genes aren't real genes).

9.5.2.1 Approach

TBLASTX is computationally expensive because it translates both strands of the query and database sequences in three frames on each strand. To make matters worse, the sequences and databases searched by TBLASTX tend to be large. To counteract these factors we choose insensitive seeding parameters, which is appropriate, considering that the extension algorithm is gapless and therefore also less sensitive.

Like any other search employing genomic DNA, it is always a good idea to mask repeats first. Here we prefer hard masking instead of soft-masking and normal complexity filtering. Our reasoning is that low-complexity sequence is common in genomic DNA and random word hits near low-complexity sequence may result in lengthy extensions, high alignment scores, and misleading statistical significance.

For WU-BLAST, we offer two command lines. The second, which uses a single, large word rather than two small words, is faster and more sensitive, but requires more memory. It also shows how to change scoring matrix values from the command line with the altscore parameter.

9.5.2.2 NCBI-BLAST

```
 blastall -p tblastx -d <db> -i <genomic> -f 999
```

9.5.2.3 WU-BLAST

```
 tblastx <db> <genomic> filter=seg W=3 T=999 hitdist=40 nogap
tblastx <db> <genomic> filter=seg W=5 T=25 nogap altscore="* any -999" altscore="any *
```

Part IV: Industrial-Strength BLAST

Chapter 10. Installation and Command-Line Tutorial

This chapter shows you how to install NCBI-BLAST and WU-BLAST software on your own computer. This is necessary if you want to use BLAST in a high-throughput setting or develop specialized applications. After the installation section, this chapter presents a command-line tutorial that also serves as a test suite to make sure your BLAST installation behaves as expected.

10.1 NCBI-BLAST Installation

NCBI-BLAST, as the name implies, is available from the National Center for Biotechnology Information (NCBI). Precompiled binaries and source code are available for free and without restriction. The source code is in the public domain, so there are quite a few derivative works, both commercial and free (see Chapter 12). NCBI-BLAST is currently available as precompiled binaries for 11 popular operating system-hardware combinations. In addition, there is this very generous statement in the *README.bls* file:

BLAST binaries are provided for IRIX6.2, Solaris2.6 (Sparc) Solaris2.7 (Intel), DEC OSF1 (ver. 5.1), LINUX/Intel, HPUX, AIX, BSD Unix, Darwin, MacIntosh, and Win32 systems. We will attempt to produce binaries for other platforms upon request.

If you have a platform that isn't supported as a precompiled binary, you may wish to take up the offer from the NCBI, or you may be able to find one using an Internet search engine such as Google. You can also compile the executables yourself; the source code may be obtained as part of the NCBI toolbox: ftp://ftp.ncbi.nih.gov/toolbox/ncbi_tools. For more information about the toolbox, see http://www.ncbi.nlm.nih.gov/IEB/ToolBox.

This chapter will take you through the installation procedures for Unix, Windows, and Macintosh. It doesn't cover how to build the NCBI executables from source. If you are a Windows or Macintosh user, please read the Unix installation first because it has some information that isn't duplicated in the other sections.

10.1.1 Unix Installation

The first step is to download a compressed Unix tape archive, often called a tarball, to your computer. Find the appropriate executable for your system at ftp://ftp.ncbi.nih.gov/blast/executables. A note of caution here: the files in the tarball aren't contained in a subdirectory so it is a good idea to place the tarball in its own directory before you expand the archive. If you're downloading via a browser, you may have plug-ins that automatically expand the archive. This could leave you with a bunch of files all over your system, or it may create a directory for you. To be safe, if you're using a browser, download the tarball to a new directory, for example, */usr/local/pkg/ncbi-blast*, or perhaps *ncbi-blast* in your home directory if you don't have root access.

If the archive hasn't already been expanded, you can expand it with this command, where *your_platform_name* will be something like *linux.tar.Z* or *linux.tar.gz*:

```
tar -xzf blast.your_platform_name
```

Not all versions of *tar* support the *-z* option above, in which case you can use the following command line:

```
zcat blast.your_blastform_name | tar -xf -
```

10.1.1.1 Files and directories

More than 20 files come with the installation. Table 10-1 shows the files and a very brief description in logical order. See the NCBI-BLAST reference in Chapter 13 for comprehensive coverage of each program.

Table 10-1. NCBI-BLAST installation files

File	Description
blastall	The main blast executable. This program runs the five most common BLAST programs: *blastn*, *blastp*, *blastx*, *tblastn*, and *tblastx*.
blastpgp	The executable for running PSI-BLAST and PHI-BLAST searches.

10.2 WU-BLAST Installation

Obtaining WU-BLAST software is slightly more complicated than NCBI-BLAST because it requires a license from Washington University in St. Louis. If you are affiliated with an academic institution or a nonprofit organization, the license is free. If you are part of a for-profit enterprise, you must pay a licensing fee. The price is expensive by shrink-wrapped software standards, but is similar to other bioinformatics software packages available from universities. If you find the cost prohibitive, an earlier version of WU-BLAST is available for free. The free version contains fewer features, and is available for a limited number of operating systems, but for most people, it works just fine. If your operating system isn't supported and your specific use doesn't require gapped alignment, a free version of the classic, ungapped BLAST with public domain source code also exists. This older version, 1.4.9, is nearly identical to NCBI-BLAST Version 1.4, which is no longer available from the NCBI.

Should you wish to license WU-BLAST or download the free versions, visit the official site for the WU-BLAST software at http://blast.wustl.edu. The free versions can be downloaded with a couple clicks, but more patience is required for the licensed version. After the license is issued, you will be sent a user-specific URL from which to download the software. It's a good idea to save this information because you will use it again to download the free updates. Licensed users are notified by email as new features are added (usually a few times per year).

WU-BLAST is available only for Unix operating systems. If you don't have access to a Unix computer, you can run Linux or FreeBSD under a virtual machine with products such as VMWare (http://www.vmware.com) or VirtualPC (http://www.connectix.com).

10.2.1 Expanding the tarball

The software comes as a compressed Unix archive, or *tarball*. First, create a directory such as */usr/local/pkg/wu-blast*; if you don't have root access, create a *wu-blast* directory inside your home directory. Next, download the tarball to that directory. If you do this from a browser, the files may be extracted automatically. If not, use the following command, where *your_platform_name* will be something like *linuxi686.tar.Z*:

```
tar -xzf blast2.your_platform_name
```

Not all versions of *tar* support the *-z* option above, in which case you can use the following command line:

```
zcat blast.your_blastform_name | tar -xf -
```

Before you continue with the rest of the installation procedures, look at what's inside the tarball.

10.2.2 Files and Directories

There are a number of files and two subdirectories. The most important items are described very briefly in Table 10-2 in logical, rather than alphabetical, order. See the WU-BLAST reference in Chapter 14 for more information.

Table 10-2. WU-BLAST files and directories

Name	**Description**
blasta	The WU-BLAST executable. Unlike the free version, which comes with five different BLAST executables, the licensed version has only one.
blastn, *blastp*, *blastx*, *tblastn*, *tblastx*	Symbolic links (aliases) to *blasta*. *blasta* figures out what kind of program to run based on the name of the symbolic link.
xdformat	Executable for formatting both nucleotide and protein databases.

10.3 Command-Line Tutorial

Now that you've installed the NCBI-BLAST and/or WU-BLAST software, it's time to try it out. To do this, you will need sequences for both queries and databases. It's generally a good idea to start with something small and make sure it works before attempting to analyze large databases. To begin, download the book's example files from http://examples.oreilly.com/BLAST. Copy the *Sequence* directory to a local hard disk. This directory contains several database and sequences in FASTA format. The six files you will need for testing are described in Table 10-3.

Table 10-3. Contents of the Sequence directory

Name	**Description**
ESTs	2000 nucleotide sequences from the nematode Caenorhabditis elegans. These sequences originated from http://www.wormbase.org.
EST	A single EST from the previous collection.
globins	1203 protein sequences corresponding to the globin family (Pfam Version 7.6). These sequences and other protein families are available from http://www.sanger.ac.uk/Software/Pfam.
globin	A single protein from the previous collection.
AF287139	Latimeria chalumnae (Coelacanth) *Hoxa-11* nucleotide sequence.
AAG39070	Latimeria chalumnae (Coelacanth) *HoxA-11* protein sequence.
fugu_genomic	*Takifugu rubripes* (Pufferfish) genomic sequence containing globin genes.
chicken_genomic	*Gallus gallus* (Chicken) genomic sequence containing globin genes.
HoxDB	Nine *homeobox* protein sequences from different organisms.
p53	The *Drosophila melanogaster* p53 protein sequence.
p53DB	Twelve p53 homologous protein sequences from different organisms.
hit_file.p53	p53 pattern file for PHI-BLAST search.

10.3.1 NCBI-BLAST

10.4 Editing Scoring Matrices

The amino scoring matrix files distributed with NCBI-BLAST and WU-BLAST assign a score of +1 to paired stop codons. This doesn't make much biological sense and reduces the ability of TBLASTX to discriminate between coding and noncoding similarities. Therefore, you should edit the scoring matrices to change stop codon pairs to a highly negative score. Be sure to edit the original matrices. The NCBI-BLAST scoring matrices are in the *data* directory. For WU-BLAST, they are in the *matrix/aa* directory. The final line of the scoring matrix files looks like this:

```
* -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4  1
```

Just change the final number to -999:

```
 * -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -999
```

You have to do this only once if you remember to keep your edited matrices when updating your BLAST installation.

For any of the translating BLAST programs, you can also change all stop scores to highly negative values. If used in conjunction with ungapped extension, doing so prevents a lot of noncoding sequences from appearing in significant alignments. The following Perl script modifies the standard matrices:

```
#!/usr/bin/perl
while (<>) {
    if (/^#|^\s/) {
        print;
    }
    elsif (/^\*/) {
        print '*', ' -999' x 24, "\n";
    }
    else {
        s/\S+\s*$/\-999\n/;
        print;
    }
}
```

Both NCBI-BLAST and WU-BLAST require matrices to have specific names. Unrecognized names cause NCBI-BLAST to terminate the search. WU-BLAST continues searching, but it employs ungapped values for λ, k, and H (it issues a warning to this effect). Try to maintain the names of the matrices, but in a location with an obvious name such as *stop-999-matrices*. Both versions of BLAST look for scoring matrices in the local directory, and on Unix systems, they recognize the BLASTMAT environment variable. Therefore, prior to the search, you can either create a symbolic link (alias) to the scoring matrix of choice or set the BLASTMAT environment variable to point to the location of specialized matrices. In the following examples, the derivative matrices are located in */my_computer/stop-999-matrices*:

```
ln -s /my_computer/stop-999-matrices/BLOSUM62 .
blastall -p tblastx -d db -i query
rm BLOSUM62

setenv BLASTMAT /my_computer/stop-999-matrices
blastall -p tblastx -d db -i query
unsetenv BLASTMAT
```

WU-BLAST users can use the altscore parameter to change the scores of any pair of letters rather than edit the matrix files. See Chapter 14 for more information on the altscore parameter.

Chapter 11. BLAST Databases

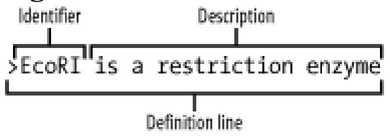
This chapter shows how to create and maintain BLAST databases—one of the most neglected yet important aspects of using BLAST. We begin with a discussion of the proper use of the FASTA format, and then turns to BLAST database issues. We finish with a general exploration of sequence databases as well as the International Nucleotide Sequence Database.

11.1 FASTA Files

Regardless of where you get your sequences, you will eventually want them in FASTA format because it is the standard currency for sequence data. The FASTA format has a very simple specification consisting of two parts: the definition line and the sequence lines.

The *definition line* is a single line that begins with the mandatory > symbol immediately followed by an identifier and then a description. There are no spaces between the > and the identifier. The identifier itself must not contain any whitespace because it is the delimiter between the identifier and the description. The description is free-form text that may contain any characters except an end-of-line character. Figure 11-1 shows a simple definition line in which the identifier is "EcoRI" and the description reads "is a restriction enzyme."

Figure 11-1. The FASTA definition line



The sequence lines follow a very simple format: they may be any length and there may be any number of them. Usually, you'll see 50, 60, or 80 characters per line, but the choice is arbitrary. Some software relies on sequence lines not being too long, so it's generally a good idea to follow the convention of 50 to 80 characters per line.

The first and most important guideline for FASTA definition lines is that the identifier uniquely specifies the sequence in some database. The identifier and description are actually optional. The following is a valid, though confusing, FASTA file because there is no identifier (dumb is the description).

```
> dumb
GAATTC
```

The following definition lines are confusing because the identifier isn't unique:

```
>chromosome 1 sequence 1
>chromosome 1 sequence 2
```

This is easily remedied by replacing the whitespace with some other character:

```
>chromosome_1-sequence.1
>chromosome_1-sequence.2
```

On the surface, these look like good identifiers, but another researcher may have the same identifiers for completely different sequences from another organism. How can you prevent this? You can't, but you can minimize potential conflicts by including a unique tag, based on your name or institution. If your data will be made public, the best solution is to submit your sequences to the public databases and use the accession numbers they provide. If not, choose identifiers you think will be unique (and make sure you read about creating fake GI numbers in Section 11.2.3).

The sequencing world is usually very cooperative, and standards have been developed to minimize name conflicts. In particular, there is a tight collaboration between DDBJ, EMBL, and GenBank so that accession numbers among these databases are guaranteed to be unique. But this isn't true of all databases. Although the identifier "AAG39070" points to a specific DDBJ/EMBL/GenBank record, it may also point to a wholly different sequence in another database. A good way to avoid name conflicts is to make sure the identifier specifies a database in addition to some unique tag for the sequence. Let's look at how the NCBI solves this problem.

11.1.1 NCBI Identifier Format

The NCBI identifier format[1] indicates the name of the database in addition to an accession number. These tokens are separated by the "|" symbol, often called a bar or a pipe. This symbol can be confusing in some fonts, as it may look like a lowercase L or the number one. Try using a constant-width serif font such as Courier if you're having trouble seeing them.

11.2 BLAST Databases

The mechanics of creating BLAST databases is quite simple; just run *formatdb* or *xdformat* with the proper syntax. Chapter 10 discussed this topic, and you'll find the command summaries in Chapter 13 and Chapter 14. There are, however, subtleties that make this process more complicated than it may appear.

11.2.1 Large Databases

One of the most common database complications occurs with large files. Most computers today use 32-bit operating systems and 32-bit filesystems. This puts a physical limit of 4 GB on the amount of RAM and 4 GB on the size of any particular file. (You may find that you are actually limited to less than 4 GB in both cases, and a 2-GB limit is quite common.) Most computers these days don't have or need 4-GB RAM. However, most hard disks are quite a bit larger than 4 GB, and files can sometimes exceed these limits. Therefore many operating systems have the option of using 64-bit filesystems. Unfortunately you can't just change the filesystem and expect everything to work. Making software applications aware of large files often means recompiling them with special flags, and the process of migrating to a 64-bit filesystem can be painful because the applications don't tell you useful things like "I'm not large-file-aware." Instead, they just sit there quietly burning CPU time while they run in endless loops.

11.2.1.1 Large NCBI databases

The standard protocol for formatting a database is to run *formatdb* on a FASTA database:

```
formatdb -p F -i fasta_db -o
```

NCBI-BLAST databases are physically limited to 4 GB of sequence, which corresponds to about 4 billion amino acids or 16 billion nucleotides (nucleotides are compressed 4:1). On a 32-bit filesystem, the previous approach won't let you use all this space because the FASTA file can't contain more than 2 or 4 billion letters. Creating a database larger than 2 or 4 billion letters requires piping sequence to *formatdb*.

```
cat fasta1 fasta2 fasta3 | formatdb -p F -i stdin -n my_db -o
```

But what if you happen to have more than 16 billion letters? This isn't a problem because *formatdb* automatically segments individual BLAST databases to files containing 16 billion nucleotides and creates something called an alias database that stitches them all together. This is really convenient because it means that you can search enormous databases even on 32-bit filesystems. Alias databases are discussed in more detail later in this chapter.

It's still possible to run into file size issues by piping FASTA files to *formatdb* because the filesystem maximum may be 2 GB and the implicit BLAST maximum is 4 GB. Fortunately, *formatdb* lets you set the size of each database volume with the *-v* parameter. The following example sets this size this to 2 billion and includes a bit more realism by piping the FASTA files from a compressed format.

```
zcat file*.gz | formatdb -i stdin -p F -o -n my_db -v 2000000000
```

A word of caution: be precise. If you accidentally leave off one of the zeroes, you can create 10 times as many files. For large databases, this can be a problem because the maximum number of volumes is 100.

11.2.1.2 Large WU-BLAST databases

WU-BLAST doesn't use alias databases, so the only way to create a database larger than 4 GB is on a 64-bit filesystem. If you're accessing or distributing databases over a network, the network must also be 64-bit aware. To index a large number of compressed files, use a command such as the following:

```
zcat file* | xdformat -n -I -o ESTs -- -
```

In the typical Unix command line syntax, the double-dash indicates the end of the command line options, and the single-dash denotes standard input rather than a file.

If you're stuck with a 32-bit filesystem and need to search large BLAST databases, you can use virtual databases, which are explained next. If you use the free version of WU-BLAST, there is no large file support and no virtual database mechanism. Your best solution is to create several databases within the limits of your filesystem, search each

11.3 Sequence Databases

The sequences in BLAST databases come from sequence databases. But what are sequence databases and where do you get them? The answers to these simple questions are surprisingly complex. Sequence databases come in many shapes and sizes. Some are just collections of raw sequence data from genome sequencing projects, while others contain comprehensive information about the origin and function of the sequences. Unfortunately, there isn't a one-stop shopping place to get all the information you may want, but there is one particular service worth mentioning above all others: the International Nucleotide Sequence Database.

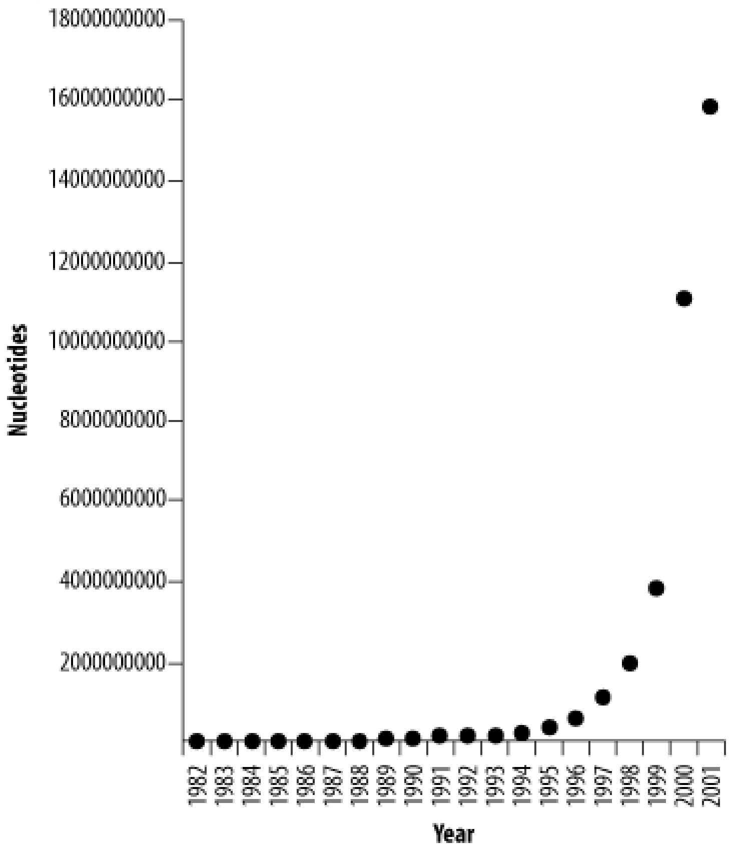
11.3.1 International Nucleotide Sequence Database

Probably the most important molecular biology resource is the public sequence database maintained by the International Nucleotide Sequence Database (INSD). It is composed of three parties: the DNA Data Bank of Japan (DDBJ, http://www.ddbj.nig.ac.jp), the European Molecular Biology Laboratory, (EMBL, http://www.embl.org), and GenBank from the National Center for Biotechnology Information (NCBI, http://ncbi.nlm.nih.gov/GenBank). This consortium collaborates to form the largest public repository for DNA and protein sequences in the world. Because it is such an important resource, this chapter spends some time exploring it.

11.3.2 Database Growth

The amount of publicly available sequence has been growing geometrically, doubling approximately every 14 months (see Figure 11-2). Fortunately, computer technology has also kept pace. While it seems scary that GenBank is currently approaching 100 GB and will be half a terabyte in a few years, it's nice to know that this isn't going to be a problem. Not every database grows so fast, though. Organism-specific databases such as the Saccharomyces Genome Database, WormBase, and FlyBase are growing at a more moderate pace, principally because the sequence of their genomes is complete. But many new genome projects are just getting started, and they will probably grow very quickly.

Figure 11-2. Growth of DDBJ/EMBL/GenBank



11.3.3 Flat Files

Sequence databases usually offer their data in several different formats. The FASTA format is universally accepted

11.4 Sequence Database Management Strategies

There are many useful public sequence databases, and you may have access to some private ones as well. Because this is a book about BLAST, we assume you want to use these collections of sequences in BLAST searches. Some sequences may be used as queries, and others in databases. How are you going to manage them all in a rational way? Several possible strategies exist, and the correct one for you depends on your needs and resources. To demonstrate some of the issues, let's review a typical sequence analysis scenario.

Suppose a colleague of yours has just found the gene that makes cats go crazy for catnip. She wants to learn more about this gene and comes to you for help because you are a BLAST expert. The first thing she wants to do is a BLAST search to find out what vertebrate proteins are similar to this one. Where are you going to get such a database of proteins? Once you perform the BLAST search, you find several interesting similarities. Your colleague tells you that these are probably all part of a family of proteins, and she would like to build a phylogenetic tree to determine their relationships to one another. How are you going to get the individual sequences? Finally, she decides she wants more information about the human sequences, and to do that, she would like references to the scientific literature like the ones she would find in a DDBJ/EMBL/GenBank report. How are you going to retrieve such information? You could just refuse to help her because these aren't really BLAST problems, but these are the kinds of tasks many BLAST users must face. Let's take a look at how they can be solved.

This example has basically two solutions to each question: the first is to use tools available on the Internet. The second is to build the tools yourself. In general, it is much easier to use the Internet, but for high speed or high-throughput operations you'll want a local solution. After you read this chapter, you may decide that you want some services to be provided locally, while others are Internet-only operations. This section begins with a brief review of databases.

11.4.1 Queries, Indexes, and Reports

The most common database operation is a query. One person may want to retrieve a particular sequence. Another may want all human sequences. As you have seen, sequence records have quite a bit of useful information, and a user may request nonsequence information such as all the MEDLINE references for all sequences with the word disease in the description.

The efficiency with which a query is executed depends a lot on how the database is indexed. If there is no indexing, a query must operate on every record of the database. So, for example, if you want to find all the coelacanth sequences, you would have to look through millions of records to find the handful whose sequences originate from the coelacanth. Clearly, this isn't going to be efficient, so databases usually have indexes that, for example, keep lists of species and all the sequences for each species.

The most straightforward kind of indexing occurs when there is a unique relationship between a property and a sequence. This is called a *one-to-one mapping*, and an example would be an accession number. A more complex indexing occurs when a property points to many sequences. This is called a *one-to-many mapping*, and an example is a species name that is shared by millions of records.

Once a query is executed, the data must be reported in some format. For sequences, this is usually the FASTA format. For other kinds of data, there are other appropriate formats, such as lists, tables, and graphs.

11.4.2 Local Database Considerations

Having a local sequence database has some real advantages. First, local databases are faster and more reliable because they don't rely on an Internet connection. If you're involved in high-throughput research, these reasons are sufficient. Another compelling reason is that you can combine several databases, and even include your own sequences that aren't in the public databases. The downside to creating a local sequence database is the amount of work it takes. Depending on the scale of the operation, it can be a full-time job. Here are six important issues to address when building a local sequence database:
Downloading

Each database you support must be downloaded from time to time to keep the data current. For example, GenBank

Chapter 12. Hardware and Software Optimizations

This chapter explores how to optimize BLAST searches for maximum throughput and will help you get the most out of your current and future hardware and software. The first rule of BLAST performance is optimize your BLAST parameters. Incorrect settings can cause BLAST to run slowly, and you can often achieve surprising increases in speed by adjusting a parameter or two. Chapter 9 can help you choose the correct parameters for a particular experiment. If you're already running BLAST efficiently and want to get the most BLAST performance possible, read on.

12.1 The Persistence of Memory

Modern operating systems cache files. You may hear it referred to as RAM cache or disk cache, but we'll just call it cache. Once a file is read from the filesystem (e.g., hard disk), the file is kept in memory even after it is no longer used, assuming there's enough free RAM to do so. Why cache files? It's frequently the case that the same file is requested repeatedly. Retrieving from memory is much faster than from a disk, so keeping it in memory can save a lot of time. Caching can be very important in sequential BLAST searches if the database is located on a slow disk or across a network. While the first search may be limited by the speed that the database can be read, subsequent searches can be much faster.

The advantage of caching is most appreciable for insensitive BLAST searches, such as BLASTN with a large word size. In more sensitive searches, retrieving sequences from the database becomes a smaller fraction of the total elapsed time. In Table 12-1, note how the speed increase from caching is a function of sensitivity (here, word size).

Table 12-1. How caching benefits insensitive searches

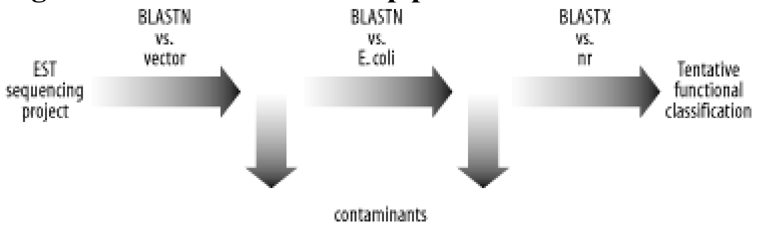
Program	Word size	Search 1	Search 2	Speed increase
BLASTN	W=12	12 sec	7 sec	1.71 x
BLASTN	W=10	33 sec	28 sec	1.18 x
BLASTN	W=8	57 sec	52 sec	1.10 x
BLASTN	W=6	243 sec	238 sec	1.02 x

BLAST itself doesn't take much memory, but having a lot of memory assists caching. Look at the amount of RAM in your current systems and the size of your BLAST databases. As a rule, your RAM should be at least 20 percent greater than the size of your largest database. If it isn't and you do a lot of insensitive searches, a simple memory upgrade may boost your throughput by 50 percent or more. However, if most of your searches are sensitive searches or involve small databases, adding RAM to all your machines may be less cost-effective than purchasing a few more servers.

12.1.1 BLAST Pipelines and Caching

If you're running BLAST as part of a sequence analysis pipeline involving several BLAST searches and multiple databases, you may want to consider how caching will affect the execution of the pipeline. For example, look at the typical BLAST-based sequence analysis pipeline for ESTs depicted in Figure 12-1. The most obvious approach is to take each EST and pass it through each step. But is this the most efficient way?

Figure 12-1. EST annotation pipeline



It's common to design sequence analysis pipelines with the following structure:

```
for each sequence to analyze {
    for each BLAST search in the pipeline {
        execute BLAST search
    }
}
```

12.2 CPUs and Computer Architecture

The clock speed of a CPU isn't necessarily an accurate indicator of how fast it will run BLAST. There are other complicating factors such as the amount of L2 cache, the memory latency and the speed of the front-side bus. Unfortunately, there is no good rule to predict how fast BLAST will perform on a particular computer except for the obvious within-family predictions—for example, that a 1-GHz Pentium III will be faster than an 800-MHz Pentium III. The best you can do is to benchmark a bunch of systems or contact people who have already done so.

Two benchmarks are provided Table 12-3. Before reading the description, please understand that you should use extreme caution whenever interpreting any benchmarks because the benchmarking protocol may be very different from your everyday tasks, and therefore may not reflect real-world performance. The best benchmark procedure should mimic your daily routine. In addition, if you use benchmarks to decide what hardware to purchase, you may be in for a nasty surprise, as other important considerations may override a simplistic interpretation of the "most BLAST for the buck." Total cost of ownership is a complicated equation that includes maintenance, support, facilities, cooling, and interfacing with legacy equipment and culture.

Chapter 12 shows the performance on various platforms when searching all members of a database against themselves. There are two databases, and both can be found at http://examples.oreilly.com/BLAST. The tests were performed using default parameters for NCBI-BLAST. The following command lines were used:

```
 time blastall -p blastn -d ESTs -i ESTs > /dev/null
time blastall -p blastp -d globins -i globins > /dev/null
```

Table 12-2. Performance benchmarks of various systems

CPU; clock speed	**blastn test**		**blastp test**	
	Time (sec)	**Giga-cycles**	**Time (sec)**	**Giga-cycles**
Macintosh G4: 550 MHz	1011	556	1599	879
Sun Ultra Sparc III; 750 MHz	835	626	1427	1070
Intel Pentium III; 1 GHz	649	649	1187	1187
Intel Pentium IV Xeon; 1.8 GHz	469	844	788	1418
AMD Athlon 1800+; 1.533 GHz	416	638	741	1136

12.2.1 Multiprocessor Computers

One way to speed up BLAST is to employ multiprocessor computers. BLAST is a multithreaded application and can utilize the additional processors. Adding additional processors to a computer is sometimes cheaper than purchasing multiple machines because you don't have to duplicate all the other components. That said, once outside the commodity computer market, prices rise steeply, and a computer with 32 CPUs is likely to cost you much more than 16 dual-CPU computers. The improvement with multiple processors isn't completely linear, and it depends on the type of search.

12.3 Compute Clusters

The price and performance of commodity computer hardware and the sophistication of modern free operating systems have made it very attractive to set up computer clusters rather than purchase a multiprocessor behemoth. Clusters don't have to be dedicated rack-mounted towers of blinking lights and buzzing drives; they can also be a mixture of desktop computers that use their idle time to run jobs. There are two fundamental kinds of clusters: Beowulf-style clusters and compute farms. *Beowulf clusters* act as a single computer, sharing memory and CPU cycles to cooperatively solve the same problem. *Compute farms* don't share memory or CPU cycles and solve separate, but possibly related problems. The field of bioinformatics has algorithms that are appropriate for both kinds of clusters, but BLAST is really a job that is best suited to compute farms. There are two major reasons for this: (1) BLAST is more data-intensive than compute-intensive, and (2) large-scale BLAST searches consist of many small jobs that are easily parallelized on separate machines.

If you wish to build your own cluster, be prepared for quite a bit of work. There are plenty of considerations outside the normal window-shopping for the best price-performance ratio. For example, one of the most common problems is having sufficient power and cooling. It doesn't do much good to have a super computer that is constantly overheating and burning out its components. Total cost of ownership is a complicated equation, and you're better off not trying to solve this entirely on your own. Your best bet is to talk with people who build clusters for a living. Several companies will sell you prepackaged compute farms for running BLAST. For those who like getting their hands dirty, the bioclusters mailing list at http://bioinformatics.org has plenty of useful information in their archives and helpful members who will gladly give advice.

12.3.1 Remote Versus Local Databases

When designing a cluster, one of the most important decisions is where to put your BLAST databases. There are two general choices: (1) store the database on a file server and let the cluster access it remotely over a network, or (2) keep a local copy of the database on each computer. Both methods have their advantages and disadvantages, so there is no simple way to determine which is better.

12.3.1.1 Remote databases

It's simpler to manage the files on one computer than on multiple computers. This is particularly true if you update your BLAST databases on a frequent, perhaps daily basis. So this is one good reason to use remote databases. If you run your compute nodes diskless, it is really the only choice. The main concerns with this approach are network bandwidth and the speed of the file server. Most computers today have 100-Mbps network interfaces. This translates to 12.5 MBps. Fast computers performing insensitive searches (e.g., BLASTN) can actually exceed this transfer rate. In this case, the compute nodes will sit idle, waiting for data. But what happens when multiple computers are all connected to the same database server? Unfortunately, they must all share the same network bandwidth from the server, so if 10 compute nodes are connected to a database server, each one may get only data at 1.25 MBps. Not good. But remember that if the compute nodes have enough RAM and the databases aren't falling out of cache, subsequent searches will be very fast because they can read the database directly from memory.

One obvious improvement is to employ faster networking. Doing so increases the cost of each compute node a little and significantly increases the cost of network switches because gigabit network switches are still quite expensive. However, it is possible to use a hybrid solution in which the database server is connected to a hybrid network switch via a gigabit line and the compute nodes are connected to the switch via the more common 100-Mb interface. This is much cheaper than using gigabit everywhere, and, because exceeding 12.5 MBps is rare, it doesn't hinder performance too much.

When building file servers, people often neglect to put in enough RAM. For BLAST database servers, though, you really want as much RAM as possible. Caching applies on the file-server end, too, and if several computers request data from the file server, it's much better if it can be served from memory rather than from disk. If you're thinking of using autonomous network attached servers as a BLAST database server, think again. Most don't have gigabit networking or enough RAM.

12.3.1.2 Local databases

12.4 Distributed Resource Management

If you're running a lot of BLAST jobs, one problem to consider is how to manage them to minimize idle time without overloading your computers. Being organized is the simplest way to schedule jobs. If you're the only user, you can use simple scripts to iterate over the various searches and keep your computer comfortably busy. The problem starts when you add multiple users. In a small group, it's possible for users to cooperate with one another without adding extra software. Sending email saying "hey, stay off blast-server5 until I say so" works surprisingly well. But if you have a large group or irresponsible users, you'll want some kind of distributed resource management (DRM) software.

There are a number of DRM software packages, both free and commercial. But even the free ones will cost you time to install and maintain, and users need training to use the system. Table 12-3 lists some of the most popular packages in the bioinformatics community. Condor is an established DRM that is downloadable for free. It is rare in that it supports Windows and Unix. LSF is a mature product with many bioinformatics users. It is, however, expensive. For large groups, however, the robustness makes the cost justifiable. Parasol is purpose-built for the UCSC kilocluster and throws out some of the generalities for increased performance. PBS and ProPBS are popular DRMs, and if you're an academic user, you can get ProPBS for free. SGE is a relative newcomer but has a strong following, partly due to the fact that it's an open source project.

Table 12-3. DRM software

Product	**Description (as advertised)**
Condor	Condor is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, Condor provides a job-queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to Condor; Condor then places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion. http://www.cs.wisc.edu/condor
LSF	• Platform LSF 5 is built on a grid-enabled, robust architecture for open, scalable, and modular environments. • Platform LSF 5 is engineered for enterprise deployment. It provides unlimited scalability with support for over 100 clusters, more than 200,000 CPUs, and 500,000 active jobs. • With more than 250,000 licenses spanning 1,500 customer sites, Platform LSF 5 has industrial-strength reliability to process mission-critical jobs reliably and on time. • A web-based interface puts the convenience and simplicity of global access to resources into the hands

12.5 Software Tricks

In addition to choosing appropriate BLAST parameters and optimizing your hardware set, you can use a few software tricks to increase your BLAST performance. Most of these tricks involve splitting or concatenating sequences into optimal-sized pieces because very large and very small sequences are inefficiently processed by BLAST.

12.5.1 Multiplexing/Query Packing

Input and output (I/O) can become a large fraction of the overall CPU load when the search parameters are insensitive, such as when running BLASTN. If you find yourself running a lot of BLASTN searches, you can pack multiple queries together and reduce the overhead of reading the database repeatedly. For example, let's say you have a collection of 100,000 ESTs from your favorite organism and you want to search them against all other ESTs in the public database. If you search them one at a time, you will perform 100,000 BLAST searches and therefore have to read the database 100,000 times. It should go without saying that caching is essential in such a task.

But what if you glue the sequences together in groups of 100? Well, you've just cut your database I/O down to 1 percent of what it used to be, which can be a significant savings. For ESTs and other sequences of this length, the speed up is typically tenfold. This technique is called *multiplexing* or *query packing*. It isn't as simple as it sounds because there must be a way to prevent alignments from bridging the sequences, the coordinates must be remapped, and the statistics need to be recalculated. MegaBLAST, part of the NCBI-BLAST distribution, is a specialized version of BLASTN that multiplexes queries and includes a variety of other optimizations. It's really fast, and anyone doing a lot of BLASTN searches should use this program. You can find more information about MegaBLAST in Chapter 9 and Chapter 13. Query packing can also be accomplished with a single, sophisticated Perl script (see MPBLAST at http://blast.wustl.edu).

12.5.2 Query Chopping

Larger sequences require more memory to search and align. This can blow away your cached database, or worse, cause the computer to start swapping (using the disk for RAM). In addition, for a variety of reasons, larger query sequences are processed less efficiently. One way to solve this problem is to divide the query sequence into several segments, search them independently, and then merge the results back together. This is called *query chopping* and is effectively the opposite of query packing. The main difficulty with query chopping is dealing with alignments that cross the boundaries between segments.

Both NCBI-BLAST and WU-BLAST let you specify that only a subsequence of a large query sequence is to be searched (see the -L parameter in Chapter 13 and the newstart and nwlen parameters in Chapter 14). Currently, this works a little better for WU-BLAST because alignments seeded in a restricted region can extend outside this region, so there's no need to stitch together the alignments between neighboring segments. The following Perl script searches chromosome-sized sequences in 100-KB segments using WU-BLAST. All coordinates and statistics are identical to a search with an entire chromosome. Note that complexity filters are currently applied to the whole sequence, so apply these filters ahead of time.

```
#!/usr/bin/perl -w
use strict;
die "usage: $0 <wu-blast command line>\n" unless @ARGV >= 3;
my ($BLAST, $DB, $Q, @P) = @ARGV;
die "ERROR ($0): single FASTA files only\n" if `grep -c ">" $Q` > 1;
my $params = "@P";
die "ERROR ($0): filter ahead of time\n" if $params =~ /filter|wordmask/;
open(FASTA, $Q) or die;
my $def = <FASTA>;
my $count = 0;
while (<FASTA>) {$count += length($_) -1}
my $segment = 100000;
for (my $i = 1; $i <= $count; $i += $segment) {
    system("$BLAST $DB $Q  nwstart=$i nwlen=$segment");
}
```

12.6 Optimized NCBI-BLAST

The source code for NCBI-BLAST is in the public domain, and anyone can modify it without restriction (ftp://ftp.ncbi.nih.gov/toolbox/ncbi_tools). It's therefore not surprising that there are a number of variants. The rest of this chapter discusses three of them.

12.6.1 Apple/Genentech BLAST

Macintosh G4 computers have an additional vector processing unit called VelocityEngine or Altivec that can process several similar instructions in parallel. Apple Computer and Genentech collaborated to rewrite portions of NCBI-BLAST to take advantage of the Altivec processor. These modifications affect the seeding phase of BLASTN. The result, AG-BLAST, significantly outperforms NCBI-BLAST under certain conditions.

Table 12-5 shows an experiment in which a *Caenorhabditis elegans* transcript (F44B9.10) was searched against the *Caenorhabditis briggsae* genome using various word sizes but otherwise default parameters (the hardware is a 550-MHz PowerBook). For cross-species work, it's generally a good idea to employ word sizes slightly smaller than the default 11 to minimize the chance of missing meaningful similarities. Here, AG-BLAST has a significant speed advantage over NCBI-BLAST. AG-BLAST also runs faster at very large word sizes, which is useful if you are matching sequences that are expected to be identical or nearly identical (e.g., mapping ESTs to their own genome).

Table 12-5. Apple/Genentech BLAST

W	NCBI-BLAST (sec)	AG-BLAST (sec)	Speed increase
8	56.9	37.9	1.5 x
9	50.0	9.5	5.3 x
10	46.6	5.5	8.5 x
11	2.9	2.8	1.0 x
15	2.1	2.1	1.0 x
20	1.4	1.0	1.4 x
30	1.4	0.6	2.3 x
40	1.4	0.5	2.8 x

AG-BLAST does have a few disadvantages. First, the version may be slightly out of date with respect to NCBI-BLAST. The current version of AG-BLAST is based on 2.2.2, while NCBI-BLAST is up to Version 2.2.6. Not all changes are backward-compatible; for example, the latest preformatted databases require Version 2.2.5. Second, AG-BLAST doesn't work with multiple CPUs. You can execute more than one job at a time, but you can't use the *-a* option to increase the number of CPUs used by a single process. Finally, the minimum word size on AG-BLAST is 8, or one greater than the NCBI-BLAST minimum. See http://developer.apple.com/hardware/ve/acgresearch.html for more information.

12.6.2 Paracel-BLAST and BlastMachine

Paracel makes an NCBI-BLAST derivative called Paracel-BLAST and sells it with a prepackaged computer cluster

Part V: BLAST Reference

Chapter 13. NCBI-BLAST Reference

This chapter describes the parameters and options for the NCBI suite of BLAST programs. The NCBI distribution includes the *blastall* program, plus several ancillary programs that are either necessary for *blastall* (e.g., *formatdb*) or provide other BLAST-like searches that aren't included within *blastall* (e.g., *blastpgp and blastclust*). This reference also describes the various command-line parameters for the most important executables.

13.1 Usage Statements

If you forget the syntax for a particular parameter, you can view a usage statement from most programs by typing the program name followed by a dash (in some cases the dash isn't required, but it's easier to remember to use a dash with all programs). For example:

```
blastall -
formatdb -
fastacmd -
megablast -
bl2seq -
blastpgp -
blastclust -
```

13.2 Command-Line Syntax

All parameters for NCBI-BLAST programs are single letters and must be preceded by a single dash. Unlike many common Unix programs, the parameters for NCBI programs are never concatenated. All parameters may take arguments, including those that operate as true/false (T/F) switches. For such switches, the T/F is case-insensitive, and the argument may be omitted, in which case the switch is set to T. Finally, the space between the parameter and the argument is optional. The following commands are all identical.

```
formatdb -i db -o T -V t
formatdb -i db -o -V
formatdb -idb -ot -VT
formatdb -idb -o -V
```

The following command, however, is illegal because it tries to set -o to a value of V.

```
formatdb -idb -oV
```

13.3 blastall Parameters

blastall is controlled by several parameters. Many of the parameters have default settings and don't need to be explicitly assigned. Consider this simple command:

```
blastall -p blastp
```

Behind the scenes, this command is converted to:

```
 blastall -p blastp -d nr -i stdin -e 10 -m 0 -o stdout -F T -G 11 -E 2 -X 15 -v 500
-b 250 -f 11 -g T -a 1 -M BLOSUM62 -W 3 -z 0 -K 0 -Y 0 -T F -U F -y 0.0 -Z 0 -A 40
```

You can see that many parameters are set without your express knowledge. These parameters affect the results of your experiment and, as reinforced many times throughout the book, you should try to understand these parameters and set them to fit each experiment.

The following reference section explains all the parameters available for *blastall* and lists the default values that are used if not explicitly set. The table was compiled according to the default values for the five basic programs. Although *megablast* can be run from within *blastall* (-n T), you should use the standalone program. The parameters for it are presented later in the chapter.

-a [integer]

Default: 1 ***Programs: All***

Sets the number of processors to use on of processors. If you have multiple queries, you will get better throughput by executing multiple BLAST searches. For insensitive searches such as default BLASTN, setting -a to a higher value may not appreciably improve speed if disk I/O is the bottleneck.

-A [integer]

Default: blastn 0, others 40 ***Programs: All***

Sets the multiple-hit window size. When BLAST is set to two-hit mode, this option requires two word hits on the same diagonal to be within [integer] letters of each other in order to extend from either one. The larger the [integer], the more sensitive BLAST will be. Setting [integer] to 0 sets the default behavior of 40, except for blastn, whose default is single word hit. To specify one-hit behavior, set -P 1.

-b [integer]

Default: 250 ***Programs: All***

Truncates the report to [integer] number of alignments. There is no warning when you exceed this limit, so it's generally a good idea to set [integer] very high unless you're interested only in the top hits.

-B [integer]

Default: Optional ***Programs: blastn, tblastn***

Sets the number of queries to concatenate in a single search. Concatenating queries accelerates the search because the database is scanned just one time. This is the principle underlying megablast, but the implementation is different in blastall.
This option is new in Version 2.2.6 and still experimental. The specified [integer] must be the number of sequences in the query file. If it's less, only the first set of [integer] sequences is used. Also, the output is very different than you would expect. All the query names are listed, and then all the one-line summaries are given, followed by the alignments, and finally, one footer is produced for the whole report. Given this format, it's very difficult to discern which alignments belong to which query. This option should not be used in its current implementation.

-d [database]

13.4 formatdb Parameters

formatdb turns FASTA files into BLAST databases (ASN.1 format is also acceptable, but because it isn't commonly used, it isn't covered in this book. You can find more information about ASN.1 at http://www.ncbi.nlm.nih.gov/Sitemap/Summary/asn1.html/). Chapter 11 discusses the typical methods for building BLAST databases and examines the NCBI identifier syntax required for some aspects of *formatdb* and *blastall*. Here are a few sample command lines:

```
formatdb -i protein_db
formatdb -p F -i nucleotide_db
zcat est*.gz | formatdb -p F -i stdin -o -n est -v 2000000000
```

The following reference lists the default value for each *formatdb* parameter.

-B [file]

Default: Optional

Specifies a binary GI output file. The advantage of using a binary GI file is that it's smaller than a corresponding text file and can be read directly into memory without being parsed. See the -F option.
To convert a text GI file to binary, use the following command:
formatdb -F text_gi_list -B binary_gi_list

-F [file]

Default: Optional

Specifies a GI file, either text or binary. This is used for creating an alias database that doesn't contain sequences, but pointers to sequences stored in another database (which may be an alias database as well). See the -L parameter. The databases must use the NCBI FASTA identifier syntax, include GI numbers, and be indexed with -o.

-i [file]

Default: Required

Sets the input FASTA file. You may specify that input come from stdin with -i stdin, but you must also set the -n parameter to give it a name. If you wish to make a single BLAST database from multiple FASTA files, pipe them to formatdb as follows:
cat file1 file2 file3 | formatdb -i stdin -n my_db

-l [file]

Default: formatdb.log

Specifies an output log file. Log messages are appended to this file.

-L [file]

Default: Optional

Creates an alias database, which has several uses. It can be a simple synonym for another database, a selection of specific records from a database (see the -F option), or a static virtual database. Alias databases have the .pal or .nal extension, depending on whether they are proteins or nucleotides.
To create an alias database with a selected set of GI numbers:
formatdb -i db -F gi_list -L alias_name -p [T/F]
To merge databases, first create a synonymous alias and then edit it to include additional database names. Chapter 11 covers this process in more detail.

13.5 fastacmd Parameters

fastacmd retrieves sequences, individually or in batches, from BLAST databases. When using it, you don't have to keep FASTA files on your file system after you've formatted the BLAST database. Sequences are stored in a case-insensitive format, however, so if you use lower- and uppercase for semantic purposes, this information will be lost.

Here are a few sample command lines using *fastacmd*:

```
fastacmd -d nr -s P02042
fastacmd -d nr -s 12837002,P02042
fastacmd -d nr -D
fastacmd -d est -i file_of_gi
cat file_of_gi | fastacmd -d est -i stdin
```

The following reference lists the default value for each *fastacmd* parameter.

-a [T/F]

Default: F

Retrieves all accessions even duplicates when using -s or -i to retrieve sequences. If -a isn't set, only the first accession of duplicates is retrieved.

-c [T/F]

Default: F

Uses Control-A as a nonredundant definition line separator. This parameter applies only to nonredundant databases with concatenated definition lines. By default, a normal space is used as the separator. Using Control-A unambiguously separates sequence definitions.

-d [string]

Default: nr

The database from which to retrieve sequences.

-D [T/F]

Default: F

Dumps the entire database in FASTA format.

-i [file]

Default: Optional

A batch retrieval. The format of the text file is one GI or accession per line. stdin is a valid file.
cat file_of_gi | fastacmd -d est -i stdin

-I

Default: Optional

Prints information about a formatted database. Overrides all other retrieval options. Needs to be used with -d.
fastacmd -d my_db -I

-l [integer]

13.6 megablast Parameters

megablast is similar to *blastn* but optimized to find near identities very quickly. It's much faster than the standard *blastn*, partly because it uses query packing. The extension algorithm differs from the standard *blastn* and isn't designed for cross-species searches. Many parameters are identical between *megablast* and *blastall*, but some are unique to one program or the other, and some parameters with the same symbol do different things.

Here are a few example command lines:

```
 megablast -d my_db -i my_query -F "m D"
megablast -d my_db -i my_query -D 2 -t 18 -W 11
```

-a [integer]

Default: 1

The number of processors; same as blastall.

-A [integer]

Default: 40

The two-hit algorithm window size; same as blastall.

-b [integer]

Default: 250

The number of database sequences to show; same as blastall, if -D 2 is set.

-d [string]

Default: nr

The database; same as blastall.

-D [0..3]

Default: 0

The type of output. The -m option applies only if -D 2 is set here.

Options

0

One-line output for each alignment in the form of:

'subject-id'=='[+-]query-id' (s_beg q_beg s_end q_end) Score

For example:

```
 'AF071362'=='+AF071357' (1 715 200 920) 8
```

Score for non-affine gapping parameters (the default) is the total number of differences (mismatches + gaps); it's the actual raw score when using affine gapping.

1

Same as the output of -D 0, but additionally shows the endpoints and percent identity for each ungapped segment in the alignment

13.7 bl2seq Parameters

bl2seq runs the basic BLAST searches on two sequences. Many parameters are identical between *bl2seq* and *blastall*, but some are unique to one program or the other, and some parameters with the same symbol do different things.

Here are a few sample command lines:

```
 bl2seq -p blastp -i protein1 -j protein2
bl2seq -p blastn -i nucleotide1 -j nucleotide2 -F F -D 1
bl2seq -p blastx -i nucleotide -j protein
bl2seq -p tblastn -i protein -j nucleotide
bl2seq -p tblastx -i nucleotide1 -j nucleotide2
```

The following reference describes the parameters for *bl2seq*.

-a [file]

Default: Optional

Specifies the SeqAnnot output file. The [file] will be in the Abstract Syntax Notation 1 (ASN.1) format for import into and use with the NCBI toolbox.

-A [T/F]

Default: F

Input sequences are NCBI identifiers. When set to T, the program makes an online connection to the NCBI databases to retrieve the FASTA sequences.
bl2seq -A -p blastx -i AF287139 -j AAG39070
(This function was just enabled in the 2.2.6 release.)

-d [real number]

Default: 0

Sets the theoretical size of the database. This is useful for maintaining consistent E-values between blastall and bl2seq searches. Identical to the blastall -z parameter. If -d isn't set, the database size is set to the length of the -j sequence.

-D [0/1]

Default: 0

Sets the output format to tabular, which corresponds to the blastall setting -m 8. The other -m report options available in blastall aren't available in bl2seq.
Unlike the blastall parameter of the same name, -D doesn't set the genetic code for translating database sequences. All bl2seq translations use the standard nuclear genetic code.

Options

0	Traditional
1	Tabular

-e [real number]

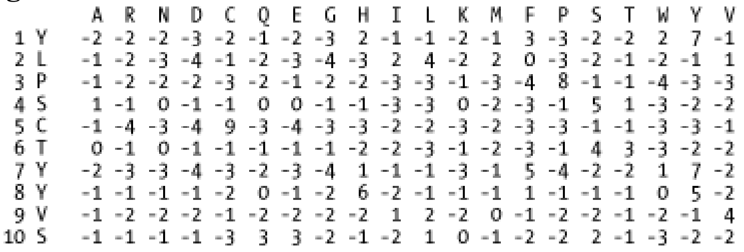
13.8 blastpgp Parameters (PSI-BLAST and PHI-BLAST)

blastpgp is the program used to run PSI-BLAST and PHI-BLAST. These programs are specialized protein BLAST comparisons that are more sensitive than the standard BLASTP search. PSI-BLAST considers position-specific information when searching for significant hits. PHI-BLAST uses a pattern, or profile, to seed an alignment, which is then extended by the normal BLASTP algorithm.

13.8.1 PSI-BLAST

PSI-BLAST (position-specific iterated BLAST) uses a specialized scoring matrix that assigns scores to each position (hence, position-specific) in the query sequence based on alignments defined by consecutive iterations of searches (hence, iterated). The specialized matrix is a position-specific scoring matrix (PSSM) that assigns a score for every amino acid at each position in the query sequence (See Figure 13-1).

Figure 13-1. PSSM for the first 10 amino acids of the coelacanth HoxA11 protein



		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
1	Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1
2	L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
3	P	-1	-2	-2	-2	-3	-2	-1	-2	-2	-3	-3	-1	-3	-4	8	-1	-1	-4	-3	-3
4	S	1	-1	0	-1	-1	0	0	-1	-1	-3	-3	0	-2	-3	-1	5	1	-3	-2	-2
5	C	-1	-4	-3	-4	9	-3	-4	-3	-3	-2	-2	-3	-2	-3	-3	-1	-1	-3	-3	-1
6	T	0	-1	0	-1	-1	-1	-1	-1	-2	-2	-3	-1	-2	-3	-1	4	3	-3	-2	-2
7	Y	-2	-3	-3	-4	-3	-2	-3	-4	1	-1	-1	-3	-1	5	-4	-2	-2	1	7	-2
8	Y	-1	-1	-1	-1	-2	0	-1	-2	6	-2	-1	-1	-1	1	-1	-1	-1	0	5	-2
9	V	-1	-2	-2	-2	-1	-2	-2	-2	-2	1	2	-2	0	-1	-2	-2	-1	-2	-1	4
10	S	-1	-1	-1	-1	-3	3	3	-2	-1	-2	1	0	-1	-2	-2	2	-1	-3	-2	-2

Figure 13-1 shows a portion of a PSSM calculated for the coelacanth Hoxa11 protein (AAG39070). The query amino acids are numbered in the left column with the position-specific scores for each of the 20 amino acids shown across each row. The diverse scores of the three Tyrosines (Y) at positions 1, 7, and 8 highlight the position-specific aspect of this scoring scheme compared to traditional BLAST matrices, which would contain the same scores for Y in all three positions.

The PSSM, or checkpoint file, is created internally by PSI-BLAST, but it can also be exported to a file using the -C option of *blastpgp*. This option is extremely useful. You can use the checkpoint file in subsequent PSI-BLAST (*blastpgp*) searches or as a database entry for the RPS-BLAST program. You can also use the PSSM in a specialized *tblastn* search in *blastall* by using the -p psitblastn and -R <checkpoint file> options with a nucleotide database.

To run PSI-BLAST, the -j parameter must be set to something greater than 1. The default of -j 1 means that there are no iterations and that it's therefore the same as a single BLASTP search. Setting -j sets the maximum number of iterations to run, with the program stopping beforehand if the search comes to convergence. Convergence occurs when no new sequences are found that are better than the E value threshold set by the -h parameter.

Here are a few sample command lines:

```
 blastpgp -d nr -i my_protein -s T -j 5
blastpgp -d nr -i my_protein -R my_protein.ckp -d nr -j 5 -h 0.001
```

13.8.2 PHI-BLAST

PHI-BLAST stands for pattern-hit initiated BLAST. The program uses an input sequence and a defined pattern to query a protein database. The pattern is defined in PROSITE format (http://ca.expasy.org/prosite/)and is used as the seed for the alignment. The pattern is used instead of the words that are usually generated for seeding alignments in BLASTP. Here's a sample profile:

```
ID  HoxA11 pattern1
PA  Y-S-[SA]-X-[LVIM]
```

The profile's syntax has a line starting with ID, followed by two spaces and the name of the pattern. The name is free text. The next line should start with PA, followed by two spaces, and then the pattern in PROSITE format. The PROSITE format is simple. A dash (-) separates letters, an X means any letter, and the brackets ([]) specify a choice of amino acids. You can find more information on the pattern syntax in the *README.bls* file that comes with the

13.9 blastclust Parameters

blastclust clusters a database of protein or nucleotide sequences. It outputs rows of sequence identifiers from the database with clustered sequences occurring on the same row and clusters sorted from largest to smallest. The program can generate a list of clusters for input into another program (e.g., an alignment program such as PHRAP); however, it should be used only on a relatively small number of sequences (10-1000) because it runs only on a single computer, and the RAM requirements quickly exceed most capacities.

Here are a few sample command lines:

```
 blastclust -i my_nucdb -p F -o my_nucdb.clusters
blastclust -i my_pepdb -o my_pepdb.clusters -L 0.7 -S 90
```

The following reference describes parameters used with *blastclust*.

-a [integer]

Default: 1 *Programs: All*

Specifies the number of CPUs to use on a multiprocessor machine.

-b [T/F]

Default: T

Requires coverage on both sequences. If set to T, the program requires both sequences to pass the coverage criteria set with -L before they are called neighbors and clustered together.

-c [file]

Default: Optional

Specifies a configuration file with advanced options. The configuration file is simply a list of the options that you commonly use.

-C [T/F]

Default: F

The crash recovery option. Set it to complete unfinished clustering. Set to T if using the -r option with a file to restore the clustering. Use the same command line as the crashed run with the same -s, with only -C, T, and -r being added. This restarts the run using the hit list file specified by -r and then appending to it (as specified by -s).

-d [file]

Default: Optional

The input file is a BLAST database, not a FASTA file.

-e [T/F]

Default: F

Enables ID parsing in the database-formatted report.

-i [file]

Default: stdin

Chapter 14. WU-BLAST Reference

WU-BLAST was developed and is maintained entirely by Warren Gish. He was one of the original authors of BLAST while at the NCBI but is now at Washington University in St. Louis (where the WU comes from). Development began in 1994 at Version 1.4, before BLAST had gapped alignments. Quite a lot has changed since then. Paradoxically, WU-BLAST is more similar to the original BLAST than the current NCBI version.

WU-BLAST is useful because it has more command-line parameters that allow advanced users to control the program with more precision. It is also faster. Table 14-1 displays features unique to WU-BLAST or significantly different from NCBI-BLAST.

Table 14-1. WU- and NCBI-BLAST feature differences

Feature	**WU-BLAST**	**NCBI-BLAST**
Word size	Any word size for any program mode. Neighborhood words are turned off for word sizes of 5 or greater, but may be activated by setting an explicit value for T.	*blastn* has a minimum word size of 7. *blastp*, *blastx*, *tblastn*, and *tblastx* have word sizes of 2 or 3. Neighborhood words are never used for *blastn*.
Nucleotide scoring	Choice of match/mismatch or scoring matrix.	Only match/mismatch scoring.
Nucleotide statistics	Karlin-Altschul parameters are available for several match/mismatch values and gap costs.	Karlin-Altschul parameters are always computed without respect to gap costs. Reported E-values may greatly overestimate significance.
`altscore`	Allows score modification for any matrix (e.g., to set stop scores lower).	Nothing similar.
H, K, L, gapH, gapK, gapL	Especially useful when using unsupported scoring schemes; allow the provision of values for Karlin-Altschul parameters.	Nothing similar. Unsupported scoring schemes are fatal errors.
Alias databases	No, but virtual databases offer similar functionality.	Yes, both alias and virtual databases are supported.
Gapped alignment	All programs.	All programs except *tblastx*.
/etc/sysblast	Allows systems administrators to set system-wide resource restrictions.	Nothing similar.
Database subset selection	Yes, via dbrecmin and dbrecmax.	No, but alias databases can be used for static splitting.
	The nwstart and nwlen parameters	-L restricts both seeding and

14.1 Usage Statements

All WU-BLAST programs provide usage statements if they are executed without any arguments. They are sometimes lengthy, so it's best to pipe them through a pager such as *less* or *more*.

```
blastn | more
xdformat | less
xdget | less
```

14.2 Command-Line Syntax

WU-BLAST command-line syntax isn't uniform between all programs. The BLAST programs *blastn*, *blastp*, *blastx* , *tblastn*, and *tblastx* use a slightly different syntax than do *xdformat*, and *xdget*.

The BLAST program options come after the mandatory arguments of database and query sequence. The command-line structure is as follows:

```
[program name] [blast database] [query sequence] [parameters]
```

The parameter names in the BLAST programs and their arguments have some flexibility. The following command lines are all identical:

```
blastn db query E=10
blastn db query -E 10
blastn db query E 10
blastn db query -E=10
```

This book uses the first form to avoid confusion with NCBI-BLAST.

xdformat and *xdget* use the traditional Unix syntax where the parameters precede the mandatory arguments:

```
[program name] [parameters] [mandatory arguments]
```

The xdformat and *xdget* options are all single letters preceded by a single dash. For parameters that require a value, a space between the parameter and its value is optional. As is typical for Unix programs, a double dash indicates the end of command-line options and a single dash signifies *stdin*.

```
xdformat -p protein_db
xdformat -n -I nucleotide_db
zcat fasta.*.gz | xdformat -n -o my_db -- -
```

14.3 WU-BLAST Parameters

WU-BLAST has many control parameters, some of which are esoteric and rarely useful. The most important parameters are listed here.

altscore=[string]

Default: Off

Defines an alternate scoring system for any pair of letters. For example, altscore="M M -3" changes the score of M-M pairs to -3, and altscore="A C 4" gives a score of 4 if the query is A and the subject is C. Letters may be designated as any to change an entire row or column. The score can be given as min or max for the minimum and maximum scores in the matrix or na to make the score infinitely low. To set the score of all rows and columns containing stop codons to negative infinity, set altscore="* any na" and altscore="any * na". If you change the scoring parameters, you may also want to adjust gapL, gapH, and gapK.

See also

nogap, gapL, gapH, gapK

B=[integer]

Default: 250

Sets the number of database hits to report. A warning is issued if this number is exceeded. It is typical to set this parameter to a very high value, such as B=100000, to ensure that no alignments are missed.

bottom

Default: Off *__Programs: blastn, tblastx, blastx__*

Search only the bottom strand of the query.

See also

top

cpus=[integer]

Default: 4 for blastn; all for blastp, blastx, tblastn, and tblastx

Sets the number of processors to use. If not set, all processors on the system may be used except blastn, which will limit itself to 4. See Chapter 10 for information on the /etc/sysblast file used for setting systemwide resource limitations.

dbrecmax=[integer]

Default: Last database record

Last database record number to search.

See also

dbrecmin, qrecmin, qrecmax

dbrecmin=[integer]

14.4 xdformat Parameters

xdformat formats BLAST databases from FASTA files. It also reports descriptive information about the database and dumps the entire content to FASTA format.

Here are some examples:

```
 xdformat -n files
xdformat -p files
zcat fasta.*.gz | xdformat -o my_db -n  --  -
xdformat -n -i database
xdformat -n -r datatbase > fasta_file
```

-A [0..2]

Default: 2

When indexing accession.version identifiers, you have three indexing options:

0
Accession only; version isn't stored
1
Stored as accession.version
2
Stored as both accession only and accession.version

-a [database]

Appends sequences to the named database. If the database is indexed, the appended sequences will also be indexed.

-c [character]

Default: Off

If an invalid letter is encountered, xdformat terminates and reports an error message. If this occurs, check the sequence file for errors. After checking, you may either skip illegal characters with -k or change them to a legal character with -c. The typical operation for nucleotides is to set -c N, and for proteins -c X.

See also

-k

-D [integer]

Default: Unlimited

Sets the maximum length for definition lines.

-d [string]

Default: None

Sets a user-defined release date for the database. The date may have 63 characters at most.

See also

14.5 xdget Parameters

xdget retrieves files in FASTA format from databases formatted with *xdformat* (not *formatdb*, *pressdb*, or *setdb*). The database must have been indexed prior to using *xdget* (see -I and -X in the previous section Section 14.4").

Here are a few example command lines. If identifiers contain vertical bars, as in the second example, you have to enclose the string in quotes to prevent the shell form interpreting them as pipes. This isn't required for identifier files.

```
xdget -n db 12345
xdget -p nr 'gi|11611819|gb|AAG39070.1|'
xdget -n -f db files_of_ids
```

-A [n, 0]

Default: n

Given an accession number without a version, xdget retrieves the latest version number. This parameter is set explicitly with -A n. If -A 0 is set, the earliest version number is retrieved.

See also

-d, -N

-a [integer]

Default: 1

The -a and -b parameters retrieve a subsequence. For example, if you want to retrieve just nucleotides 1 to 100, include -a 1 -b 100. For nucleotide sequences, if -b is greater than -a, the sequence is returned as its reverse-complement.

See also

-b, -r, -t

-b [integer]

Default: 0, end of sequence

See -a above.

-d

Default: Off

Ordinarily, when duplicate identifiers are present, only one is retrieved. With -d, all duplicates are reported. Having duplicate identifiers is generally not a good idea.

See also

-A, -N

-D [integer]

Default: Unlimited

Sets the maximum definition line length. Using definition lines to store arbitrary sequence data is common. This option is useful when you don't need the whole definition line.

Part VI: Appendixes

Appendix A. NCBI Display Formats

NCBI-BLAST has several options for displaying sequence alignments. These options are available for the five basic BLAST programs (BLASTN, BLASTP, BLASTX, TBLASTN, TBLASTX), PSI-BLAST, PHI-BLAST, and MegaBLAST. For all programs, these formats are selected by using the -m option; however, in MegaBLAST, the formats must first be set with the -D 2 option to use classic BLAST formatting. The next section gives a brief description of each option, followed by a detailed explanation and an example.

A.1 Brief Descriptions

The alignment display format is set with the -m option followed by a number from 0 to 11 as you can see in the following table.

Option number	Brief description
0	The default pairwise display. Classic BLAST format.
1-6	Various types of query-anchored multiple sequence alignments. The query is anchored and the aligned regions of the subjects are displayed underneath.
7	eXtensible Markup Language (XML) output.
8	Tabular output, without header lines.
9	Tabular output, with header lines.
10	ASN.1 text
11	ASN.1 binary

A.2 Detailed Descriptions and Examples

This section includes detailed descriptions of each format, followed by an example. To create the examples, the authors performed a BLASTP search of the coelacanth HoxA11 protein sequence (AAG39070) versus the HoxDB.pep database, which is included in the online supplement.

A.2.1 Option 0: Pairwise Alignments

Option 0 is the default alignment and the classic BLAST format. The definition line of the subject is given at the top of each entry, marked with the greater-than sign (>) and followed with the subject's total length. For each HSP of a subject, the score, expect, identities, positives, and gaps are reported and followed by a pairwise alignment. For the pairwise alignment in Figure A-1, the query sequence is shown on the first row and the subject on the third row. Gaps are represented in each as a dash (-). Between the query and subject lies the alignment row, which shows the residue for identities, a plus (+) for positive scoring alignments, and a dot (.) for mismatches. In BLASTN alignments, the middle row has vertical bars (|) for identities and nothing for mismatches.

Figure A-1. Option 0: Standard pairwise alignment

```
>HoxA11_chick gi|399992|sp|P31258|HXAB_CHICK Homeobox protein Hox-A11 (Ghox-1I)
           (Chox-1.9)
          Length = 297

 Score =  318 bits (816), Expect = 4e-091
 Identities = 163/216 (75%), Positives = 178/216 (82%), Gaps = 14/216 (6%)

Query: 1   YLPSCTYYVSGPDFSSLPSFLPQTPSSRPMTYSYSSNLPQVQPVREVTFRDYAIDTSNKW 60
           YLPSCTYYVSGPDFSSLPSFLPQTPSSRPMTYSYSSNLPQVQPVREVTFR+YAID S+KW
Sbjct: 15  YLPSCTYYVSGPDFSSLPSFLPQTPSSRPMTYSYSSNLPQVQPVREVTFREYAIDPSSKW 74

Query: 61  HPRSNLPHCYSTEEILHRDCLATTTASSIGEIFGKGNANVY-HPGSSTSSNFYNTVGRNG 119
           HPR+NLPHCYS EEI+HRDCL +TT +S+GE+FGK  ANVY HP ++ SSNFY+TVGRNG
Sbjct: 75  HPRNNLPHCYSAEEIMHRDCLPSTTTASMGEVFGKSTANVYHHPSANVSSNFYSTVGRNG 134

Query: 120 VLPQAFDQFFETAYGTTENHSS-DYSADKNSDKIP-----SAATSRSE------TCRETD 167
           VLPQAFDQFFETAYGT EN SS DY  DK+ +K P     +AATS SE
Sbjct: 135 VLPQAFDQFFETAYGTAENPSSADYPPDKSGEKAPAAAGATAATSSSEGGCGGAAAAAGK 194

Query: 168 EKERREES-SSPESSSGNNEEKSSSSSGQRTRKKRC 202
           E+ RR ES SSPESSSGNNEEKS SSSGQRTRKKRC
Sbjct: 195 ERRRRPESGSSPESSSGNNEEKSGSSSGQRTRKKRC 230
```

A.2.2 Query-Anchored Alignments

All query-anchored formats (1-6) are multiple-sequence alignments. They share the same general form, with the query repeated at the top of each line and all matching subjects aligned on subsequent lines. The difference between showing identities and not showing them is counterintuitive. For the options that show identities (1 and 3), identical residues are symbolized with a dot (.), similar amino acids are in uppercase, and mismatches are in lowercase. For the options without identities (2, 4, 5 and 6) every residue is shown with identities and similar residues in uppercase and mismatches are in lowercase.

A.2.3 Option 1: Query-Anchored Showing Identities

In the format shown in Figure A-2, the identical residues are represented by a dot (.) and insertions and deletions are represented in the subject sequences, but not the query.

Figure A-2. Option 1: Query-anchored showing identities

```
QUERY        1   YLPSCTYYVSGPDFSSLPSFLPQTPSSRPMTYSYSSNLPQVQPVREVTFRDYAIDTSNKW 60
HoxA11_chick 15  ..................................................E....p.S.. 74
HoxA11_human 14  ..................................................E...EpAT.. 73
HoxA11_mouse 15  ..................................................E...EpAT.. 74
HoxD11_shark 2   ..........A.....VsT...p.t.cQ-..Fp.....a.......LS....GLEhpT.. 60
HoxD11_chick 15  ...G.A....ps...Tk....s.S-..cq..Fp......H.......A..E.GLErG-.. 72
HoxD11_newt  15  ...G.A....psE..Tkt...s.g-..c.V.Fp......H......MA..E.Gwrr.-.. 72
HoxC11_human 33  ........M--.E..TVs.....A..-.qIS.p..AQV.---.....S---.GLEp.G.. 83
HOxD11_mouse 2   ...G.A...Aps..A.k....s.-...cq..Fp......H.......A....GLErA-.. 60
                                                       \
                                                       |
                                                       A
HoxD11_human 15  ...G.A...Aps..A.k....s.-...cq..Fp......H.......A....GLErA-.. 73
                                                       \
                                                       |
                                                       A
```

Appendix B. Nucleotide Scoring Schemes

Nucleotide scoring schemes are often summarized by their target frequency, which is the expected frequency of nucleotide pairs. This frequency is usually expressed as the expected percent identity. For example, the +1/-1 match/mismatch values have a target frequency of 75 percent identity. But this is true only for ungapped alignments between sequences of infinite length. Short sequences and gapped alignment change the true target frequency. In the following table, the target frequencies for a variety of match (+), mismatch (-), and simple gap costs (gap) are calculated for pairs of sequences of length 100, 500, and 1,000 by performing local alignments of random nucleotide sequences of unbiased composition. The theoretical target frequency (TF) is included for comparison.

+	**-**	**Gap**	**TF**	**100**	**500**	**1,000**
1	1	1	75	55	49	49
1	1	2	75	79	70	69
1	1	3	75	85	79	79
1	2	2	95	93	89	88
1	2	3	95	98	96	96
1	2	4	95	98	97	97
1	3	3	99	99	99	98
5	4	4	65	51	48	48
5	4	5	65	53	49	49
5	4	6	65	55	50	49
5	4	7	65	59	51	50
5	4	8	65	62	52	50
5	4	9	65	64	55	53
5	4	10	65	67	59	57
5	4	11	65	69	61	60
5	4	12	65	71	63	62
5	5	5	75	55	49	49

Appendix C. NCBI-BLAST Scoring Schemes

`Statistical parameters for gapped alignment are determined empirically from random sequence alignments. NCBI-BLAST provides several combinations of scoring matrices and gap costs. These are the only combinations of gap penalties that can be used with a given matrix. The default values for each matrix are in boldface in the following table. If your BLAST distribution doesn't have all the listed matrices, you can find more at ftp://ftp.ncbi.nih.gov/blast/matrices.

C.1 NCBI-BLAST Matrices and Gap Costs

Matrix	**G**	**E**	λ	**K**	**H**
BLOSUM62	32767	32767	0.318	0.134	0.401
	11	2	0.297	0.082	0.27
	10	2	0.291	0.075	0.23
	12	1	0.283	0.059	0.19
	9	2	0.279	0.058	0.19
	8	2	0.264	0.045	0.15
	11	1	0.267	0.041	0.14
	10	1	0.243	0.024	0.10
	7	2	0.239	0.027	0.10
	6	2	0.201	0.012	0.061
	9	1	0.206	0.010	0.052
BLOSUM80	32767	32767	0.343	0.177	0.657
	25	2	0.342	0.17	0.66
	13	2	0.336	0.15	0.57
	9	2	0.319	0.11	0.42
	11	1	0.314	0.095	0.35
	8	2	0.308	0.090	0.35
	10	1	0.299	0.071	0.27
	7	2	0.293	0.070	0.27
	9	1	0.279	0.048	0.20

Appendix D. blast-imager.pl

blast-imager.pl creates a graphical summary of a BLAST report using Thomas Boutell's GD graphics library (http://www.boutell.com/gd), which has been ported to Perl by Lincoln Stein (http://stein.cshl.org/WWW/software/GD). The latest versions support PNG or JPEG format, but not GIF. Depending on which GD installation you have, you may want to edit the output section in the program. Just toggle the comments from one output to another.

To use the *blast-imager.pl* program, the data must be in NCBI tabular format (-m 8) as described in Appendix A. Appendix E presents a program for converting standard BLAST output to tabular format. The simplest way to use *blast-imager.pl* is to have it read an entire BLAST report (but not concatenated BLAST reports).

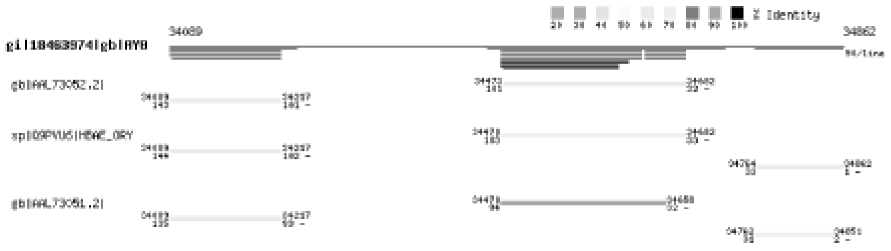
```
blast-imager.pl blast_table > blast.gif
```

In Figure D-1, a *Takifugu rubripes* genomic sequence (AY016024.1) containing an alpha globin gene cluster was used to search the nr protein database. The output of the BLASTX report was converted to tabular format with *blast2table.pl* in the region of 34,000 to 35,000.

```
blast2table.pl -m 34000 -n 35000 report | blast-imager.pl > fugu.gif
```

Database matches and alignments are shown in the order they appear in the report. Only the best alignments are color-coded (the exact number is limited by the image size). Query coordinates are given above the subject coordinates for each alignment. A minus strand is indicated by a hyphen (-) at the subject end (as shown in Figure D-1). The number of alignments at each position in the query is given by the thin black lines under the query sequence. Each line in this figure represents 90 alignments. This scale changes as necessary.

Figure D-1. Selected region of a BLASTX report



The complete program is listed here and may be downloaded from asblast-imager.plfrom this book's web site.

```
#!/usr/bin/perl -w
# blast-imager.pl
use strict;
use GD;

# Parse tabular data
my ($Q, %S, @S);
my ($MAX, $MIN, $HSPs) = (0, 1e20, 0);
while (<>) {
    my ($q, $id, $p, $l, $m, $g, $qb, $qe, $sb, $se, $e, $b) = split;
    $Q =$q;
    if ($qb > $qe) {($qb, $qe, $sb, $se) = ($qe, $qb, $se, $sb)}
    $MAX = $qe if $qe > $MAX;
    $MIN = $qb if $qb < $MIN;
    push @S, $id if not defined $S{$id};
    push @{$S{$id}}, [$qb, $qe, $sb, $se, $p];
    $HSPs++;
}

# Setup graph
my ($L, $B, $R, $H, $F) = (150, 600, 50, 40, 20); # graph regions
my ($W, $Hsep, $Ssep) = (3, 14, 18); # line width and spacing
my $vsize = $H + $F + $Hsep * $HSPs + $Ssep * (keys %S);
$vsize = 100 if $vsize < 100;
$vsize = 600 if $vsize > 600;
my $hsize = $L + $B + $R;
my $SCALE = $B / ($MAX - $MIN + 1);
```

Appendix E. blast2table.pl

It's often useful to have both the standard report and tabular data for the same search. This program converts standard WU-BLAST or NCBI-BLAST output to the NCBI tabular format (-m 8) described in Appendix A. It supports concatenated BLAST reports and is more efficient than most full-featured BLAST parsers. *blast2table.pl* can be used either on a file or in a pipe:

```
blast2table.pl my_blast_output > my_table_output
blastp nr query | blast2table.pl > table_from_wu-blast
```

The Unix *tee* program can create both the standard output and a table at the same time:

```
blastall -p blastp -d nr -i query | tee standard | blast2table.pl > table
```

blast2table.pl also has a few useful filtering options (see Table E-1). The following command displays only those with an alignment of over 50 percent identity and with a bit score greater than 20:

```
blast2table.pl -p 50 -b 20 blast_output
```

Table E-1. blast2table.pl options

Option	**Description**
-b	Minimum number of bits for each alignment
-e	Maximum E-value for each alignment or alignment group
-m	Minimum coordinate of the query sequence
-n	Maximum coordinate of the query sequence
-p	Minimum percent identity for each alignment

The complete program is listed next and may be downloaded as blast2table.pl from this book's web site.

```
#!/usr/bin/perl -w
use strict;
use Getopt::Std;
use vars qw($opt_p $opt_b $opt_e $opt_m $opt_n);
getopts('p:b:e:m:n:');
my $PERCENT = $opt_p ? $opt_p : 0;
my $BITS    = $opt_b ? $opt_b : 0;
my $EXPECT  = $opt_e ? $opt_e : 1e30;
my $START   = $opt_m ? $opt_m : 0;
my $END     = $opt_n ? $opt_n : 1e30;

my ($Query, $Sbjct);
my $HSP = "";
while (<>) {
    if    (/^Query=\s+(\S+)/) {outputHSP(  ); $Query = $1}
    elsif (/^>(\S+)/)         {outputHSP(  ); $Sbjct = $1}
    elsif (/^ Score = /) {
        outputHSP(  );
        my @stat = ($_);
        while (<>) {
            last unless /\S/;
            push @stat, $_
        }
        my $stats = join("", @stat);
        my ($bits) = $stats =~ /(\d\S+) bits/;
        my ($expect) = $stats =~ /Expect\S* = ([\d\-\.e]+)/;
```

Glossary

Numbers

1°

The abbreviation for primary. 1° sequence refers to the letters of DNA, RNA, or protein. 1° transcript refers to an unprocessed RNA that still contains its introns.

2°

The abbreviation for secondary. Most frequently used for generalizing protein and RNA structures; for example, the α-helix and hair-pin are common 2° structures.

3´

The end of a nucleic acid sequence; often used with UTR.

5´

The start of a nucleic acid (DNA or RNA) sequence; often used in conjunction with UTR (e.g., 5´UTR). Nucleotide sequences are conventionally written with the 5´ end at the left. DNA molecules are usually double-stranded but when written, usually only the 5´ to 3´ strand is displayed. The complementary strand has reversed polarity (3´ to 5´).

A-G

aa

The abbreviation for an amino acid that is often used when describing the length of a protein (e.g., the average protein is about 300 aa long).

allele

A form of a gene. Typically, the most common form is called wild-type, and each allele is given a specific (and often obscure) name.

amino acid

The basic building block for all proteins. There are 20 common amino acids.

Arabidopsis thaliana

Known by its common name, thale cress, this mustard weed is a favorite organism for plant genetics and molecular biology. It was the first plant with a complete genomic sequence. For more information, see http://www.arabidosis.org .

bit

The contraction for binary digit. The base-2 logarithm of a number is in units of bits.

BLOSUM

The abbreviation for a blocks substitution matrix. Matrix names are followed by a number (e.g., BLOSUM62) that indicate the minimum percent identity between any two aligned sequences.

bp

The abbreviation for base pair. The length of DNA is usually given in bp or nt, Common measures include Kb, Mb, and Gb for thousands, millions, and billions of bp, respectively.

C-terminus

The end of a protein. In text form, the C-terminus of the protein is always at the right.

Caenorhabditis elegans

A nematode (also called a roundworm) that is about 1 mm long and has about 1,000 cells as an adult. *C. elegans* was the first animal to have its complete genome sequenced. See http://www.wormbase.org.

CDS

The abbreviation for a coding sequence. CDS isn't synonymous with exon, since exons may contain noncoding

H-U

homologous

In sequence analysis, homologous means derived from a common ancestor. Sequences are either homologous or they aren't. It is incorrect to say that sequences are 80 percent homologous unless you mean that there is an 80 percent chance of common ancestry. Use percent identity to describe the similarity of alignments.

hydrophilic

Literally, "likes water." Water is a polar molecule that mixes well with other polar molecules. The charged amino acids K, R, D, and E, are examples of hydrophilic amino acids.

hydrophobic

Literally, "fears water." Nonpolar molecules (like those in oils) don't mix well with water. The amino acids L, I, V, and F are particularly hydrophobic.

Karlin-Altschul

The standard local alignment theory is often called Karlin-Altschul statistics after its founding authors.

lambda, λ

The Karlin-Altschul statistical parameter that converts a raw score to a normalized score.

local alignment

An alignment algorithm that finds the optimal subsequence alignment. The alignment may include all letters of each sequence, but it isn't required to do so.

low-complexity sequence

Regions of sequences that are highly predictable—for example, a region that is 90 percent A or T.

methionine

One of the 20 common amino acids. Methionine is abbreviated as M or Met, and is especially important because all proteins begin with a methionine. There is only one codon for this amino acid: ATG.

mutation

Any change in sequence to a DNA molecule.

N-terminus

The start of a protein. In text form, a protein's N-terminus is always at the left.

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of BLAST is a coelacanth.The modern coelacanth, Latimeria chalumnae, is the sole living representative of Actinista, an ancient family of lobe-finned fish. The first live coelacanth was discovered off the east coast of South Africa in 1938 and caused a stir in the scientific community because they were believed to have been extinct for approximately 80 million years. A second, closely related species of coelacanths was found off the coast of Sulawesi in 1998.

Coelacanths grow to about 6 feet long; weigh up to 150 pounds; and are covered with bony, dark blue scales flecked with white. Coelacanths are unlike other living fish in that they have a three-lobed tail, fleshy fins, and a partially developed vertebral column. They are the only living vertebrate with a functional intercranial joint. It's this joint that allows the coelacanth's jaws to open exceptionally wide when inhaling its prey. Coelacanths are opportunistic predators (they eat whatever they can find) and are ovoviviparous (they give birth to live young).

From an evolutionary perspective, coelacanths are unique, and they provide a key perspective on the evolution of many different genes in vertebrate species. With their front and rear paired fins, which move in a similar fashion to the arms and legs of land vertebrates, coelacanths may be one of the closest links to those vertebrates that first crawled out of the sea to live on land more than 350 million years ago.

Mary Anne Weeks Mayo was the production editor and proofreader, and Ann Schirmer was the copyeditor for BLAST. Derek DiMatteo, and Darren Kelly provided quality control. Lucie Haskins wrote the index.

Emma Colby designed the cover of this book, based on a series design by Edie Freedman. The cover image is an original illustration created by Lorrie LeJeune. Emma produced the cover layout with QuarkXPress 4.1 using Adobe's ITC Garamond font.

David Futato designed the interior layout. This book was converted by Julie Hawks to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSans Mono Condensed. The illustrations that appear in the book were produced by Robert Romano and Jessamyn Read using Macromedia FreeHand 9 and Adobe Photoshop 6. This colophon was compiled by Lorrie LeJeune.

The online edition of this book was created by the Safari production group (John Chodacki, Becki Maisch, and Madeleine Newell) using a set of Frame-to-XML conversion and cleanup tools written and maintained by Erik Ray, Benn Salter, John Chodacki, and Jeff Liggett.

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]