

BIOINFORMATICS

FALL 2017

SUPPLEMENTARY READINGS

WEEK 1 READINGS

Regular Expressions for DNA pattern searching

"Regular expressions" (or "RegExes") are a pattern-matching tool utilized by many UNIX-based tools (sed, awk, grep, perl etc). With RegExes you can search for many complex and interesting patterns.

/Pattern/ Pattern to be matched in a regular expression.

In a regular expression, you specify some literal characters and their order you want to match in a text. Some meta characters (e.g. {}()[]+?.) are reserved for special use. Because we are matching only DNA sequence (G's, A's, T's and C's), we do not have to worry about how to specify one of these meta characters literally.

In some applications, there will be no need to insert starting and ending forward slashes. Don't forget that in DNA applications RegExes won't automatically match the reverse-complement!

G	Match a "G"
A	Match an "A"
T	Match a "T"
C	Match a "C"
GATTACA	Match the sequence "GATTACA"
.	Match any character
\n	Match newline

You can follow any character, wildcard, or series of characters and/or wildcard with a repetition. Here's where you start getting some unique abilities not found in IUPAC consensus or PWM (position-weighted matrix) descriptions of DNA motifs:

*	Match 0 or more times
+	Match 1 or more times
?	Match 1 or 0 times
{n}	Match exactly n times
{n,}	Match at least n times
{n,m}	Match at least n but not more than m times

G{5} will match the sequence **GGGGG**.

By default, a quantified subpattern is "greedy", that is, it will match as many times as possible (given a particular starting location) while still allowing the rest of the pattern to match. If you want it to match the minimum number of times possible, follow the quantifier with a "?". Note that the meanings don't change, just the "greediness":

*?	Match 0 or more times
+?	Match 1 or more times
??	Match 0 or 1 time
{n}?	Match exactly n times
{n,}?	Match at least n times
{n,m}?	Match at least n but not more than m times

Greedy and Ungreedy Matching

Perl regular expressions normally match the longest string possible. For instance:

```
SEQUENCE = "GATTATTACCA"
```

```
REGEX = "(A.*T)"
```

```
MATCH = "ATTATT"
```

It matches the first **A**, the last **T**, and everything in between them. But what if you want to match the first **A** to the **T** most closely following it? Use this REGEX:

```
SEQUENCE = "GATTATTACCA"
```

```
REGEX = "(A.*?T)"
```

```
MATCH = "AT"
```

Using Groups () in Matching

Note: Many situations can be done either with groups **()** or character classes **[]**. Groups are less quirky and they more often yield the results you were looking for.

Groups are regular expression characters surrounded by parentheses. They have two major uses:

1. To allow alternative phrases as in **(GATC|AATG|CATG)**. Note that for single character alternatives, you can also use character classes.

2. As a means of retrieving selected text in selection, translation and substitution, used with the **\1**, **\2**, **\3** for first, second and third occurrences of text closed by parentheses etc (**\$1**, **\$2**, **\$3** in perl).

Using Character Classes []

Character classes are alternative single characters within square brackets. If not used carefully, they can yield unexpected results.

Character classes have three main advantages:

1. Shorthand notation, as **[GATC]** instead of **(G|A|T|C)**. The letter **N** matches the literal character "N", which often appears in sequenced DNA to represent an unreadable nucleotide. **R** matches the literal character "R" and not a purine (A or G). Therefore, if you accidentally search for an "R" you will not get a match in DNA sequence. You must write **[AG]** when you want to specify a purine.

ROLL YOUR OWN

perl global substitution command:

```
perl -pi -e 's/pattern1/pattern2/g' file.txt
```

is a command to **substitute pattern1 with pattern2 globally** in the specified **file**, where **pattern1** and **pattern2** are Regular Expressions.

1) Download sequence text file: **DNA_file.txt**

2) Strip file of newlines (\n):

```
perl -pi -e 's/\n//g' DNA_file.txt
```

Why? Your RegEx will not recognize a pattern that occurs over the artifactual newline (hard return; end of the line; line break). Therefore to make sure you do not miss hits in the genome, get rid of newline characters (\n).

- 3) Make new lines (\n):

```
perl -pi -e 's/(G{20}|A{20}|T{20}|C{20})/$1\n/g' DNA_file.txt
```

I like to make new line breaks corresponding to some uninteresting feature not likely to be found in the middle of a target regulatory sequence but frequent enough to break sequence into "lines". Programs like grep, which also uses RegExes, print the content of an entire line, so it's easier to get output with manageable sized lines. The perl expression above replaces a string of 20 mononucleotides, with the same thing followed by a newline.

- 4) Make other substitutions to annotate your DNA sequence:

```
perl -pi -e 's/(.{100}pattern.{100})/\nX $1\n/g' DNA_file.txt
```

Here we look for a pattern everywhere in the file and then break a new line +/- 100 bp from the pattern. You can mark the line with an X or other unique character, which you can later search to pull the line out. Don't forget that Regular Expressions do not normally match the reverse complement. For most patterns, you will have to do two searches, one for each case.

- 5) grep is another simple pattern matching tool

```
grep [options] PATTERN [FILE...]
```

no options = print lines with matching pattern
-c = count # of lines in which pattern is found

- 6) Write matches into new file

```
grep X DNA_file.txt > DNA_matches.txt
```

This looks for lines with an X in DNA_file.txt and then writes that lines into the new file named DNA_matches.txt

- 7) "man program_name" for manual information.

E.g. man grep

E.g. man perl

NOTE: In learning how to use these pattern matching tools, you will make many mistakes. This is good; this is part of learning. However, remember that it is very easy to corrupt your DNA sequence file in this manner. You may want to work with a copy of a backup reference file, which you do not manipulate in the way described here.

For other useful tips, look for information on how to work within a UNIX environment/shell.

Unix Primer - Basic Commands In the Unix Shell

If you have no experience with the Unix command shell, it will be best to work through this primer. The last section summarizes the basic file manipulation commands.

1. Unix Shell

The *shell* is a command programming language that provides an interface to the UNIX operating system. The remainder of this tutorial presents basic commands to use within the UNIX shell.

1. Directories

The shell should start you in your home directory. This is your individual space on the UNIX system for your files. You can find out the name of your current working directory by typing:

```
% pwd  
/Users/username
```

(The '%' designates your command line prompt, and you should type the letters 'p', 'w', 'd', and then "enter" - always conclude each command by pressing the "enter" key. The response that follows on the next line will be the name of your home directory, where the name following the last slash should be your *username*.) The directory structure can be conceptualized as an inverted tree.

No matter where in the directory structure you are, you can always get back to your home directory by typing:

```
% cd
```

(without specifying a directory name).

From your home directory, create a new subdirectory named "primer" for working through this tutorial:

```
% mkdir primer
```

You can remove an empty subdirectory with the following command (but don't do this right now):

```
% rmdir primer
```

(Note: if you do remove "primer", please create it again.)

Now change to the "primer" subdirectory, making it your current working directory:

```
% cd primer
```

1. Files

Files live within directories. You can see a list of the files in your "primer" directory (which should be your current working directory) by typing:

```
% ls
```

Since you just created the directory, nothing will be listed because the directory is empty. Create your first file using the pico text editor:

```
% pico first
```

The pico editor fills the entire console window. You can type text and move the cursor around with the arrow keys; the bottom of the screen presents the commands available. Type the sentence: "My first file." Then press "^O" (hold the left "control" key while pressing 'O') and "enter" to save the file, then "^X" (hold the left "control" key while pressing 'X') to exit pico. Now when you list your files, you will see file "first" listed:

```
% ls  
first
```

You can view a text file with the following command:

```
% cat first  
My first file.
```

("cat" is short for concatenate - you can use this to display multiple files together on the screen.) If you have a file that is longer than your 24-line console window, use instead "more" to list one page at a time or "less" to scroll the file down and up with the arrow keys. Don't use these programs to try to display binary (non-text) files on your console - the attempt to print the non-printable control characters might alter your console settings and render the console unusable.

Copy file "first" using the following command:

```
% cp first 2nd
```

By doing this you have created a new file named "2nd" which is a duplicate of file "first". The file listing reveals:

```
% ls  
2nd      first
```

Now rename the file "2nd" to "second":

```
% mv 2nd second
```

Listing the files still shows two files because you haven't created a new file, just changed an existing file's name:

```
% ls  
first    second
```

If you "cat" the second file, you'll see the same sentence as in your first file:

```
% cat second  
My first file.
```

"mv" will allow you to move files, not just rename them. Perform the following commands:

```
% mkdir sub  
% mv second sub  
% ls sub
```

```
second
% ls
first    sub
```

This creates a new subdirectory named "sub", moves "second" into "sub", then lists the contents of both directories. You can list even more information about files by using the "-l" option with "ls":

```
% ls -l
-rw-r--r--  1 username      group          15 May 22 16:26 first
drwxr-xr-x  2 username      group        512 May 22 17:11 sub
```

(where "*username*" will be your *username* and "group" will be your group name). Among other things, this lists the creation date and time, file access permissions, and file size in bytes. The letter 'd' (the first character on the line) indicates the directory names.

Next perform the following commands:

```
% cd sub
% pwd
/Users/username/primer/sub
% ls -l
-rw-r--r--  1 username      group          15 May 22 16:55 second
% cd ..
% pwd
/Users/username/primer
```

This changes your current working directory to the "sub" subdirectory under "primer", lists the files there, then changes you back up a level. The ".." always refers to the parent directory of the given subdirectory.

Finally, clean up the duplicate files by removing the "second" file and the "sub" subdirectory:

```
% rm sub/second
% rmdir sub
% ls -l
-rw-r--r--  1 username      group          15 May 22 16:26 first
```

This shows that you can refer to a file in a different directory using the relative path name to the file (you can also use the absolute path name to the file - something like "/Users/username/primer/sub/second", depending on your home directory). You can also include the ".." within the path name (for instance, you could have referred to the file as "../primer/sub/second").

1. Other Useful Commands

The current date and time are printed with the following command:

```
% date
Thu May 22 17:39:04 CDT 2003
```

Remote login to another machine can be accomplished using the "ssh" command:

```
% ssh -l myname host
```

where "myname" will be your *username* on the remote system (possibly identical to your *username* on this system) and "host" is the name (or IP address) of the machine you are logging into. Note that

"ssh" is secure unlike older programs such as "telnet".

Transfer files between machines using "scp". For example, to copy file "myfile" from the remote machine named "host", the command would be:

```
% scp myname@host:myfile .
```

(The "." refers to your current working directory, meaning that the destination for "myfile" is your current directory.)

If you are running X11, you should be able to start some applications from the command line, for example:

```
% matlab &
```

(The '&' character tells the console to run Matlab in the background - this way you immediately get a new prompt without first having to quit Matlab.)

In OS X, you can open file using the command "open". This command will open the file with the application corresponding to the file type. For example

```
% open README.pdf
```

will open the file README.pdf using Preview. You can also open applications using the -a flag

```
% open -a Safari
```

or open a file usingTextEdit with the -e flag

```
% open -e mytext.txt
```

1. Online Help

You can get online help from the "man" pages ("man" is short for "manual"). The information is terse, but generally comprehensive, so it provides a nice reminder if you have forgotten the syntax of a particular command or need to know the full list of options.

Use the "-k" option to provide a list of commands that pertain to a particular topic. For instance, try:

```
% man -k "copy files"
```

(and make sure to include the quotation marks). One of the commands listed should be the "cp" file copy command. Display the man page for "cp" by typing:

```
% man cp
```

1. Logging Out

It is very important to log out of your account whenever you are done using it, especially if you are on a public machine.

To close a shell window in a graphical environment, you can type:

```
% exit
```

Logging out from a graphical environment will require clicking on the appropriate icon. Logging out of a remote session can be done by either using the "exit" command or by typing:

```
% logout
```

1. Summary Of Basic Shell Commands

% pico myfile	<i>text edit file "myfile"</i>
% ls	<i>list files in current directory</i>
% ls -l	<i>long format listing</i>
% cat myfile	<i>view contents of text file "myfile"</i>
% more myfile	<i>paged viewing of text file "myfile"</i>
% less myfile	<i>scroll through text file "myfile"</i>
% cp srcfile destfile	<i>copy file "srcfile" to new file "destfile"</i>
% mv oldname newname	<i>rename (or move) file "oldname" to "newname"</i>
% rm myfile	<i>remove file "myfile"</i>
% mkdir subdir	<i>make new directory "subdir"</i>
% cd subdir	<i>change current working directory to "subdir"</i>
% rmdir subdir	<i>remove (empty) directory "subdir"</i>
% pwd	<i>display current working directory</i>
% date	<i>display current date and time of day</i>
% ssh -l myname host	<i>remote shell login of username "myname" to "host"</i>
% scp myname@host:myfile .	<i>remote copy of file "myfile" to current directory</i>
% netscape &	<i>start Netscape web browser (in background)</i>
% man -k "topic"	<i>search manual pages for "topic"</i>
% man command	<i>display man page for "command"</i>
% exit	<i>exit a terminal window</i>
% logout	<i>logout of a console session</i>

WEEK 2 READINGS

Note: Week 2 readings in the Bioinformatics Reader refer to the “Unix and Perl Primer for Biologists, which was too long to include right here. The Introductory section and Part 1 of this primer deals with much of the same material we covered in class. You may want to look at it to review or to cement your understanding. You can find this material in the appendix of this reader (pages 183 - 229).

Unix and Perl Primer for Biologists

Keith Bradnam & Ian Korf

Version 3.1.1 – October 2012

Unix and Perl Primer for Biologists by Keith Bradnam & Ian Korf is licensed under a Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 License. Please send feedback, questions, money, or abuse to krbradnam@ucdavis.edu or ifkorf@ucdavis.edu. (c) 2012, all rights reserved.



Contents

- [Shameless plug](#)
- [Introduction](#)
- [Preamble](#)
- [First steps](#)
- [Part 1 — Unix - Learning the essentials](#)
- [Part 2 — Advanced Unix](#)
- [Part 3 — Perl](#)
 - [Project 0 — Poisson](#)
 - [Project 1 — DNA composition](#)
 - [Project 2 — Descriptive statistics](#)
 - [Project 3 — Sequence shuffler](#)
 - [Project 4 — The name game](#)
 - [Project 5 — K-mer analysis](#)
 - [Project 6 — Codon usage of a GenBank file](#)
 - [Project 7 — Useful functions](#)
- [Troubleshooting — Troubleshooting guide](#)
- [Common errors — Table of common error messages](#)
- [Version history — Version history of this document](#)

WEEK 3 READINGS

Review

PubMed and beyond: a survey of web tools for searching biomedical literature

Zhiyong Lu*

National Center for Biotechnology Information (NCBI), National Library of Medicine, Bethesda, MD 20894, USA

*Corresponding author: Tel: 301-594-7089; Fax: 301-480-2288; Email: zhiyong.lu@nih.gov

Submitted 21 June 2010; Revised 6 December 2010; Accepted 7 December 2010

The past decade has witnessed the modern advances of high-throughput technology and rapid growth of research capacity in producing large-scale biological data, both of which were concomitant with an exponential growth of biomedical literature. This wealth of scholarly knowledge is of significant importance for researchers in making scientific discoveries and healthcare professionals in managing health-related matters. However, the acquisition of such information is becoming increasingly difficult due to its large volume and rapid growth. In response, the National Center for Biotechnology Information (NCBI) is continuously making changes to its PubMed Web service for improvement. Meanwhile, different entities have devoted themselves to developing Web tools for helping users quickly and efficiently search and retrieve relevant publications. These practices, together with maturity in the field of text mining, have led to an increase in the number and quality of various Web tools that provide comparable literature search service to PubMed. In this study, we review 28 such tools, highlight their respective innovations, compare them to the PubMed system and one another, and discuss directions for future development. Furthermore, we have built a website dedicated to tracking existing systems and future advances in the field of biomedical literature search. Taken together, our work serves information seekers in choosing tools for their needs and service providers and developers in keeping current in the field.

Database URL: <http://www.ncbi.nlm.nih.gov/CBBresearch/Lu/search>

Introduction and background

Literature search refers to the process in which people use tools to search for literature relevant to their individual needs. In the context of this review, tools are Web-based online systems; literature is limited to the biomedical domain; and typical user information needs include, but are not limited to, finding the bibliographic information about a specific article, or searching for publications pertinent to a specific topic (e.g. a disease). With the ease of Internet access, the amount of biomedical literature in electronic format is on the rise. As a matter of fact, as pointed out in previous work and shown in Figure 1, the size of the biome has grown exponentially over the past few years (1). As of 2010, there are over 20-million citations indexed through PubMed, a free Web literature search service developed and maintained by the National Center for

Biotechnology Information (NCBI). PubMed is as part of NCBI's Entrez retrieval system that provides access to a diverse set of 38 databases (2). PubMed currently includes citations and abstracts from over 5000 life science journals for biomedical articles back to 1948. Since its inception, PubMed has served as the primary tool for electronically searching and retrieving biomedical literature. Millions of queries are issued each day by users around the globe (3), who rely on such access to keep abreast of the state of the art and make discoveries in their own fields.

Although PubMed provides a broad, up-to-date and efficient search interface, it has become more and more challenging for its users to quickly identify information relevant to their individual needs, owing mainly to the ever-growing biomedical literature. As a result, users are often overwhelmed by the long list of search results: over one-third of PubMed queries result in 100 or more citations (3).

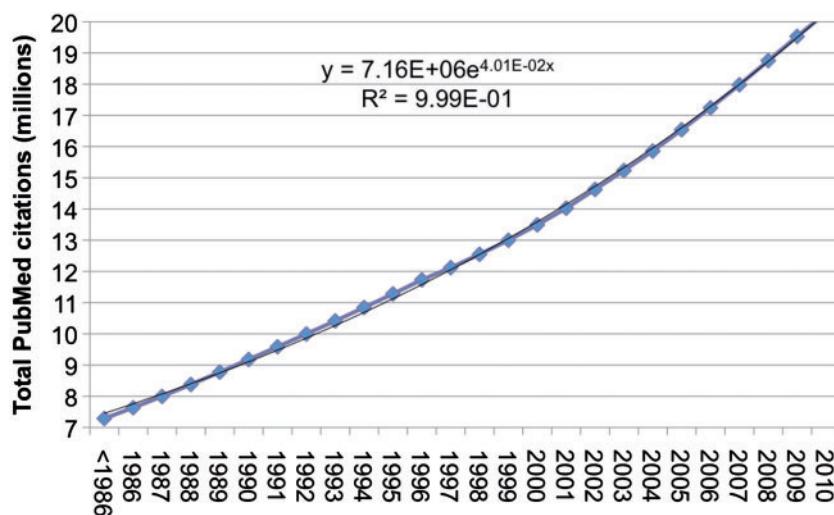


Figure 1. Growth of PubMed citations from 1986 to 2010. Over the past 20 years, the total number of citations in PubMed has increased at a ~4% growth rate. There are currently over 20-million citations in PubMed. 2010 is partial data (through December 1).

In response to such a problem of information overload, the NCBI has made efforts (see detailed discussion in 'Changes to PubMed and looking into the future' section) in enhancing standard PubMed searches by suggesting more specific queries (4). At the same time, the free availability of MEDLINE data and Entrez Programming Utilities (2) make it possible for external entities—from either academia or industry—to create alternative Web tools that are complementary to PubMed.

We present herein a list of 28 such systems, group them by their unique features, compare their differences (with PubMed and one another), and highlight their individual innovations. First and foremost, we aim to provide general readers an overview of PubMed and its recent development, as well as short summaries for other comparable systems that are freely accessible from the Internet. The second objective is to provide researchers, developers and service providers a summary of innovative aspects in recently developed systems, as well as a comparison of different systems. Finally, we have developed a website that is dedicated to online biomedical literature search systems. In addition to the systems discussed in this article, we will keep it updated with new systems so that readers can always be informed of the most current advances in the field.

We believe this work represents the most comprehensive review of systems for seeking information in biomedical literature to date. Unlike many other review articles on text-mining systems (5–11), we limited our focus exclusively to systems that are: (i) for biomedical literature search and (ii) comparable to the PubMed system. The most comparable work is an earlier survey of 18 tools in 2008 (12). However, our review is significantly different in several

major aspects. First, the majority of the systems (19/28) in our review were not previously discussed due to different selection criteria or emergence since 2008. Second, we use different classification criteria for categorizing and comparing systems so readers can find discussion from different perspectives. Third, we provide a more detailed overview of each system and its unique features. In particular, we describe PubMed and its recent development in greater detail based on our own experience. Lastly, we have built a website with links to existing systems and mechanisms for registering future systems. All together, our work complements the previous survey, and more importantly it provides one-stop shopping for biomedical literature search systems.

PubMed: the primary tool for searching biomedical literature

Contents and intended audience

PubMed's intended users include researchers, healthcare professionals and the general public, who either have a need for some specific articles (e.g. search with an article title) or more generally, they search for the most relevant articles pertaining to their individual interests (e.g. information about a disease). A general workflow of how users interact with PubMed is displayed in Figure 2: a user queries PubMed or other similar systems for a particular biomedical information need. Offered a set of retrieved documents, the user can browse the result set and subsequently click to view abstracts or full-text articles, issue a new query, or abandon the current search.

From a search perspective, PubMed takes as input natural language, free-text keywords and returns a list of

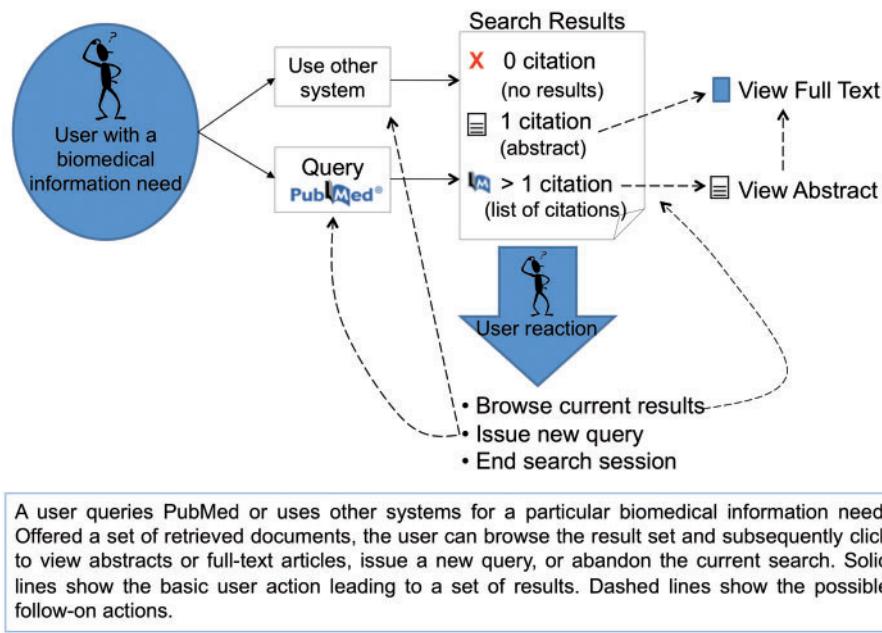


Figure 2. Overview of general user interactions with PubMed (or similar systems) for searching biomedical literature. Adapted from Islamaj Dogan *et al.*, (3).

citations that match input keywords (PubMed ignores stop-words). Its search strategy has two major characteristics: first, by default it adds Boolean operators into user queries and uses automatic term mapping (ATM). Specifically, the Boolean operator 'And' is inserted between multi-term user queries to require retrieved documents to contain all the user keywords. For example, if a user issued the query 'pubmed search', the Boolean operator 'AND' would be automatically inserted between the two words as 'pubmed AND search'.

In addition, PubMed automatically compares and maps keywords from a user query to lists of pre-indexed terms (e.g. Medical Subject Headings MeSH®) through its ATM process (http://www.nlm.nih.gov/pubs/techbull/mj08/mj08_pubmed_atm_cite_sensor.html; 13). That is, if a user query can be mapped to one or more MeSH concepts, PubMed will automatically add its MeSH term(s) to the original query. As a result, in addition to retrieving documents containing the query terms, PubMed also retrieves documents indexed with those MeSH terms. Take the earlier example 'pubmed search' for illustration, because the word 'pubmed' can be mapped to MeSH so the final executed search is ['pubmed' (MeSH terms) or 'pubmed' (all fields)] and 'search' (all fields) where the PubMed search tags (all fields) and (MeSH terms) indicate the preceding word will be searched in all indexed fields or only the MeSH indexing field, respectively.

The second major uniqueness of PubMed is its choice for ranking and displaying search results in reverse

chronological order. More specifically, PubMed returns matched citations in the time sequence of when they were first entered in PubMed by default. This date is formally termed as the Entrez Date (EDAT) in PubMed.

Other tools comparable to PubMed

Standards for selecting comparable systems

In this work, we selected systems for review based on the following three criteria. First, they should be Web-based and operate on equivalent or similar content as PubMed. Systems that are designed to search beyond abstract, such as full text (e.g. PubMed Central; Google Scholar) or figure/tables [e.g. BioText (14); Yale image finder (15)] are thus not included for consideration in this work. Moreover, we focus on tools developed specifically for the biomedical domain. Hence, some general Web-based services such as Google Scholar are excluded in the discussion. Second, a system should be capable of searching an arbitrary topic in the biomedical literature as opposed to some limited areas. Although most citations in PubMed are of biologically relevant subjects (e.g. gene or disease), the topics in the entire biomedical literature are of a much broader coverage. For example, it includes a number of interdisciplinary subjects such as bioinformatics. In other words, the proposed system needs to be developed generally enough so that different kinds of topics can be searched. Third, the online Web system should require no installation or subscription fee (i.e. freely accessible),

which would allow the users to readily experience the service. By these three standards, a total of 28 qualified systems were found and they are listed in Tables 1 and 2 below. Moreover, we classified them into four categories depending on the best match between their most notable features and the category theme. Note that some systems may have features belonging to multiple groups and that within each group, we list systems in reverse chronological order. In Table 1, we show the year when a system was first introduced and highlight major features that distinguish different systems from the technology development perspective. In Table 2,

we compare a set of features that affect the value and utility of different tools from a user perspective. For instance, we report the last content update time for each system as most users would like to keep informed with the latest publications. Specifically, we used the PubMed content as the study control and searched for the latest PubMed citation (PMID: 20726112 on 23 August 2010) in all the systems during comparison. When the citation can be found in a system, we consider its content as 'current' with PubMed. Otherwise, either an exact date (if such information is provided at the Website) or approximate year is labeled.

Table 1. PubMed derivatives are grouped according to their most notable features

Systems	Year	Major features
Ranking search results		
RefMed	2010	Featuring multi-level relevance feedback for ranking
Quertle	2009	Allowing searches with concept categories
MedlineRanker	2009	Finding relevant documents through classification
MiSearch	2009	Using implicit feedback for improving ranking
Hakia	2008	Powered by Hakia's proprietary semantic search technology
SemanticMEDLINE	2008	Powered by cognition's proprietary search technology
MSscanner	2008	Finding relevant documents through classification
eTBLAST	2007	Finding documents similar to input text
PubFocus	2006	Sorting by impact factor and citation volume
Twease	2005	Query expansion with relevance ranking technique
Clustering results into topics		
Anne O'Tate	2008	Clustering by important words, topics, journals, authors, etc.
McSyBi	2007	Clustering by MeSH or UMLS concepts
GoPubMed	2005	Clustering by MeSH or GO terms
ClusterMed	2004	Clustering by MeSH, title/abstract, author, affiliation, or date
XplorMed	2001	Clustering by extracted keywords from abstracts
Extracting and displaying semantics and relations		
MedEvi	2008	Providing textual evidence of semantic relations in output
EBIMed	2007	Displaying proteins, GO annotations, drugs and species
CiteXplore	2006	EBI's tool for integrating biomedical literature and data
MEDIE	2006	Extracting text fragments matching queried semantics
PubNet	2005	Visualizing literature-derived network of bio-entities
Improving search interface and retrieval experience		
iPubMed	2010	Allow fuzzy search and approximate match
PubGet	2007	Retrieving results in PDFs
BabelMeSH	2006	Multi-language search interface
HubMed	2006	Export data in multiple format; visualization; etc
askMEDLINE	2005	Converting questions into formulated search as PICO
SLIM	2005	Slider interface for PubMed searches
PICO	2004	Search with patient, intervention, comparison, outcome
PubCrawler	1999	Alerting users with new articles based on saved searches

Within each group, systems are sorted in reverse chronological order.

Table 2. Comparison of system features

Systems	Content last update	Service provider profile	Source code available	System output format	PubMed ID links	Full-text links	Related article links	Export search results
RefMed	2010	Academic	×	List	✓	×	×	×
Quertle	2010	Private	×	List	✓	✓	×	✓
MedlineRanker	Current	Academic	×	List	✓	×	×	×
MiSearch	Current	Academic	×	List	✓	×	×	×
Hakia	2010	Private	×	List	✓	×	×	×
SemanticMEDLINE	8 June 2010	Private	×	List	✓	×	×	×
MScanner	2007	Academic	✓	List	✓	×	×	×
eTBLAST	2010	Academic	×	List	✓	×	×	×
PubFocus	Current	Private	×	List	×	×	×	×
Twease	Current	Academic	✓	List	✓	×	✓	×
Anne O'Tate	Current	Academic	×	List	✓	×	✓	×
McSyBi	Current	Academic	×	List	✓	×	×	×
GoPubMed	Current	Private	×	List	✓	✓	✓	✓
ClusterMed	Current	Private	×	List	✓	×	×	✓
XplorMed	Current	Academic	×	List	✓	×	×	×
MedEvi	2010	Govn't	×	Table	✓	×	×	×
EBIMed	2010	Govn't	×	Table	✓	×	×	×
CiteXplore	Current	Govn't	×	List	✓	✓	×	✓
MEDIE	12 October 2009	Academic	×	List	✓	×	×	×
PubNet	Current	Academic	×	Graph	✓	×	×	✓
iPubMed	Current	Academic	×	List	✓	×	×	×
PubGet	Current	Private	×	List	✓	✓	×	✓
BabelMeSH	2010	Govn't	×	List	✓	✓	×	×
HubMed	Current	Private	×	List	✓	✓	✓	✓
askMEDLINE	2010	Govn't	×	List	✓	✓	✓	×
SLIM	Current	Govn't	×	List	✓	✓	✓	×
PICO	Current	Govn't	×	List	✓	✓	✓	×
PubCrawler	Current	Academic	×	List	✓	×	✓	✓

Tools are listed in the same order as they appear in Table 1. PubMed was used as the study control (assessed on 23 August 2010) for content last update (i.e. current means its content is current with the PubMed content). Latest year information was used when no exact date can be determined. Symbol ✓ stands for yes, and × for no. Govn't, government.

Based on the content of both tables, we have the following observations:

- (1) The majority (16/28) of systems contains either 'Pub' or 'Med' in their name, indicating their strong bond to the PubMed system.
- (2) All reviewed systems have been developed continuously during the past 10 or so years, starting from the introduction of PubCrawler in 1999 to iPubMed, the newest member in 2010. It is roughly the same period of time that a significant advance and maturity take place in the fields of text mining and Web technology. Many novel techniques in those two fields (e.g. named entity recognition techniques) were

driving forces in the development of various systems reviewed in this work.

- (3) Most systems were developed by academics researchers. Yet, several systems also came from the private sector (i.e. Hakia, Cognition, ClusterMed, Quertle) or the public sector (e.g. CiteXplore from the European Bioinformatics Institute). In addition to free access (a requirement for all the systems), the source code of two academic systems (MScanner and Twease) are freely available at their websites under the GNU General Public License.
- (4) Similar to the general Web search engines such as Google, the presentation of search results in the

- reviewed tools is primarily list based. For some systems that perform result clustering, the list can be further grouped into different topics. Other output formats include tabular and graph presentations, which are designed for systems that are able to extract and display semantic relations.
- (5) Although only few systems offer links to full-text and related articles, and allow export to bibliographic management software after searches (desirable functions in literature search), one can always (except in one system) follow the PubMed link to use those utilities.
- (6) When comparing the four different development themes, improving ranking and the user interface seem to be the more popular directions. In the following sections, we describe each of the 28 systems in greater detail.
- ### Ranking search results
- PubMed returns search results in reverse chronological order by default. In other words, most recent publications are always returned first. Although returning results by time order has its own advantages, several systems are devoted to seeking alternative strategies in ranking results.
- RefMed (16) is a recent development based on both machine learning and information retrieval (IR) techniques. It first retrieves search results based on user queries. Next, it asks for explicit user feedback on relevant documents and uses such information to learn a ranking function by a so-called learning-to-rank algorithm RankSVM (17,18). Subsequently, the learned function ranks retrieval results by relevance in the next iteration.
 - Quertle (19) is a recent biomedical literature search engine developed by a for-profit private enterprise. Its core concept recognition features allow the users to incorporate concept categories into their searches. For instance, one of their concept categories represents all protein names, thus users can search all specific proteins as a whole. It is also claimed that they extract relationships based on the context for improving text retrieval. However, its details are not clearly described to the public.
 - MedlineRanker (20) takes as input a set of documents relating to a certain topic, and automatically learns a list of most discriminative words representing that topic based on a Naïve Bayes classifier. Then it can use the learned words to score and rank newly published articles pertaining to the topic.
 - MiSearch (21) is an online tool that ranks citations by using implicit relevance feedback (22). Unlike RefMed, it uses user clickthrough history as implicit feedback for identifying terms relevant to user's information need in the form of log likelihood ratios. MEDLINE citations that contain a larger number of such relevant terms would be ranked higher than those with a lesser number of such terms. In their implicit relevance feedback model, they also take the recency effect into consideration.
 - Hikia (23) offers access to more than 10-million MEDLINE citations through pubmed.hakia.com. Because it is a product of a private company, it is unclear which ranking algorithm is employed in their system, except that it is said of some kind of semantic search technology.
 - Semantic MEDLINETM (24) was built based on CognitionSearchTM, a system developed by Cognition's proprietary Semantic NLPTM technology, which incorporates word and phrase knowledge for understanding the semantic meaning of the English language. The Semantic MEDLINE system adds specific vocabularies from biomedicine in order to better understand the domain specific language. Like Hikia, details are not revealed to the public.
 - MScanner (25) is mostly comparable to MedlineRanker in terms of its functionality. The major difference is that it uses MEDLINE annotations (MeSH and journal identifiers) instead of words (nouns) in the abstract when doing the classification. As a result, Mscanner is able to process documents faster but it cannot process articles with incomplete or missing annotations.
 - eTBLAST (26) is capable of identifying relevancy by finding documents similar to the input text. Unlike PubMed's related articles (27) that uses summed weights of overlapping words between two documents, eTBLAST determines text similarity based on word alignment. Thus, abstract-length textual input is superior to short queries in obtaining good results.
 - PubFocus (28) sorts articles based on a hybrid of domain specific factors for ranking scientific publications: journal impact factor, volume of forward references, reference dynamics, and authors' contribution level.
 - Twease (29) was built on the classic Okapi BM25 ranking algorithm (30) with twists such that retrieval performance can be maintained when query terms are automatically expanded through the biomedical thesauri or post-indexing stemming.

Clustering results into topics

The common theme of the five systems in the second group is about categorization of search results, aiming for quicker navigation and easier management of large numbers of returned results. Such a technique is developed to respond to the problem of information overload: users are often overwhelmed by a long list of returned documents. As pointed out in ref. (31), this technique is generally shown

to be effective and useful for seeking relevant information from medical journal articles. As discussed in details below, the five systems mainly differ in the manner by which search results are clustered.

- Anne O'Tate (32) post-processes retrieved results from PubMed searches and groups them into one of the pre-defined categories: important words, MeSH topics, affiliations, author names, journals and year of publication. Important words have more frequent occurrences in the result subset than in the MEDLINE as a whole, thus they distinguish the result subset from the rest of MEDLINE. Clicking on a given category name will display all articles in that category. To find a article by multiple categories, one can follow the categories progressively (e.g. first restricting results by year of publication, then by journals).
- McSyBi (33) presents clustered results in two distinct fashions: hierarchical or non-hierarchical. While the former provides an overview of the search results, the latter shows relationships among the search results. Furthermore, it allows users to re-cluster results by imposing either a MeSH term or ULMS Semantic Type of her research interest. Updated clusters are automatically labeled by relevant MeSH terms and by signature terms extracted from title and abstracts.
- GOPubMed (34) was originally designed to leverage the hierarchy in Gene Ontology (GO) to organize search results, thus allowing users to quickly navigate results by GO categories. Recently, it was made capable of sorting results into four top-level categories: what (biomedical concepts), who (author names), where (affiliations and journals) and when (date of publications). In the what category, articles are further sorted according to relevant GO, MeSH or UniProt concepts.
- ClusterMed (35) can cluster results in six different ways: (i) title, abstract and MeSH terms (TiAbMh); (ii) title and abstract (TiAb); (iii) MeSH terms (Mh); (iv) author names (Au); (v) affiliations (Ad) and (vi) date of publication (Dp). For example, when clustering results by TiAbMh, both selected words from title/abstract and MeSH terms are used as filters. Like Hakia, ClusterMed is a proprietary product from a commercial company (Vivisimo) that specializes in enterprise search platforms. Thus, how the filters are selected is not known to the public.
- XplorMed (36) not only organizes results by MeSH classes, it also allows users to explore the subject and words of interest. Specifically, it first returns a coarse level clustering of results using MeSH, offering an opportunity for users to restrict their search to certain categories of interest. Next, the tool displays keywords in the selected abstracts. At this step, users can choose to either go directly to the next step or start a deeper analysis of the displayed subjects. The former would

present chains of closely related keywords, while the latter allows you to explore the relationships between different keywords and their mentions in MEDLINE articles. Finally by selecting one or more chained keywords, the system returns a list of articles ranked by those selected keywords.

Enriching results with semantics and visualization

The five systems in this group aim to analyze search results and present summarized knowledge of semantics (biomedical concepts and their relationships) based on information extraction techniques. They differ in three aspects: (i) the types of biomedical concepts and relations to be extracted; (ii) the computational techniques used for information extraction; and (iii) how they present extraction results.

- MedEvi (37) provides 10 concept variables of major biological entities (e.g. gene) to be used in semantic queries such that the search results are bound to the associated biological entities. Additionally, it also prioritizes search results to return first those citations with matching keywords aligned to the order as they occur in original queries.
- EBIMED (38) extracts proteins, GO annotations, drugs and species from retrieved documents. Relationships between extracted concepts are identified based on co-occurrence analysis. The overall results are presented in table format.
- CiteXplore (39) is a system that combines literature search with text-mining tools in order to provide integrated access to both literature and biological data. In addition to the content of PubMed, it also contains abstract records from patent applications from the European Patent office and from the Shanghai Information Center for Life Sciences, Chinese Academy of Sciences. One other feature of CiteXplore is its inclusion of reference citation information.
- MEDIE (40) provides semantic search in addition to standard keyword search in the format of (subject, verb, object) and returns text fragments (abstract sentences) that match the queried semantic relations. Its output is based on both syntactic and semantic parses of the abstract sentences. For example, a semantic search such as 'what causes colon cancer?' will require the output sentences to match 'cause' and 'colon cancer' as the event verb and object, respectively.
- PubNet (41) stands for Publication Network Graph Utility. It parses the XML output of standard PubMed queries and creates different kinds of networks depending on the type of nodes and edges a user selects. Nodes can be representatives of article, author or some database IDs (e.g. PDB ids) and edges are constructed based on shared authors, MeSH terms or location (articles have identical affiliation zip codes). The graph

networks are drawn with the aid of private visualization software.

Improving search interface and retrieval experience

Systems in this group provide alternative interfaces to the standard PubMed searches. They aim to improve the efficiency of literature search and often take advantage of new Web technologies. They feature novel search/retrieval functions that are currently not available through PubMed, which may be preferred by some users in practice.

- iPubMed (42) provides an interactive search interface: search as you type. When a user types several characters into the search box, the system will instantly show any citations containing that text so that users may narrow their searches. In addition, the system allows minor spelling errors.
- PubGet (43) displays PDFs directly in search results so that users do not have to follow links in PubMed results to PubMed Central or specific journal websites to get PDFs.
- Babelmesh (44) provides an interface so that users can search medical terms and phrases in languages other than English. Currently supported languages include Arabic, Chinese, Dutch, etc. A user's original query is translated into English and then searched for relevant citations.
- HubMed (45) uses Web services to provide various functions ranging from those available in PubMed such as date-sorted search results and automatic term expansion, to new features like relevance-ranked search results; clustering and graphical display of related articles; direct export of citation metadata in many formats; linking of keywords to external sources of information; and manual categorization and storage of interesting articles.
- askMEDLINE (46,47) is designed for handling user queries in the form of questions or complex phrases in the medical setting. It was originally developed as a tool for parsing clinical questions to automatically complete the patient, intervention, comparison, outcome (PICO) form, but was later launched as a tool for the non-expert medical information seeker owing to its ability to retrieve relevant citations from parsed medical terms.
- SLIM (48) is a slider interface for PubMed searches. It features several slide bars to control search limits in a different fashion.
- PICO (49) which stands for patient/problem, intervention, comparison and outcome, is a method used for structuring clinical questions. Its search interface is also available on handhelds.

- PubCrawler (50,51) checks and emails daily updates in MEDLINE to the pre-specified searches saved by the users.

Other honorable mentions

Several other systems are noteworthy even though they are not listed in Table 1 due to failing to meet one or more of our predefined requirements:

- PubMed Assistant (52), Alibaba (53) and PubMed-EX (54) are three non Web-based systems in the PubMed family (disobey selection criterion #1 which requires systems to be Web-based). PubMed assistant belongs to the group of systems for improving usability: it provides useful functions such as keyword highlighting, easy export to citation managers, etc. Both Alibaba and PubMed-EX are geared towards semantic enrichment by identifying gene/protein, disease and other biomedical entities from the text. In addition, Alibaba also presents co-occurrence results in a graph.
- iHop (55), Chilibot (56), PolySearch (57) and Semedico (58) are four representative systems that focus on mining associations between special topics (disobey selection criterion #2 which requires systems to handle general topics). iHop and Chilibot limit their mining to identifying genes and proteins in MEDLINE sentences, while PolySearch supports search over a much broader classes (e.g. diseases). Semidico currently indexes only articles in molecular biology (a sub-area in biomedicine); it mines various biomedical concepts (e.g. gene/protein names) from retrieved documents for enabling faceted navigation. Authority (59) is another example of specialized systems. It uses statistical methods to disambiguate author names, thus making it possible for finding articles written by individual authors.
- To improve biomedical literature search, other systems such as PubFinder (60) ReleMed (61), MedMiner (62) and PubClust (63,64) have been proposed. Unfortunately, none of these systems was in service when they were tested on 31 May 2010 (disobey selection criterion #3). PubFinder is like MScanner and MedlineRanker in that it was designed to rank documents by relevancy based on an input set of topic-specific documents. Based on the selected abstracts, a list of words pertinent to the topic is automatically calculated, which is subsequently used in selecting documents belonging to the defined topic. Unlike MScanner or MedlineRanker, it finds informative words based on their occurrences in the input and reference set. ReleMed, recently proposed by Siadaty et al. (61), uses sentence-level co-occurrence as a surrogate for the existence of relationships between query words. MedMiner proposes to filter and organize the large amount of search results returned by PubMed,

similar to the idea of categorizing search results. Similarly, PubClust was developed on the basis of self-organizing maps (65) to cluster retrieved abstracts in a hierarchical fashion.

Use cases beyond typical PubMed searches

Based on the novel features in each system described above, we show in Figure 3 a list of specific use scenarios that are beyond typical searches in PubMed. Specifically, we first identified a diverse set of 12 use cases, to each of which we further attached applicable systems accordingly. For instance, one can use tools surveyed in this work to search for experts on a specific topic or to visualize search results in networks. Although traditionally PubMed can not meet many of the listed special user needs, its recent development allowed it to perform certain tasks such as identifying similar publications, alerting users with updates and providing feedback in query refinement. More details are presented in 'Changes to PubMed and looking into the future' section.

Discussions on new features

Comparing the 28 systems to PubMed and each other, we see novel proposals for mainly three areas: searching, results analysis and interface/usability.

Searching

Since most users only examine a few returned results on the first result page [Figure 7 in ref. (3)], it is unquestionable that displaying citations by relevance is a desired feature in literature search. The 10 systems listed in 'Ranking search results' section differed with PubMed in this regard. Although most of those systems take as input user keywords, they differ from each other on how they process the keywords and subsequently use them to retrieve relevant citations. Like PubMed's ATM, Twease also has its own query expansion component where additional MeSH terms and others can be added to the original user keywords. This technique can typically boost recall and is especially useful when the original query retrieves few or zero results (13). On the other hand, other systems listed in 'Ranking search results' section are mostly aim for improved precision over PubMed's default reverse time sorting scheme. Their

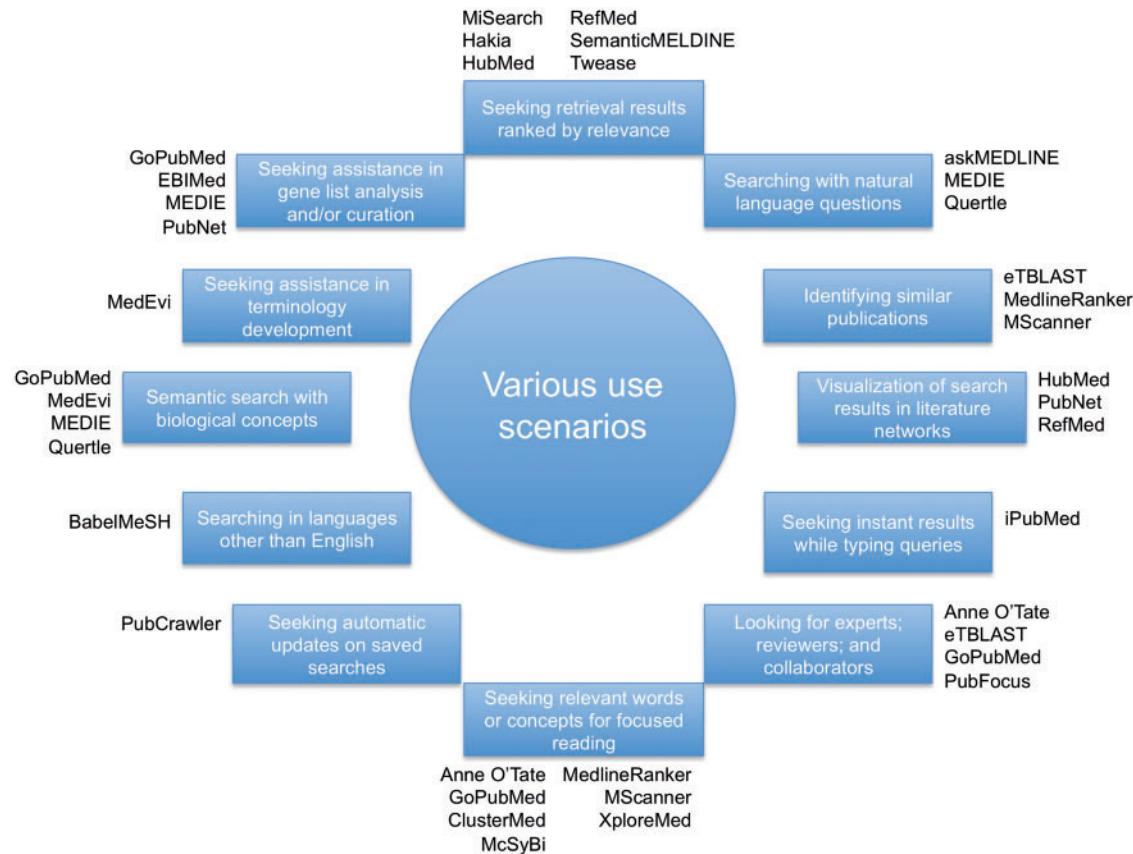


Figure 3. A diverse set of use cases in which different tools may be used.

ranking strategies are very different from one another, ranging from traditional IR techniques like explicit/implicit feedback (RefMed/MiSearch) and relevance ranking (Twease), to utilizing domain specific importance factors like journal impact factors and citation numbers (PubFocus), to some unknown proprietary semantic NLP technologies (Hikia and SemanticSearch).

Results analysis

By default, PubMed returns 20 search results in a page and displays the title, abstract and other bibliographic information when a result is clicked. Recent studies focus on two kinds of extensions to the standard PubMed output. First, because a PubMed search typically results in a long list of citations for manual inspection, systems mentioned in 'Clustering results into topics' section aim to provide an aid with a short list of major topics summarized from the retrieved articles. Thus, users can navigate and choose to focus on the subjects of interest. This is similar to building filters for the result set (66). In this regard, choosing appropriate topic terms to cluster search results into meaningful groups is the key to the success of such approaches. Currently, most systems rely on selecting either important words from title/abstract or terms from biomedical controlled vocabularies/ontologies (e.g. MeSH) as representative topic terms.

The second extension to the standard PubMed output is due to the advances in text-mining techniques. In particular, semantic annotation is believed to be one of the probable cornerstones in future scientific publishing (67) despite the fact that its full benefits are yet to be determined. Thus with the development and maturity of techniques in named entity recognition and biomedical information extraction, some systems present summarized results of deep semantic enrichment. Existing systems ('Enriching results with semantics and visualization' section) have mostly focused on finding genes, proteins, drugs, diseases and species in free text and their biological relationships such as protein–protein interactions. Problems in these areas have received the most attention in the text mining community (68,69).

Interface and usability

In addition to providing improved search quality, a number of systems strive to provide a better search interface, including various changes to input and output. An innovative feature in iPubMed is 'search-as-you-type', thus enabling users to dynamically choose queries while inspecting retrieved results. Other proposals for an alternative input interfaces facilitate user-specific questions (PICO, askMedline), allow non-English queries (BabelMeSH), and promote use of sliders to set limits (SLIM). With respect to changes to output, there are two major directions. First, two systems employ additional components to make

summarized results visible in graphs (ALiBaba and PubNet). Second, several systems provide easier access to PDFs (PubGet) and external citation mangers (PubMed assistant; HubMed).

Changes to PubMed and looking into the future

In response to the great need and challenge in literature search, PubMed has also gone through a series of significant changes to better serve its users. As shown in Figure 4, many of the recent changes happened during the same time period the 28 reviewed systems were developed. So they may have learned from each other. Indeed, some features were first developed in PubMed (e.g. related articles) while others in third party applications (e.g. email alerts).

A new initiative geared towards promoting scientific discoveries was introduced to PubMed a few years ago. Specifically, by providing global search across NCBI's different databases through the Entrez System (<http://www.ncbi.nlm.nih.gov/gquery/>), users now have integrated access to all the stored information in different databases to know about a biological entity—be it related publications, DNA sequences or protein structures. Furthermore, inter-database links have been established and made obvious in search result pages, making the related data readily accessible between literature and other NCBI's biological databases. For instance, through integrated links originating in PubMed results, users can access information about chemicals in PubChem or protein structures in the Structure database. Another category of discovery components is known as sensors (http://www.ncbi.nlm.nih.gov/pubs/techbull/nd08/nd08_pm_gene_sensor.html; http://www.ncbi.nlm.nih.gov/pubs/techbull/mj08/mj08_pubmed_atm_cite_sensor.html). A sensor detects certain types of search terms and provides access to relevant information other than literature. For instance, PubMed's gene sensor detects gene mentions in user queries and shows links directing users to the associated gene records in Entrez Gene. Although these new additions are specific to PubMed and developed independently, they nevertheless all reflect the idea of semantically enriching the literature with biological data of various kinds, to achieve the goal of more efficient acquisition of knowledge.

With respect to research and retrieval, there are also several noteworthy endeavors in PubMed development although its default sorting schema has been kept intact. First, the related article feature was integrated into PubMed so that users can readily examine similar articles in content. eTBLAST has a similar feature, but as explained earlier, the two systems rely on different techniques for obtaining similar documents. Second, specific tools were added into PubMed for different information needs. For instance, the citation matcher is designed for those who search for specific articles. Another example is clinical queries, an interface designed to serve the specific needs

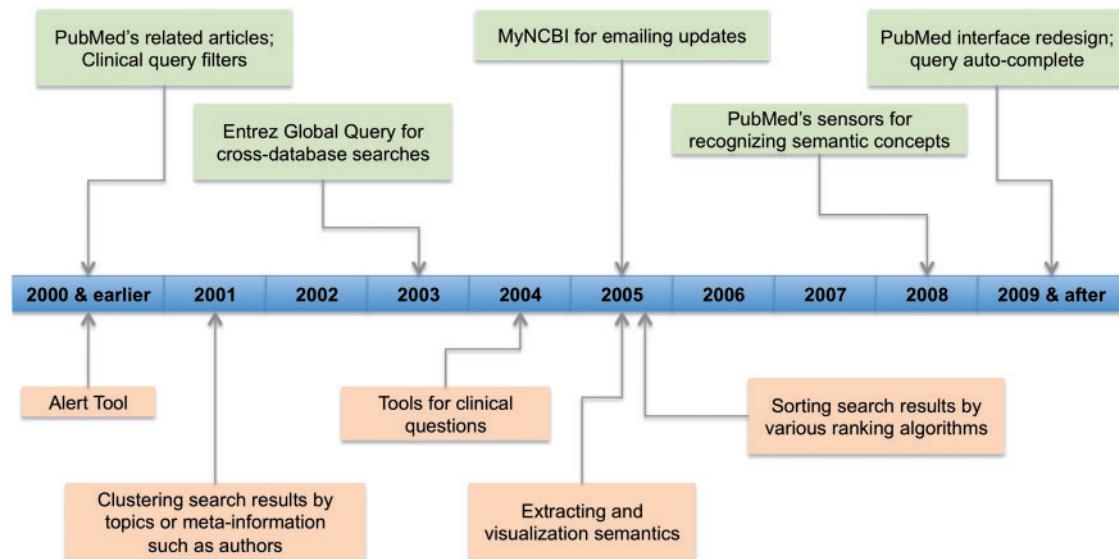


Figure 4. Technology development timeline for PubMed (in light green color) and other biomedical literature search tools (in light orange color). For PubMed, it shows the starting year when various recent changes (limited to those mentioned in 'Changes to PubMed and looking into the future' section) were introduced. For other tools, we show the time period in which tools of various features were first appeared.

of clinicians. It is fundamentally akin to the idea of categorizing search results ('Ranking search results' section) because the tool essentially discards any non-clinical results using a set of predefined filters. Finally, in order to help users avert a long list of return results and narrow their searches, a new feature named 'also try' was recently introduced, which offers query suggestions from the most popular PubMed queries that contain the user search term (4).

Regarding the user interface and usability, the My NCBI tool was introduced to PubMed, which let users select and create filter options, save search results, apply personal preferences like highlighting search terms in results, and share collections of citations. Similar to PubCrawler, it also allows users to set automatic emails for receiving updates of saved searches. Additional search help such as a spell checker and query auto-complete have also been deployed in PubMed. Finally in 2009, the PubMed interface including its homepage was substantially redesigned such that it is now simplified and easier to navigate and use.

Literature search is a fundamentally important problem in research and it will only become harder as the literature grows at a faster speed and broader scope (across the traditional disciplinary boundaries). Therefore we expect continuous developments and new emerging systems in this field. In particular, with the advances in search and Web technologies in general, we are likely to see progress in literature search as well. With the maturity of biomedical text-mining techniques in recognizing biological entities and their relations, better semantic identification and summarization of search results may be achieved, especially for

such entities as author names, disorders, genes/proteins and chemicals/drugs as they are repeatedly and heavily sought topics (3,70) in biomedicine. In addition, one key factor for future system developers is the need to keep their content current with the growth of the literature, as literature search has a recency effect—most users still prefer to be informed of the most current findings in the literature. Finally, to be able to provide one-stop shopping for all 28 reviewed systems plus the ones in the 'Other honorable mentions' section and keep track of future developments in this area, we have built a website at <http://www.ncbi.nlm.nih.gov/CBResearch/Lu/Search>. It contains for every system, a highlight and short description of its unique features, one or more related publications, and a link to the actual system on the Internet. To facilitate busy scientists to quickly find appropriate tools for their specific search needs, we have built a set of search filters. For instance, one can narrow down the entire list of systems to the only ones that keep its content current with PubMed. Future systems will be added to the website either through our quarterly update or by individual request. On the website, we have set up a mechanism for registering future systems. Once we receive such a request, we will curate the necessary information (e.g. system highlights) about the submitted system and make it immediately available at the website.

Conclusions

By our three selection standards, a total of 28 Web systems were included in this review. They are comparable to

PubMed given that they are designed for the same purpose and make use of full or partial PubMed data. We first provided a general description of PubMed including its content and unique characteristics. Next, according to their different features, we classified the 28 systems into four major groups in which we further described each of them in greater detail and showed their differences. Finally we reviewed the 28 systems as a whole and discussed their innovative aspects with respect to searching, result analysis and enrichment, and user interface/usability. This review can directly serve both non-experts and expert users when they wish to find systems other than PubMed. Moreover, the review provides a detailed summary for the recent advances in the field of biomedical literature search. This is particularly useful for existing service providers and anyone interested in future development in the field. Finally the constructed website make an integrated and readily access to all reviewed systems and provides a venue for registering future systems.

Acknowledgements

The author is grateful to the helpful discussion with John Wilbur, Minlie Huang and Natalie Xie.

Funding

Funding for this work and open access charge: Intramural Research Program of the National Institutes of Health, National Library of Medicine.

Conflict of interest: None declared.

References

1. Hunter,L. and Cohen,K.B. (2006) Biomedical language processing: what's beyond PubMed? *Mol. Cell.*, **21**, 589–594.
2. Sayers,E.W., Barrett,T., Benson,D.A. et al. (2010) Database resources of the National Center for Biotechnology Information. *Nucleic Acids Res.*, **38**, D5–D16.
3. Islamaj Dogan,R., Murray,G.C., Neveol,A. et al. (2009) Understanding PubMed user search behavior through log analysis. *Database* doi:10..1093/database/bap018.
4. Lu,Z., Wilbur,W.J., McEntyre,J.R. et al. (2009) Finding query suggestions for PubMed. *AMIA Annu. Symp. Proc.*, **2009**, 396–400.
5. Jensen,L.J., Saric,J. and Bork,P. (2006) Literature mining for the biologist: from information retrieval to biological discovery. *Nat. Rev. Genet.*, **7**, 119–129.
6. Rodriguez-Esteban,R. (2009) Biomedical text mining and its applications. *PLoS Comput. Biol.*, **5**, e1000597.
7. Rzhetsky,A., Seringhaus,M. and Gerstein,M.B. (2009) Getting started in text mining: part two. *PLoS Comput. Biol.*, **5**, e1000411.
8. Krallinger,M., Erhardt,R.A. and Valencia,A. (2005) Text-mining approaches in molecular biology and biomedicine. *Drug Discov. Today*, **10**, 439–445.
9. Krallinger,M., Valencia,A. and Hirschman,L. (2008) Linking genes to literature: text mining, information extraction, and retrieval applications for biology. *Genome Biol.*, **9** (Suppl. 2), S8.
10. Cohen,K.B. and Hunter,L. (2008) Getting started in text mining. *PLoS Comput. Biol.*, **4**, e20.
11. Clegg,A.B. and Shepherd,A.J. (2008) Text mining. *Methods Mol. Biol.*, **453**, 471–491.
12. Kim,J.J. and Rebholz-Schuhmann,D. (2008) Categorization of services for seeking information in biomedical literature: a typology for improvement of practice. *Brief. Bioinform.*, **9**, 452–465.
13. Lu,Z., Kim,W. and Wilbur,W.J. (2009) Evaluation of query expansion using MeSH in PubMed. *Inf. Retr.*, **12**, 69–80.
14. Hearst,M.A., Divoli,A., Guturu,H. et al. (2007) BioText Search Engine: beyond abstract search. *Bioinformatics*, **23**, 2196–2197.
15. Xu,S., McCusker,J. and Krauthammer,M. (2008) Yale Image Finder (YIF): a new search engine for retrieving biomedical images. *Bioinformatics*, **24**, 1968–1970.
16. Yu,H., Kim,T., Oh,J. et al. (2010) Enabling multi-level relevance feedback on PubMed by integrating rank learning into DBMS. *BMC Bioinformatics*, **11** (Suppl. 2), S6.
17. Joachims,T. (2002) Optimizing search engines using clickthrough data. In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* ACM, Edmonton, Alberta, Canada.
18. Liu,T.-Y., Joachims,T., Li,H. et al. (2010) Introduction to special issue on learning to rank for information retrieval. *Inform. Retr.*, **13**, 197–200.
19. Quertle (2009) <http://www.quertle.info> (23 August 2010, date last accessed).
20. Fontaine,J.F., Barbosa-Silva,A., Schaefer,M. et al. (2009) MedlineRanker: flexible ranking of biomedical literature. *Nucleic Acids Res.*, **37**, W141–W146.
21. States,D.J., Ade,A.S., Wright,Z.C. et al. (2009) MiSearch adaptive PubMed search tool. *Bioinformatics*, **25**, 974–976.
22. Crestani,F., Girolami,M., van Rijsbergen,C. et al. (2002) The use of implicit evidence for relevance feedback in web retrieval. In: *Advances in Information Retrieval*, Vol. 2291. Springer, Berlin, Heidelberg, pp. 449–479.
23. Hakia (2008) <http://medical.hakia.com/> (23 August 2010, date last accessed).
24. SemanticMedline (2008) <http://medline.cognition.com/> (23 August 2010, date last accessed).
25. Poulter,G., Poulter,G., Rubin,D. et al. (2008) MScanner: a classifier for retrieving Medline citations. *BMC Bioinformatics*, **9**, 108.
26. Errami,M., Wren,J., Hicks,J. et al. (2007) eTBLAST: a web server to identify expert reviewers, appropriate journals and similar publications. *Nucleic Acids Res.*, **35**, W12.
27. Lin,J. and Wilbur,W.J. (2007) PubMed related articles: a probabilistic topic-based model for content similarity. *BMC Bioinformatics*, **8**, 423.
28. Plikus,M.V., Zhang,Z. and Chuong,C.M. (2006) PubFocus: semantic MEDLINE/PubMed citations analytics through integration of controlled biomedical dictionaries and ranking algorithm. *BMC Bioinformatics*, **7**, 424.
29. Dorff,K., Wood,M. and Campagne,F. (2006) Twease at TREC 2006: Breaking and fixing BM25 scoring with query expansion, a biologically inspired double mutant recovery experiment. *Text REtrieval Conference (TREC) 2006*. NIST Gaithersburg Maryland, USA.

30. Robertson,S.E., Walker,S., Jones,S. et al. (1994) Okapi at TREC-3. *Third Text REtrieval Conference*. NIST Gaithersburg Maryland, USA.
31. Pratt,W. and Fagan,L. (2000) The usefulness of dynamically categorizing search results. *J. Am. Med. Inform. Assoc.*, **7**, 605–617.
32. Smalheiser,N.R., Zhou,W. and Torvik,V.I. (2008) Anne O'Tate: a tool to support user-driven summarization, drill-down and browsing of PubMed search results. *J. Biomed. Discov. Collab.*, **3**, 2.
33. Yamamoto,Y. and Takagi,T. (2007) Biomedical knowledge navigation by literature clustering. *J. Biomed. Inform.*, **40**, 114–130.
34. Doms,A. and Schroeder,M. (2005) GoPubMed: exploring PubMed with the Gene Ontology. *Nucleic Acids Res.*, **33**, W783.
35. ClusterMed (2004) <http://demos.vivisimo.com/clustermed> (23 August 2010, date last accessed).
36. Perez-Iratxeta,C. (2001) XplorMed: a tool for exploring MEDLINE abstracts. *Trends Biochem. Sci.*, **26**, 573–575.
37. Kim,J.J., Pezik,P. and Rebholz-Schuhmann,D. (2008) MedEvi: retrieving textual evidence of relations between biomedical concepts from Medline. *Bioinformatics*, **24**, 1410–1412.
38. Rebholz-Schuhmann,D., Kirsch,H., Arregui,M. et al. (2007) EBIMed-text crunching to gather facts for proteins from Medline. *Bioinformatics*, **23**, e237.
39. CiteXplore (2006) <http://www.ebi.ac.uk/citexplore/> (23 August 2010, date last accessed).
40. Ohta,T., Tsuruoka,Y., Takeuchi,J. et al. (2006) An intelligent search engine and GUI-based efficient MEDLINE search tool based on deep syntactic parsing. In: *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, Sydney, Australia.
41. Douglas,S., Montelione,G. and Gerstein,M. (2005) PubNet: a flexible system for visualizing literature derived networks. *Genome Biol.*, **2008**, **9**, S1.
42. Wang,J., Cetindil,I., Ji,S. et al. (2010) Interactive and fuzzy search: a dynamic way to explore MEDLINE. *Bioinformatics*, **26**, 2321–2327.
43. Pubget (2007) <http://pubget.com/> (23 August 2010, date last accessed).
44. Liu,F., Ackerman,M. and Fontelo,P. (2006) BabelMeSH: development of a cross-language tool for MEDLINE/PubMed. *AMIA Ann. Symp. Proc.*, **2006**, 1012.
45. Eaton,A.D. (2006) HubMed: a web-based biomedical literature search interface. *Nucleic Acids Res.*, **34**, W745–W747.
46. Fontelo,P., Liu,F. and Ackerman,M. (2005) askMEDLINE: a free-text, natural language query tool for MEDLINE/PubMed. *BMC Med. Inform. Decis. Mak.*, **5**, 5.
47. Fontelo,P., Liu,F., Ackerman,M. et al. (2006) askMEDLINE: a report on a year-long experience. *AMIA Ann. Symp. Proc.*, 923.
48. Muin,M., Fontelo,P., Liu,F. et al. (2005) SLIM: an alternative Web interface for MEDLINE/PubMed searches - a preliminary study. *BMC Med. Inform. Decis. Mak.*, **5**, 37.
49. Schardt,C., Adams,M.B., Owens,T. et al. (2007) Utilization of the PICO framework to improve searching PubMed for clinical questions. *BMC Med. Inform. Decis. Mak.*, **7**, 16.
50. Hokamp,K. and Wolfe,K.H. (2004) PubCrawler: keeping up comfortably with PubMed and GenBank. *Nucleic Acids Res.*, **32**, W16–W19.
51. Hokamp,K. and Wolfe,K. (1999) What's new in the library? What's new in GenBank? let PubCrawler tell you. *Trends Genet.*, **15**, 471–472.
52. Ding,J., Hughes,L.M., Berleant,D. et al. (2006) PubMed Assistant: a biologist-friendly interface for enhanced PubMed search. *Bioinformatics*, **22**, 378–380.
53. Plake,C., Schiemann,T., Pankalla,M. et al. (2006) ALIBABA: PubMed as a graph. *Bioinformatics*, **22**, 2444.
54. Tsai,R.T., Dai,H.J., Lai,P.T. et al. (2009) PubMed-EX: a web browser extension to enhance PubMed search with text mining features. *Bioinformatics*, **25**, 3031–3032.
55. Fernandez,J.M., Hoffmann,R. and Valencia,A. (2007) iHOP web services. *Nucleic Acids Res.*, **35**, W21–W26.
56. Chen,H. and Sharp,B.M. (2004) Content-rich biological network constructed by mining PubMed abstracts. *BMC Bioinformatics*, **5**, 147.
57. Cheng,D., Knox,C., Young,N. et al. (2008) PolySearch: a web-based text mining system for extracting relationships between human diseases, genes, mutations, drugs and metabolites. *Nucleic Acids Res.*, **36**, W399–W405.
58. Wermter,J., Tomaneck,K. and Hahn,U. (2009) High-performance gene name normalization with GeNo. *Bioinformatics*, **25**, 815–821.
59. Torvik,V.I. and Smalheiser,N.R. (2009) Author name disambiguation in MEDLINE. *ACM Trans. Knowl. Discov. Data*, **3**, 11:1–11:29.
60. Goetz,T. and Von Der Lieth,C.-W. (2005) PubFinder: a tool for improving retrieval rate of relevant PubMed abstracts. *Nucleic Acids Res.*, **33**, W774.
61. Siadaty,M.S., Shu,J. and Knaus,W.A. (2007) Relemed: sentence-level search engine with relevance score for the MEDLINE database of biomedical articles. *BMC Med. Inform. Decis. Mak.*, **7**, 1.
62. Tanabe,L., Scherf,U., Smith,L.H. et al. (1999) MedMiner: an Internet text-mining tool for biomedical information, with application to gene expression profiling. *Biotechniques*, **27**, 1210–1214, 1216–1217.
63. Fattore,M. and Arrigo,P. (2005) Knowledge discovery and system biology in molecular medicine: an application on neurodegenerative diseases. *In Silico Biol.*, **5**, 199–208.
64. Kolchanov,N., Hofstaedt,R., Milanesi,L. et al. (2006) Topical clustering of biomedical abstracts by self-organizing maps. In: *Bioinformatics of Genome Regulation and Structure II*. Springer, US, pp. 481–490.
65. Lopez-Rubio,E. (2010) Probabilistic self-organizing maps for qualitative data. *Neural Netw.*, **23**, 1208–1225.
66. Kilicoglu,H., Demner-Fushman,D., Rindflesch,T.C. et al. (2009) Towards automatic recognition of scientifically rigorous clinical research evidence. *J. Am. Med. Inform. Assoc.*, **16**, 25–31.
67. Rinaldi,A. (2010) For I dipped into the future. *EMBO Rep.*, **11**, 345–359.
68. Krallinger,M., Leitner,F., Rodriguez-Penagos,C. et al. (2008) Overview of the protein-protein interaction annotation extraction task of BioCreative II. *Genome Biol.*, **9** (Suppl. 2), S4.
69. Morgan,A.A., Lu,Z., Wang,X. et al. (2008) Overview of BioCreative II gene normalization. *Genome Biol.*, **9** (Suppl. 2), S3.
70. Neveol,A., Islamaj-Dogan,R. and Lu,Z. (2010) Semi-automatic semantic annotation of PubMed Queries: a study on quality, efficiency, satisfaction. *J. Biomed. Inform.*



STUDY DESIGNS

A beginner's guide to eukaryotic genome annotation

Mark Yandell and Daniel Ence

Abstract | The falling cost of genome sequencing is having a marked impact on the research community with respect to which genomes are sequenced and how and where they are annotated. Genome annotation projects have generally become small-scale affairs that are often carried out by an individual laboratory. Although annotating a eukaryotic genome assembly is now within the reach of non-experts, it remains a challenging task. Here we provide an overview of the genome annotation process and the available tools and describe some best-practice approaches.

Genome annotation

A term used to describe two distinct processes. 'Structural' genome annotation is the process of identifying genes and their intron–exon structures. 'Functional' genome annotation is the process of attaching meta-data such as gene ontology terms to structural annotations. This Review focuses on structural annotation.

RNA-sequencing data (RNA-seq data). Data sets derived from the shotgun sequencing of a whole transcriptome using next-generation sequencing (NGS) techniques. RNA-seq data are the NGS equivalent of expressed sequence tags generated by the Sanger sequencing method.

Department of Human Genetics, Eccles Institute of Human Genetics, School of Medicine, University of Utah, Salt Lake City, Utah 84112-5330, USA.
Correspondence to M.Y.
e-mail: myandell@genetics.utah.edu
doi:10.1038/nrg3174

Sequencing costs have fallen so dramatically that a single laboratory can now afford to sequence large, even human-sized, genomes. Ironically, although sequencing has become easy, in many ways, genome annotation has become more challenging. Several factors are responsible for this. First, the shorter read lengths of second-generation sequencing platforms mean that current genome assemblies rarely attain the contiguity of the classic shotgun assemblies of the *Drosophila melanogaster*^{1,2} or human genomes^{3,4}. Second, the exotic nature of many recently sequenced genomes also presents annotation challenges, especially for gene finding. Whereas the first generation of genome projects had recourse to large numbers of pre-existing gene models, the contents of today's genomes are often terra incognita. This makes it difficult to train, optimize and configure gene prediction and annotation tools.

A third new challenge is posed by the need to update and merge annotation data sets. RNA-sequencing data (RNA-seq data)^{5–8} provide an obvious means for updating older annotation data sets; however, doing so is not trivial. It is also not straightforward to ascertain whether the result improves on the original annotation. Furthermore, it is not unusual today for multiple groups to annotate the same genome using different annotation procedures. Merging these to produce a consensus annotation data set is a complex task.

Finally, the demographics of genome annotation projects are changing as well. Unlike the massive genome projects of the past, today's genome annotation projects are usually smaller-scale affairs and often involve researchers who have little bioinformatics and computational biology expertise. Eukaryotic genome annotation is not a point-and-click process; however,

with some basic UNIX skills, 'do-it-yourself' genome annotation projects are quite feasible using present-day tools. Here we provide an overview of the eukaryotic genome annotation process, describe the available toolsets and outline some best-practice approaches.

Assembly and annotation: an overview

Assembly. The first step towards the successful annotation of any genome is determining whether its assembly is ready for annotation. Several summary statistics are used to describe the completeness and contiguity of a genome assembly, and by far the most important is N50 (BOX 1). Other useful assembly statistics are the average gap size of a scaffold and the average number of gaps per scaffold (BOX 1). Most current genomes are 'standard draft' assemblies, meaning that they meet minimum standards for submission to public databases⁹. However, a 'high-quality draft' assembly⁹ is a much better target for annotation, as it is at least 90% complete.

Although there are no strict rules, an assembly with an N50 scaffold length that is gene-sized is a decent target for annotation. The reason is simple: if the scaffold N50 is around the median gene length, then ~50% of the genes will be contained on a single scaffold; these complete genes, together with fragments from the rest of the genome, will provide a sizable resource for downstream analyses^{10,11}. As can be seen in FIG. 1, median gene lengths are roughly proportional to genome size. Thus, if the size of the genome of interest is known, it is possible to use this figure to obtain a rough estimate of gene lengths and hence to obtain an estimate of the minimum N50 scaffold length for annotation. CEGMA¹² provides another,

N50

A basic statistic for describing the contiguity of a genome assembly. The longer the N50 is, the better the assembly is. See box 1 for details.

Long interspersed nuclear elements

(LINEs). Retrotransposons that encode reverse transcriptase and that make up a substantial fraction of many eukaryotic genomes.

Short interspersed nuclear elements

(SINEs). Retrotransposons that do not encode reverse transcriptase and that parasitize LINE elements. ALU elements, which are very common in the human genome, are one example of a SINE.

complementary means of estimating the completeness and contiguity of an assembly. This tool screens an assembly against a collection of more or less universal eukaryotic single-copy genes and also determines the percentage of each gene lying on a single scaffold.

Obtaining a high-quality draft assembly is an achievable goal for most genome projects. If an assembly is incomplete or if its N50 scaffold length is too short, we would recommend doing additional shotgun sequencing, as tools are available for the incremental improvement of draft assemblies^{13–15}.

Annotation. Although genome annotation pipelines differ in their details, they share a core set of features. Generally, genome-wide annotation of gene structures is divided into two distinct phases. In the first phase, the ‘computation’ phase, expressed sequence tags (ESTs), proteins, and so on, are aligned to the genome and *ab initio* and/or evidence-driven gene predictions are generated. In the second phase, the ‘annotation’

phase, these data are synthesized into gene annotations (BOX 2). Because this process is intrinsically complicated and involves so many different tools, the programs that assemble compute data (evidence) and use it to create genome annotations are generally referred to as annotation pipelines. Current pipelines are focused on the annotation of protein-coding genes, although Ensembl also has some capabilities for annotating non-coding RNAs (ncRNAs). Tools for annotation of ncRNAs are described in BOX 3.

Step one: the computation phase

Repeat identification. Repeat identification and masking is usually the first step in the computation phase of genome annotation. Somewhat confusingly, the term ‘repeat’ is used to describe two different types of sequences: ‘low-complexity’ sequences, such as homopolymeric runs of nucleotides, as well as transposable (mobile) elements, such as viruses, long interspersed nuclear elements (LINEs) and short interspersed nuclear elements (SINEs)^{16,17}. Eukaryotic genomes can be very repeat rich; for example, 47% of the human genome is thought to consist of repeats¹⁸, and this number is likely to be the lower limit. Also, the borders of these repeats are usually ill-defined; repeats often insert within other repeats, and often only fragments within fragments are present — complete elements are found quite rarely. Repeats complicate genome annotation. They need to be identified and annotated, but the tools used to identify repeats are distinct from those used to identify the genes of the host genome.

Identifying repeats is complicated by the fact that repeats are often poorly conserved; thus, accurate repeat detection usually requires users to create a repeat library for their genome of interest. Available tools for doing so generally fall into two classes: homology-based tools^{19–21} and *de novo* tools^{22–25} (for an overview, see REFS 26,27). Note, however, that *de novo* tools identify repeated sequences — not just mobile elements — so their outputs can include highly conserved protein-coding genes, such as histones and tubulins, as well as transposon sequences. Users must therefore carefully post-process the outputs of these tools to remove protein-coding sequences. These same outputs probably also contain some novel repeat families. Repeats are interesting in and of themselves, and the life cycles and phylogenetic histories of these elements are growing areas of research^{17,28,29}. Adequate repeat annotation should thus be a part of every genome annotation project.

After it has been created, a repeat library can be used in conjunction with a tool such as RepeatMasker³⁰, which uses BLAST^{31–33} and Crossmatch³⁴ to identify stretches of sequence in a target genome that are homologous to known repeats. The term ‘masking’ simply means transforming every nucleotide identified as a repeat to an ‘N’ or, in some cases, to a lower case a, t, g or c — the latter process is known as ‘soft masking’^{32,35}. The masking step signals to downstream sequence alignment and gene prediction tools that these regions are repeats. Failure to mask genome

Box 1 | Common statistics for describing genome assemblies

Genome assemblies are composed of scaffolds and contigs. Contigs are contiguous consensus sequences that are derived from collections of overlapping reads. Scaffolds are ordered and orientated sets of contigs that are linked to one another by mate pairs of sequencing reads.

Scaffold and contig N50s

By far the most widely used statistics for describing the quality of a genome assembly are its scaffold and contig N50s. A contig N50 is calculated by first ordering every contig by length from longest to shortest. Next, starting from the longest contig, the lengths of each contig are summed, until this running sum equals one-half of the total length of all contigs in the assembly. The contig N50 of the assembly is the length of the shortest contig in this list. The scaffold N50 is calculated in the same fashion but uses scaffolds rather than contigs. The longer the scaffold N50 is, the better the assembly is. However, it is important to keep in mind that a poor assembly that has forced unrelated reads and contigs into scaffolds can have an erroneously large N50. Note too that scaffolds and contigs that comprise only a single read or read pair — often termed ‘singletons’ — are frequently excluded from these calculations, as are contigs and scaffolds that are shorter than ~800 bp. The procedures used to calculate N50 may therefore vary between genome projects.

Percent gaps

Another important assembly statistic is its percent gaps. Unsequenced regions between mate pairs in contigs and between scaffolds are often represented as runs of ‘N’s in the final assembly. Thus two assemblies can have identical scaffold N50s but can still differ in their percent gaps: one has very few gaps, and the other is heavily peppered with them. Estimates of gap lengths are often made based on library insert sizes and read lengths; when these are available, the number of ‘N’s in these gaps usually, but not always, represents the most likely estimate of that gap’s size; sometimes, all gaps are simply represented by a run of 50 ‘N’s regardless of their size.

Percent coverage

Percent coverage is used in two senses: genome coverage and gene coverage. The first number, genome coverage, refers to the percentage of the genome that is contained in the assembly based on size estimates; these are usually based on cytological techniques^{16,17}. Genome coverage of 90–95% is generally considered to be good, as most genomes contain a considerable fraction of repetitive regions that are difficult to sequence. So it is not a cause for concern if the genome coverage of an assembly is a bit less than 100%. Gene coverage is the percentage of the genes in the genome that are contained in the assembly. Gene and genome coverage can differ from one another, as hard-to-assemble repetitive regions are often gene-poor. As a result, the percentage gene coverage is often substantially larger than the percentage genome coverage for some difficult-to-assemble genomes.

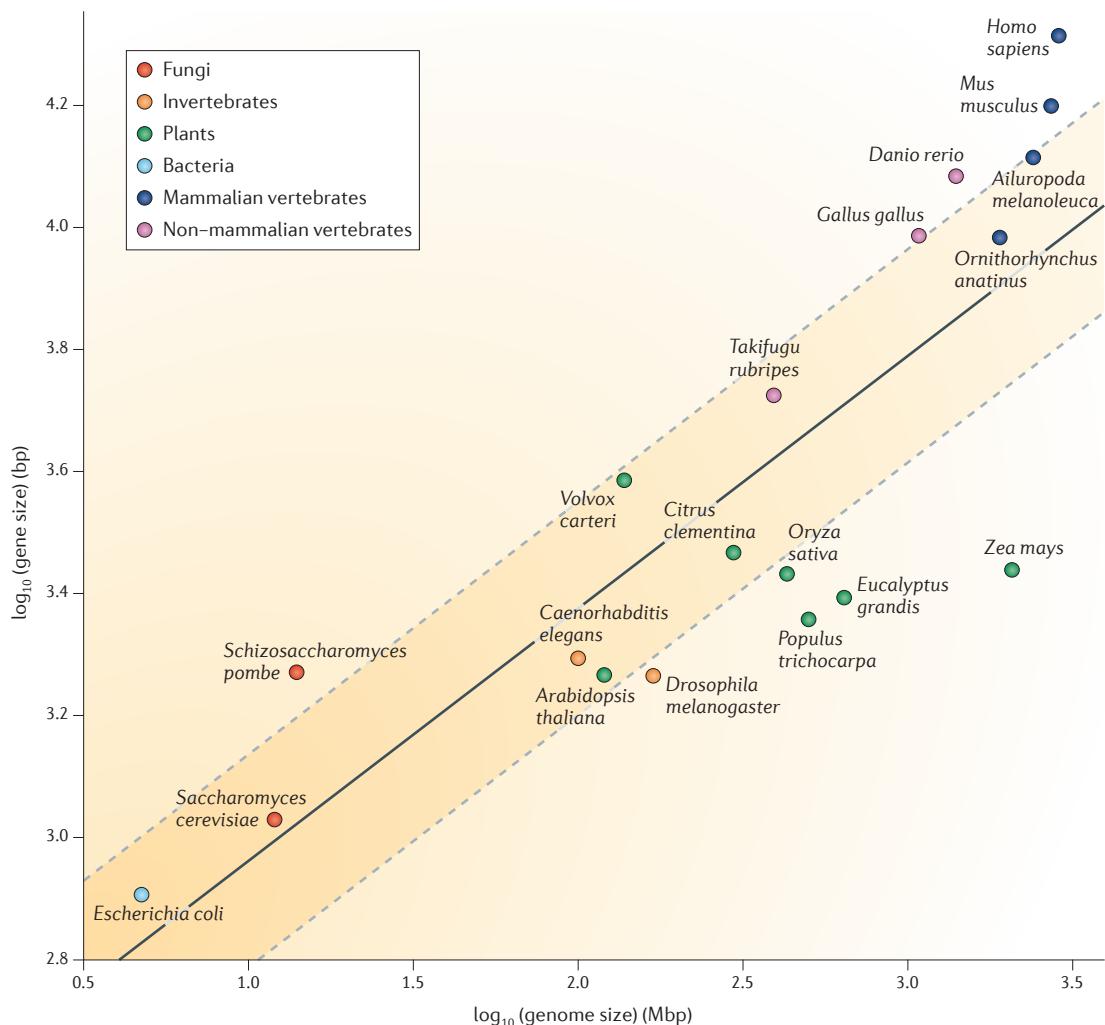


Figure 1 | Genome and gene sizes for a representative set of genomes. Gene size is plotted as a function of genome size for some representative bacteria, fungi, plants and animals. This figure illustrates a simple rule of thumb: in general, bigger genomes have bigger genes. Thus, accurate annotation of a larger genome requires a more contiguous genome assembly in order to avoid splitting genes across scaffolds. Note too that although the human and mouse genomes deviate from the simple linear model shown here, the trend still holds. Their unusually large genes are likely to be a consequence of the mature status of their annotations, which are much more complete as regards annotation of alternatively spliced transcripts and untranslated regions than those of most other genomes.

sequences can be catastrophic. Left unmasked, repeats can seed millions of spurious BLAST alignments³², producing false evidence for gene annotations. Worse still, many transposon open reading frames (ORFs) look like true host genes to gene predictors, causing portions of transposon ORFs to be added as additional exons to gene predictions, completely corrupting the final gene annotations. Good repeat masking is thus crucial for the accurate annotation of protein-coding genes.

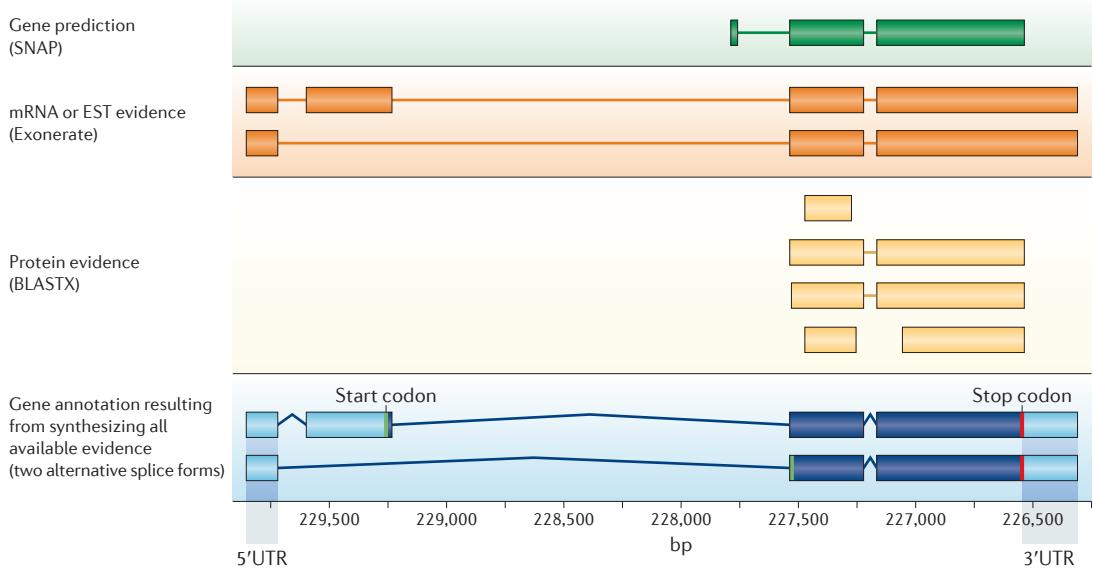
Evidence alignment. After repeat masking, most pipelines align proteins, ESTs and RNA-seq data to the genome assembly. These sequences include previously identified transcripts and proteins from the organism whose genome is being annotated. Sequences from other organisms are also included; generally, these are restricted to proteins, as these retain substantial

sequence similarity over much greater spans of evolutionary time than nucleotide sequences do. In principle, TBLASTX^{31,32,36} can be used to align ESTs and RNA-seq data from phylogenetically distant organisms but, owing to high computational costs, this is only done rarely.

UniProtKB/SwissProt^{37–39} is an excellent core resource for protein sequences. As SwissProt is restricted to highly curated proteins, many users might want to supplement this database with the proteomes of related, previously annotated genomes. One easy way to assemble additional protein and EST data sets is to download sequences from related organisms using the [NCBI taxonomy browser](#)^{40,41}.

EST and protein sequence data sets are often aligned to the genome in a two-tiered process. Frequently, BLAST^{31,32,36} and BLAT⁴² are used to

Box 2 | Gene prediction versus gene annotation



Although the terms 'gene prediction' and 'gene annotation' are often used as if they are synonyms, they are not. With a few exceptions, gene predictors find the single most likely coding sequence (CDS) of a gene and do not report untranslated regions (UTRs) or alternatively spliced variants. Gene prediction is therefore a somewhat misleading term. A more accurate description might be 'canonical CDS prediction'.

Gene annotations, conversely, generally include UTRs, alternative splice isoforms and have attributes such as evidence trails. The figure shows a genome annotation and its associated evidence. Terms in parentheses are the names of commonly used software tools for assembling particular types of evidence. Note that the gene annotation (shown in blue) captures both alternatively spliced forms and the 5' and 3'UTRs suggested by the evidence. By contrast, the gene prediction that is generated by SNAP (shown in green) is incorrect as regards the gene's 5' exons and start-of-translation site and, like most gene-predictors, it predicts only a single transcript with no UTR.

Gene annotation is thus a more complex task than gene prediction. A pipeline for genome annotation must not only deal with heterogeneous types of evidence in the form of the expressed sequence tags (ESTs), RNA-seq data, protein homologies and gene predictions, but it must also synthesize all of these data into coherent gene models and produce an output that describes its results in sufficient detail for these outputs to become suitable inputs to genome browsers and annotation databases.

identify approximate regions of homology rapidly. These alignments are usually filtered to identify and to remove marginal alignments on the basis of metrics such as percent similarity or percent identity. After filtering, the remaining data are sometimes clustered to identify overlapping alignments and predictions. Clustering has two purposes. First, it groups diverse computational results into a single cluster of data, all supporting the same gene. Second, it identifies and purges redundant evidence; highly expressed genes, for example, may be supported by hundreds if not thousands of identical ESTs.

The term 'polishing' is sometimes used to describe the next phase of the alignment process. After clustering, highly similar sequences identified by BLAST and BLAT are realigned to the target genome in order to obtain greater precision at exon boundaries. BLAST, for example, although rapid, has no model for splice sites, and so the edges of its sequence alignments are only rough approximations of exon boundaries⁴³. For this reason, splice-site-aware alignment algorithms, such as *Splign*⁴⁴, *Spidey*⁴⁵, *sim4* (REF. 46) and Exonerate⁴³, are often used to realign matching and highly similar ESTs, mRNAs and proteins to the genomic

input sequence. Although these programs take longer to run, they provide the annotation pipeline with much improved information about splice sites and exon boundaries.

Of all forms of evidence, RNA-seq data have the greatest potential to improve the accuracy of gene annotations, as these data provide copious evidence for better delimitation of exons, splice sites and alternatively spliced exons. However, these data can be difficult to use because of their large size and complexity. The use of RNA-seq data currently lies at the cutting edge of genome annotation, and the available toolset is evolving quickly⁴⁷. Currently, RNA-seq reads are usually handled in two ways. They can be assembled *de novo* — that is, independently of the genome — using tools such as *ABySS*⁴⁸, *SOAPdenovo*⁴⁹ and *Trinity*⁵⁰; the resulting transcripts are then realigned to the genome in the same way as ESTs. Alternatively, the RNA-seq data can be directly aligned to the genome using tools such as *TopHat*⁵¹, *GSNAP*⁵² or *Scripture*⁵³ followed by the assembly of alignments (rather than reads) into transcripts using tools such as *Cufflinks*⁵⁴. See REF. 55 for guidance on the best way to use TopHat with Cufflinks.

Percent similarity

The percent similarity of a sequence alignment refers to the percentage of positive scoring aligned bases or amino acids in a nucleotide or protein alignment, respectively. The term positive scoring refers to the score assigned to the paired nucleotides or amino acids by the scoring matrix that is used to align the sequences.

Percent identity

The percent identity of a sequence alignment refers to the percentage of identical aligned bases or amino acids in a nucleotide or protein alignment, respectively.

Box 3 | Non-coding RNAs

Non-coding RNA (ncRNA) annotation is still in its infancy compared with protein-coding gene annotation, but it is advancing rapidly. The heterogeneity and poorly conserved nature of many ncRNA genes present major challenges for annotation pipelines. Unlike protein-encoding genes, ncRNAs are usually not well-conserved at the primary sequence level; even when they are, nucleotide homologies are not as easily detected as protein homologies, which limits the power of evidence-based approaches.

One common approach is to identify ncRNA genes using conserved secondary structures and motifs. Established examples of these types of tools include [tRNAscan-SE](#)¹¹⁸ and [Snoscan](#)¹¹⁹. MicroRNA (miRNA) gene finders are also available¹²⁰. A more general approach is first to align nucleotide sequences — genomic, RNA-seq and ESTs — from closely related organisms to the target genome and then search these for signs of conserved secondary structures. This is a complex process, however, and can require substantial computational resources; [qRNA](#) is one such tool¹²¹, another is [StemLoc](#)¹²². Be aware that these tools have high false-positive rates. RNA sequencing is also greatly aiding ncRNA identification. For example, miRNAs can be directly identified using specialized RNA preps and sequencing protocols^{123,124}. Even with such sophisticated tools and techniques, distinguishing between bona fide ncRNA genes, spurious transcription and poorly conserved protein-encoding genes that produce small peptides remains difficult, especially in the cases of long intergenic non-coding RNAs (lincRNAs)^{125,126} and expressed pseudogenes^{127,128}.

Another approach is to annotate possible ncRNA genes liberally and then use [Infernal](#)¹²⁹ and [Rfam](#)¹¹⁴ to triage and classify these genes based on primary and secondary sequence similarities. Even with these resources, however, many ncRNAs will remain unclassifiable. Currently, ncRNA annotation is cutting edge, and those using ncRNA annotations should bear in mind that ncRNA annotation accuracies are generally much lower than those of their protein-coding counterparts.

Opinions differ as to the best approach for using RNA-seq data, and the most promising avenue will probably heavily depend on both genome biology (for example, gene density) and the contiguity and completeness of the genome assembly. Gene density is an important consideration. If genes are closely spaced in the genome, then tools such as Cufflinks⁵⁴ sometimes erroneously merge RNA-seq reads from neighbouring genes. In such cases, *de novo* assembly of the RNA-seq data mitigates the problem; in fact, Trinity⁵⁰ is designed to deal with this issue. Several annotation pipelines are now compatible with RNA-seq data: these include [PASA](#)⁵⁶, which uses inchworm⁵⁰ outputs, and [MAKER](#)¹⁰, which can operate directly from Cufflinks⁵⁴ outputs or can use preassembled RNA-seq data.

Ab initio gene prediction. When gene predictors^{57–60} first became available in the 1990s (see REF. 61 for an overview), they revolutionized genome analyses because they provided a fast and easy means to identify genes in assembled DNA sequences. These tools are often referred to as *ab initio* gene predictors because they use mathematical models rather than external evidence (such as EST and protein alignments) to identify genes and to determine their intron–exon structures.

The great advantage of *ab initio* gene predictors for annotation is that, in principle, they need no external evidence to identify a gene or to determine its intron–exon structure. However, these tools have practical limitations from an annotation perspective.

For instance, most gene predictors find the single most likely coding sequence (CDS) and do not report untranslated regions (UTRs) or alternatively spliced transcripts (BOX 2). Training is also an issue; *ab initio* gene predictors use organism-specific genomic traits, such as codon frequencies and distributions of intron–exon lengths, to distinguish genes from intergenic regions and to determine intron–exon structures. Most gene predictors come with precalculated parameter files that contain such information for a few classic genomes, such as *Caenorhabditis elegans*, *D. melanogaster*, *Arabidopsis thaliana*, humans and mice. However, unless your genome is very closely related to an organism for which precompiled parameter files are available, the gene predictor needs to be trained on the genome that is under study, as even closely related organisms can differ with respect to intron lengths, codon usage and GC content⁶².

Given enough training data, the gene-level sensitivity of *ab initio* tools can approach 100%^{63,64} (BOX 4). However, the accuracy of the predicted intron–exon structures is usually much lower, ~60–70%. It is also important to understand that large numbers of pre-existing, high-quality gene models and near base-perfect genome assemblies are usually required to produce highly accurate gene predictions^{63,65}; such data sets are rarely available for newly sequenced genomes.

In principle, alignments of ESTs, RNA-seq and protein sequences to a genome can be used to train gene predictors even in the absence of pre-existing reference gene models. Although many popular gene predictors can be trained in this way, doing so often requires the user to have some basic programming skills. The MAKER pipeline provides a simplified process for training the predictors [Augustus](#)^{66,67} and [SNAP](#)⁶² using the EST, protein and mRNA-seq alignments that MAKER has produced^{10,56}. An alternative is to use GeneMark-ES^{68,69}: a self-training, but sometimes less-accurate, algorithm^{69,70}.

Evidence-driven gene prediction. In recent years, the distinction between *ab initio* prediction and gene annotation has been blurred. Many *ab initio* tools, such as [TwinScan](#)⁷¹, [FGENESH](#)⁷², [Augustus](#), [Gnomon](#)⁷³, [GAZE](#)⁷⁴ and [SNAP](#), can use external evidence to improve the accuracy of their predictions. ESTs, for example, can be used to identify exon boundaries unambiguously. This process is often referred to as evidence-driven (in contrast to *ab initio*) gene prediction. Evidence-driven gene prediction has great potential to improve the quality of gene prediction in newly sequenced genomes, but in practice it can be difficult to use. ESTs and proteins must first be aligned to the genome; RNA-seq data must be aligned too, if they are available. Splice sites must then be identified, and the assembled evidence must be post-processed before a synopsis of these data can be passed to the gene finder. In practice, this is a lot of work, requiring a lot of specialized software. In fact, it is one of the main obstacles that genome annotation pipelines attempt to overcome.

Box 4 | How gene prediction and gene annotation accuracies are calculated

Three commonly used measures of gene-finder performance are sensitivity, specificity and accuracy¹³⁰. Each is measured relative to some standard, usually a reference annotation. Sensitivity (SN) is the fraction of the reference feature that is predicted by the gene predictor. To be more precise, $SN = TP / (TP + FN)$, where TP is true positives and FN is false negatives. By contrast, specificity (SP) is the fraction of the prediction overlapping the reference feature: for example, $SP = TP / (TP + FP)$, where FP is false positives. Note that the definition of SP given here is the one that is commonly used by the gene-finding community¹³⁰ but, more correctly, this measure is positive predictive value (PPV) or precision.

Both measures can be calculated for any portion of a gene model, such as genes, transcripts or exons. At the nucleotide level, TP is the number of exonic nucleotides in the reference gene model, FN is the number of these that are not included in the prediction, and FP is the number of exonic nucleotides in the prediction that are not found in the reference gene model. At the exon level, SN is the number of correct exons in the prediction divided by the number of exons in the reference gene model, and SP is the number of correct exons in the prediction divided by the number of exons in the prediction¹³⁰. So-called 'site measures' are also used: for example, the SN and SP for predicting features such as start codons or splice donors. SN and SP are often combined into a single measure called accuracy (AC): for example, $AC = (SN + SP) / 2$ (see REFS 130–132 for reviews of commonly used accuracy measures).

Panel **A** of the figure shows SN, SP and AC for two different gene models. The reference model is shown in blue, and the two different predictions at the same locus are shown in red. The table on the right gives the values of SN, SP and AC for the two predictions. For the purposes of calculation, exons 1, 2 and 3 of the reference gene model and of prediction 1 have identical start and end coordinates and are 100, 50 and 50 nucleotides long, respectively. In prediction 2, exons are 75 and 50 nucleotides long, respectively, and the start coordinate of its first exon is identical to that of the reference, but its end is not; its second exon is identical to the third exon in the reference.

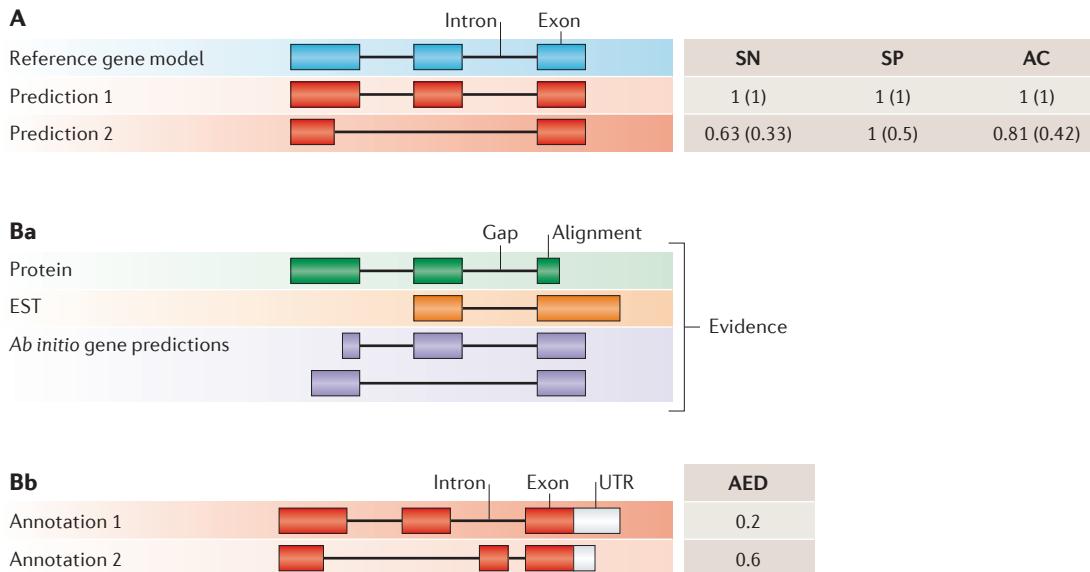
Numbers in parentheses are the values at the exon level; the others are nucleotide-level values. Note that the values for prediction 2 are lower at the exon level than they are at the nucleotide level. This is because exon-level calculations have an 'all-or-nothing' aspect to them: that is, a model in which the exons each differ by a single nucleotide from the reference will have nucleotide-level SN, SP and AC values near 1; its exon-level SN, SP and AC values, however, will all be 0.

With a few modifications, SN, SP and AC can also be used to compare two annotations to one another. This is the approach taken by the Sequence Ontology Project to calculate annotation edit distance (AED), which can be used to measure the congruence between an annotation and its supporting evidence⁹⁶. AED is calculated in the same manner as SN and SP, but in place of a reference gene model, the coordinates of the union of the aligned evidence (see panel **Ba**) are used instead: $AED = 1 - AC$, where $AC = (SN + SP) / 2$.

An AED of 0 indicates that the annotation is in perfect agreement with its evidence, whereas an AED of 1 indicates a complete lack of evidence support for the annotation. More information regarding AED can be found in REF. 96.

Panel **B** illustrates how AED is used. Panel **Ba** shows the protein, expressed sequence tag (EST) and *ab initio* gene predictions that are produced during the computation phase of the annotation process. Panel **Bb** shows two hypothetical annotations based on this evidence. Solid portions of boxes in panel **Bb** delimit coding sequence; note that the two annotations differ at their 3' untranslated regions (UTRs) as well as their coding-exon coordinates.

The table on the right in panel **Bb** shows how nucleotide-level AED values can be used to summarize the goodness of fit of an annotation to its overlapping evidence. Annotation 1 has the lower AED (of 0.2), meaning that it is a better fit to the evidence than annotation 2 (with an AED of 0.6) is; thus, bringing annotation 1 into perfect synchrony with the evidence would require fewer manual editing operations than would be required for annotation 2.



Step two: the annotation phase

The ultimate goal of annotation efforts is to obtain a synthesis of alignment-based evidence with *ab initio* gene predictions to obtain a final set of gene annotations. Traditionally, this was done manually; human genome annotators would review the evidence for each gene in order to decide on their intron-exon structures⁷⁵. Although this results in high-quality annotation^{76,77}, it is so labour-intensive that, for budgetary reasons, smaller genome projects are increasingly being forced to rely on automated annotations.

There are almost as many strategies for creating automated annotations as there are annotation pipelines, but the common theme is to use evidence to improve the accuracy of gene models, usually through some combination of pre- and post-processing of the gene predictions. FIGURE 2 and TABLE 1 provide an overview of some of the more commonly used approaches.

Automated annotation. The simplest form of automated annotation is to run a battery of different gene finders on the genome and then to use a ‘chooser algorithm’ (also known as a ‘combiner’) to select the single prediction whose intron-exon structure best represents the consensus of the models from among the overlapping predictions that define each putative gene locus. This is the process used by JIGSAW⁷⁸. EVidenceModeler (EVM)⁷⁹ and GLEAN⁸⁰ (and its successor, Evigan⁸¹) go one step further, attempting to choose the best possible set of exons automatically and to combine them to produce annotations. This is done by estimating the types and frequencies of errors that are made by each source of gene evidence and then choosing combinations of evidence that minimize such errors. Like *ab initio* gene predictors, JIGSAW must be retrained for each new genome, and so it requires a source of known gene models that were not already used to train the underlying *ab initio* gene predictors. EVM allows the user to set expected evidence error rates manually or to learn them from a training set. By contrast, GLEAN and Evigan use an unsupervised learning method to estimate a joint error model, and thus they require no additional training. In a recent gene prediction competition⁶⁴, the combiners nearly always improved on the underlying gene prediction models, and JIGSAW, EVM or Evigan performed similarly.

Another popular approach is to feed the alignment evidence to the gene predictors at run time (that is, evidence-driven prediction) to improve the accuracy of the prediction process — a chooser can then be used to identify the most representative prediction. The predictions can also be processed — before or after running the chooser — to attain still greater accuracies by having the annotation pipeline add UTRs as suggested by the RNA-seq and EST data. This is the process used by PASA^{56,82}, Gnomon⁷³ and MAKER¹⁰. The evidence can also be used to inform the choices made by the chooser algorithm — by picking the post-processed gene model that is most consistent with the protein, EST and RNA-seq alignments⁸³; EVM, MAKER and PASA all provide methods for doing so (TABLE 1; FIG. 2).

Unsupervised learning methods

Refers to methods that can be trained using unlabelled data. One example is a gene prediction algorithm that can be trained without a reference set of correct gene models; instead, the algorithm is trained using a collection of annotations, not all of which might be correct.

So which approach should you use? Probably the best way to think about the problem is in terms of effort versus accuracy. Simply running a single *ab initio* gene finder over even a very large genome can be done in a few hours of central processing unit (CPU) time. By contrast, a full run by an annotation pipeline such as MAKER or PASA can take weeks, but because these pipelines align evidence to the genome, their outputs provide starting points for annotation curation and downstream analyses, such as differential expression analyses using RNA-seq data. Another factor to consider is the phylogenetic relationship of the study genome to other annotated genomes. If it is the first of its taxonomic order or family to be annotated, it would definitely be preferable to use a pipeline that can use the full repertory of external evidence, especially RNA-seq data, to inform its gene annotations; not doing so will almost certainly result in low-quality annotations⁸⁰.

Visualizing the annotation data

Output data: the importance of using a fully documented format. The outputs of a genome annotation pipeline will include the transcript and protein sequences of every annotation, which are almost always provided in FASTA format⁸⁴. Although FASTA files are useful, they only enable a small subset of possible downstream analyses. Visualizing annotations in a genome browser and creating a genome database requires a more descriptive output file. At a bare minimum, output files need to describe the intron-exon structures of each annotation, their start and stop codons, UTRs and alternative transcripts. Ideally, these outputs should go one step further and should include information about the sequence alignments and gene predictions that support each gene model.

Four commonly used formats for describing annotations are the GenBank, GFF3, GTF and EMBL formats. Using a fully documented format is important for three reasons. First, doing so will remove the trouble of writing software to convert outputs into a format that other tools can use. Second, common formats, especially those such as GenBank and GFF3, which use controlled vocabularies and ontologies to define their descriptive terminologies, guarantee ‘interoperability’ between analysis tools. Third, unless a common vocabulary is used to describe gene models⁸⁵, comparative genomic analyses can be frustratingly difficult or downright impossible. In response to these needs, the Generic Model Organism Database (GMOD) project community has developed a series of standards and tools for description, analyses, visualization and redistribution of genome annotations, all of which use the GFF3 file format as inputs and outputs. Leveraging GMOD tools and GFF3 substantially simplifies curation, analysis, publication and management of genome annotations.

GMOD. The GMOD project is an umbrella organization that provides a large suite of tools for creating, managing and using genome annotations, including the analysis, visualization and redistribution of annotation data. Users who have browsed the

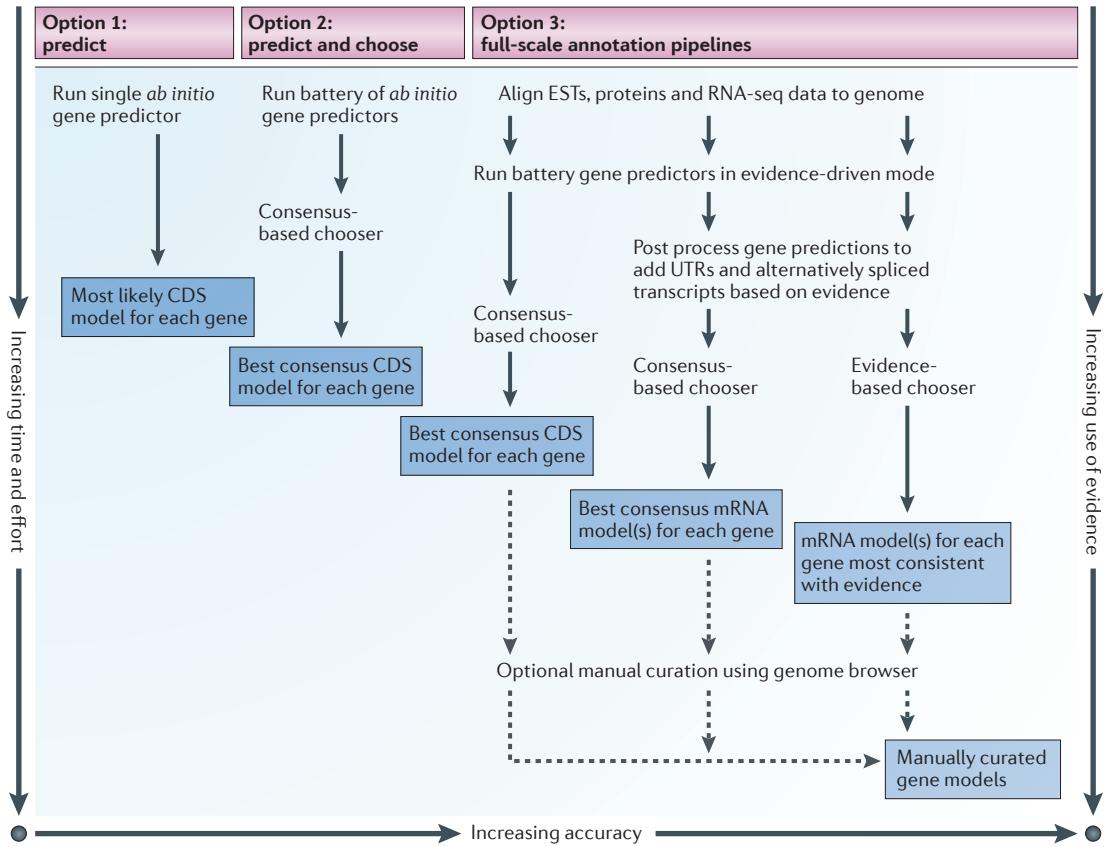


Figure 2 | Three basic approaches to genome annotation and some common variations. Approaches are compared on the basis of relative time, effort and the degree to which they rely on external evidence, as opposed to *ab initio* gene models. The y axis shows increasing time and effort; the x axis shows increasing use of external evidence and, consequently, increasing accuracy and completeness of the resulting gene models. The type of final product produced by each kind of pipeline is shown in the dark blue boxes. Relative positions in the figure are for summary purposes only and are not based on precisely computed values. See TABLE 1 for a list of commonly used software components. CDS, coding sequence; EST, expressed sequence tag; RNA-seq, RNA sequencing; UTR, untranslated region.

[Saccharomyces Genome Database](#), [WormBase](#), [FlyBase](#), [The Arabidopsis Information Resource](#) (TAIR) or the [University of California Santa Cruz \(UCSC\) Genome Browser](#) will have used GMOD tools. GMOD tools also aid in creating an online genome database. The key is having annotations and their associated evidence in GFF3 format, which is useable by GMOD tools. Users can directly visualize these files using [GBROWSE](#)⁸⁶ and [JBROWSE](#)⁸⁷ to produce views of their data just like those offered at WormBase and UCSC. They can also directly edit the gene models using the [Apollo](#) genome browser and JBROWSE. [BioPerl](#)⁸⁸ also provides a set of database tools for loading GFF3 files into a ready-made [Chado](#)⁸⁹ database schema with which an online genome database can be rapidly created that contains a genome and its annotations in a ‘browse-able’ format.

Quality control

Incorrect annotations poison every experiment that makes use of them. Worse still, the poison spreads because incorrect annotations from one organism are often unknowingly used by other projects to help annotate their own genomes. Standard practices for

genome annotation have been proposed for bacterial⁹⁰, viral⁹¹ and eukaryotic genomes⁹², but even when followed, quality control remains an issue. Even the best gene predictors and genome annotation pipelines rarely exceed accuracies of 80% at the exon level⁶³, meaning that most gene annotations contain at least one mis-annotated exon. Given these facts, assessing how accurately a genome is annotated is an important part of any project.

Over the years, there have been various contests aimed at assessing gene prediction accuracy^{63,65} (BOX 4). These contests have played an important part in improving the power and accuracy of gene prediction. However, less progress has been made regarding genome annotations⁶⁴. The heart of the problem is the absence of reference data sets with which to obtain accuracy estimates. The first generation of genome projects — *Saccharomyces cerevisiae*, *C. elegans* and *D. melanogaster*, for instance — all had decades of work to draw on when training and measuring the accuracy of gene predictors and annotation pipelines. However, no such data set exists for most of the organisms being sequenced today. Moreover, just because a gene

Table 1 | Five basic categories of annotation software and some selected examples

Software	Description	Refs
Ab initio and evidence-drivable gene predictors		
Augustus	Accepts expressed sequence tag (EST)-based and protein-based evidence hints. Highly accurate	66,67
mGene	Support vector machine (SVM)-based discriminative gene predictor. Directly predicts 5' and 3' untranslated regions (UTRs) and poly(A) sites	133
SNAP	Accepts EST and protein-based evidence hints. Easily trained	62
FGENESH	Training files are constructed by <i>SoftBerry</i> and supplied to users	72
Geneid	First published in 1992 and revised in 2000. Accepts external hints from EST and protein-based evidence	134
Genemark	A self-training gene finder	69,70
Twinscan	Extension of the popular Genscan algorithm that can use homology between two genomes to guide gene prediction	71
GAZE	Highly configurable gene predictor	74
GenomeScan	Extension of the popular Genscan algorithm that can use BLASTX searches to guide gene prediction	135
Conrad	Discriminative gene predictor that uses conditional random fields (CRFs)	136
Contrast	Discriminative gene predictor that uses both SVMs and CRFs	137
CRAIG	Discriminative gene predictor that uses CRFs	138
Gnomon	Hidden Markov model (HMM) tool based on Genscan that uses EST and protein alignments to guide gene prediction	73
GeneSeqr	A tool for identifying potential exon–intron structure in precursor mRNAs (pre-mRNAs) by splice site prediction and spliced alignment	139
EST, protein and RNA-seq aligners and assemblers		
BLAST	Suite of rapid database search tools that uses Karlin–Altschul statistics	31–33
BLAT	Faster than BLAST but has fewer features	42
Splign	Splice-aware tool designed to align cDNA to genomic sequence	44
Spidey	mRNA-to-DNA alignment tool that is designed to account for possible paralogous alignments	45
Prospalign	Global alignment tool that uses BLAST hits to align in a splice-site- and paralogy-aware manner	140
sim4	Splice-aware cDNA-to-DNA alignment tool	46
Exonerate	Splice-site-aware alignment algorithm that can align both protein and EST sequences to a genome	43
Cufflinks	Extension to TopHat. Uses TopHat outputs to create transcript models	54
Trinity	High-quality <i>de novo</i> transcriptome assembler	50
MapSplice	Spliced aligner that does not use a model of canonical splice junction	141
TopHat	Transcriptome aligner that aligns RNA sequencing (RNA-seq) reads to a reference genome using Bowtie to identify splice sites	51
GSNAP	A fast short-read assembler	52
Choosers and combiners		
JIGSAW	Combines evidence from alignment and <i>ab initio</i> gene prediction tools to produce a consensus gene model	78
EVidenceModeler	Produces a consensus gene model by combining evidence from protein and transcript alignments together with <i>ab initio</i> predictions using weights for both abundance and the sources of the evidence	79
GLEAN	Tool for creating consensus gene lists by integrating gene evidence through latent class analysis	80
Evigan	Probabilistic evidence combiner that uses a Bayesian network to weigh and integrate evidence from <i>ab initio</i> predictors, alignments and expression data to produce a consensus gene model	81

Table 1 (cont.) | Five basic categories of annotation software and some selected examples

Software	Description	Refs
Genome annotation pipelines		
PASA	Annotation pipeline that aligns EST and protein sequences to the genome and produces evidence-driven consensus gene models	56,82
MAKER	Annotation pipeline that uses BLAST and exonrate to align protein and EST sequences. Also accepts features from RNA-seq alignment tools (such as TopHat). Massively parallel	10,83
NCBI	The genome annotation pipeline from the US National Center for Biotechnology Information (NCBI). Uses BLAST alignments together with predictions from Gnomon and GenomeScan to produce gene models	142
Ensembl	Ensembl's genome annotation pipeline. Uses species-specific and cross-species alignments to build gene models. Also annotates non-coding RNAs	107
Genome browsers for curation		
Artemis	Java-based genome browser for feature viewing and annotation. Can use binary alignment map (BAM) files as input	99
Apollo	Java-based genome browser that allows the user to create and edit gene models and write their edits to a remote database	97
JBROWSE	JavaScript- and HTML-based genome browser that can be embedded into wikis for community work. Excellent for Web-based use	87
IGV	Genome browser that supports BAM files and expression data	143

These tools are widely used both as standalone applications and as modular components of genome annotation pipelines. See FIG. 2 for a schematic of the roles of each class of tool in genome annotation.

predictor does well on one genome is no guarantee of a good performance on the next⁸³. Assessing annotation quality in the absence of reference genome annotations is a difficult problem. Experimental verification is one solution, but few projects have the resources to carry this out on a large scale.

Approaches for assessing annotation quality. One simple approach for obtaining a rough indication of annotation quality is to quantify the percentage of annotations that encode proteins with known domains using tools such as InterProScan⁹³ and Pfam⁹⁴ or tools such as MAKER, which provides an automated means for carrying out such analyses⁸³. Although relative numbers of domains vary between organisms and the expansion and contraction of particular gene families have a well-established role in organismal evolution, among the eukaryotes, the overall percentage of proteins that encode a domain of any sort is reasonably constant⁸³. The domain content of the human, *D. melanogaster*, *C. elegans*, *A. thaliana* and *S. cerevisiae* proteomes varies between 57% and 75%⁹⁵. Poorly trained gene finders do not perform nearly this well — 5% to 25% is typical. Thus, a eukaryotic proteome with a low percentage of domains is a warning sign that it could be poorly annotated⁸³.

Although domain content provides a rough estimate of overall annotation quality, it provides little guidance when trying to judge the accuracy of a given annotation. One approach towards solving this problem is to ask whether the protein, EST and RNA-seq evidence support or contradict the annotated intron-exon structure of the gene. This is fairly straightforward to assess by eye, but performing this task in an automated fashion requires a computable metric. In response, the

Sequence Ontology Project⁸⁵ has developed several metrics for quality control of genome annotations⁹⁶. Annotation edit distance (AED), for example, measures how congruent each annotation is with its overlapping evidence (BOX 4). AED thus provides a means to identify problematic annotations automatically and to prioritize them for manual curation. AED scores can also be used to measure changes to annotations between annotation runs. The MAKER2 genome annotation pipeline⁸³ provides some useful tools for automatically calculating AED.

Of course, identifying inaccurate annotations is only half of the problem; errors also need to be corrected. The most direct approach to fixing an erroneous annotation is to edit its intron-exon coordinates manually. The Apollo⁹⁷, Argo⁹⁸ and Artemis⁹⁹ browsers are widely used for this purpose. Gene models can be graphically revised using a series of ‘drag-and-drops’ and mouse clicks, and the resulting edits are written back to either files or to a remote database connection⁸⁹.

Annotation jamborees. Many genome projects choose to manually review and edit their annotation data sets. Although this process is time- and resource-intensive, it provides opportunities for community building, education and training.

Annotation jamborees (a term that was coined by the *D. melanogaster* community to describe the first such gathering¹⁰⁰) provide a ready means for manual curation and analysis of the data and for putting together a genome paper. The key to hosting a successful jamboree is infrastructure. At a minimum, attendees must be able to search the annotated proteins and transcripts and to view the annotations in a genome browser. Searches can easily be handled by setting up a

BLAST database server coupled with a graphical user interface (GUI) such as a Web browser. The WWW BLAST server package¹⁰¹ provides an easy means to do so. GBrowse^{86,102} and JBrowse⁸⁷ can also easily be configured to allow remote users to view the annotated genome, as can the Apollo genome browser, which also provides a means to edit incorrect annotations. As all of these resources can be set up and configured remotely, it is now possible to support a distributed jamboree, in which the community collaborates via the Internet. This model recently proved to be successful for the ant genome community, which organized a distributed jamboree in which investigators and students collaborated to curate and analyse three different ant genomes quickly, all in a distributed manner^{103–106}.

Making data publicly available

Successful genome annotation projects do not just end with the publication of a paper; they also produce publicly available annotations. Genome annotations fuel the bench work and computational analyses that constitute the day-to-day operations of molecular biology and bioinformatics laboratories worldwide. They also provide an essential resource for other genome annotation projects; the transcripts and proteins produced by one annotation project will probably be used to help annotate other genomes. There are three basic routes to making annotations publicly available: you can build your own genome database and place it online; you can submit your annotations to GenBank and Ensembl; or you can submit them to any of a growing number of theme-based genome databases. We recommend taking all three routes.

Submitting annotations to public databases. One way to make annotations publicly available is to submit them to GenBank. Parties working on vertebrate genomes are also encouraged to contact Ensembl, which continues to incorporate new species at the rate of 5–10 per year in order to create a comprehensive annotation resource for vertebrate genomes, all of which are annotated by its gene build pipeline¹⁰⁷. Both GenBank and Ensembl have much to offer to smaller genome projects, including powerful data-marts that allow users to browse and download data. Ensembl and GenBank also automatically handle the heavy lifting that is involved in relating gene models to those of other organisms and identifying homologues, paralogues and orthologues. They also provide an easy means to search and browse data; in short, they integrate a data set into the larger landscape of genomics and genome annotations. Best of all, the entire process is free, and submission to these sites in no way abridges the rights of the generators of the data to host and maintain their own genome database. For the research communities of most organisms, members will prefer to visit the specialized genome database for that organism, whereas the larger biological community will tend to access the data through GenBank and Ensembl. In addition to these large sites, intermediate-sized projects that host, manage and maintain sets of annotated genomes that are all

related by a common theme are gaining in popularity. Examples include BeeBase¹⁰³, Gramene¹⁰⁸, PlantGDB¹⁰⁹, Phytozome¹¹⁰ and VectorBase¹¹¹.

Updating annotations. Many genomes were annotated so long ago that the existing annotations could be dramatically improved using modern tools and data sets such as RNA-seq. In many cases, improved assemblies are possible as well. The question then becomes how to merge, update and improve the existing annotations and, at the same time, to document the process. Like annotation quality control, this is a thorny problem that until recently has garnered little attention, and few published tools yet exist to automate the process. Among existing tools, GLEAN and PASA can be used to report differences between pre-existing gene models and newly created ones. Ensembl has a procedure to merge annotation data sets to produce a consensus, and PASA has one for updating annotations with RNA-seq data. The MAKER annotation pipeline provides an automated toolkit with all of these functionalities and can revise, update and merge existing annotation data sets, as well as map them forwards to new assemblies^{10,83}.

GenBank provides two avenues for redistributing the results of updates and re-annotation of genomes. If the group that is updating the annotations includes the original authors, the update can simply be submitted; if not, there are two routes for submission. If the work involves substantial improvements to the original assembly, the parties producing them can submit the new annotations to GenBank as primary authors; if not — that is, if the revisions merely improve the original annotations — those producing them can submit their work through the third party submission channel. Ensembl also allows submission of such data, although the process is less formal, and interested parties should contact Ensembl directly.

Conclusions

In some ways, cheap sequencing has complicated genome annotation. As we have explained, the fragmented assemblies and exotic nature of many of the current genome-sequencing projects are part of the reason that this is so, but it is the ever-widening scope of annotation that is presenting the greatest challenges. Genome annotation has moved beyond merely identifying protein-coding genes to include an ever-greater emphasis on the annotation of transposons, regulatory regions, pseudogenes and ncRNA genes^{112–115}. Annotation quality control and management are also increasingly becoming bottlenecks. As long as tools and sequencing technologies continue to develop, periodic updates to every genome's annotations will remain necessary. Those undertaking genome annotation projects need to reflect on this fact. Like parenthood, annotation responsibilities do not end with birth. Incorrect and incomplete annotations poison every experiment that makes use of them. In today's genomics-driven world, providing accurate and up-to-date annotations is simply a must.

Data-mart

Provides users with online access to the contents of a data warehouse through user-configurable queries. A data-mart allows users to download data that meet their particular needs: for example, all transcripts from all annotated genes on human chromosome 3.

1. Adams, M. D. *et al.* The genome sequence of *Drosophila melanogaster*. *Science* **287**, 2185–2195 (2000).
2. Celiker, S. E. *et al.* Finishing a whole-genome shotgun: release 3 of the *Drosophila melanogaster* euchromatic genome sequence. *Genome Biol.* **3**, research0079 (2002).
3. Venter, J. C. *et al.* The sequence of the human genome. *Science* **291**, 1304–1351 (2001).
4. Finishing the euchromatic sequence of the human genome. *Nature* **431**, 931–945 (2004).
5. Denoeud, F. *et al.* Annotating genomes with massive-scale RNA sequencing. *Genome Biol.* **9**, R175 (2008).
6. Ozsolak, F. *et al.* Direct RNA sequencing. *Nature* **461**, 814–818 (2009).
7. Mortazavi, A., Williams, B. A., McCue, K., Schaeffer, L. & Wold, B. Mapping and quantifying mammalian transcriptomes by RNA-seq. *Nature Methods* **5**, 621–628 (2008).
8. Wang, E. T. *et al.* Alternative isoform regulation in human tissue transcriptomes. *Nature* **456**, 470–476 (2008). **This paper provides one of the most extensively documented surveys of alternatively spliced transcripts. It is a key publication for understanding how extensive alternative splicing is in human tissues, for understanding how powerful RNA-seq data are as a tool for discovering new transcripts and for quantifying their abundance and differential expression patterns.**
9. Chain, P. S. *et al.* Genomics: Genome project standards in a new era of sequencing. *Science* **326**, 236–237 (2009).
10. Cantarel, B. L. *et al.* MAKER: an easy-to-use annotation pipeline designed for emerging model organism genomes. *Genome Res.* **18**, 188–196 (2008).
11. Ye, L. *et al.* A vertebrate case study of the quality of assemblies derived from next-generation sequences. *Genome Biol.* **12**, R31 (2011).
12. Parra, G., Bradnam, K. & Korf, I. CEGMA: a pipeline to accurately annotate core genes in eukaryotic genomes. *Bioinformatics* **23**, 1061–1067 (2007).
13. Tsai, I. J., Otto, T. D. & Berriman, M. Improving draft assemblies by iterative mapping and assembly of short reads to eliminate gaps. *Genome Biol.* **11**, R41 (2010).
14. Assefa, S., Keane, T. M., Otto, T. D., Newbold, C. & Berriman, M. ABACAS: algorithm-based automatic contiguation of assembled sequences. *Bioinformatics* **25**, 1968–1969 (2009).
15. Husemann, P. & Stoye, J. r2cat: synteny plots and comparative assembly. *Bioinformatics* **26**, 570–571 (2010).
16. Kapitonov, V. V. & Jurka, J. A novel class of SINE elements derived from 5S rRNA. *Mol. Biol. Evol.* **20**, 694–702 (2003).
17. Kapitonov, V. V. & Jurka, J. A universal classification of eukaryotic transposable elements implemented in Repbase. *Nature Rev. Genet.* **9**, 411–412; author reply 414 (2008).
18. Lander, E. S. *et al.* Initial sequencing and analysis of the human genome. *Nature* **409**, 860–921 (2001).
19. Buisine, N., Quesneville, H. & Colot, V. Improved detection and annotation of transposable elements in sequenced genomes using multiple reference sequence sets. *Genomics* **91**, 467–475 (2008).
20. Han, Y. & Wessler, S. R. MITE-Hunter: a program for discovering miniature inverted-repeat transposable elements from genomic sequences. *Nucleic Acids Res.* **38**, e199 (2010).
21. McClure, M. A. *et al.* Automated characterization of potentially active retrotransposons in the human genome. *Genomics* **85**, 512–523 (2005).
22. Bao, Z. & Eddy, S. R. Automated *de novo* identification of repeat sequence families in sequenced genomes. *Genome Res.* **12**, 1269–1276 (2002).
23. Price, A. L., Jones, N. C. & Pevzner, P. A. *De novo* identification of repeat families in large genomes. *Bioinformatics* **21** (Suppl. 1), i351–i358 (2005).
24. Smit, A. & Hubley, R. RepeatModeler 1.05. *repeatmasker.org* [online], <http://www.repeatmasker.org/RepeatModeler.html> (2011).
25. Morgulis, A., Gertz, E. M., Schaffer, A. A. & Agarwala, R. WindowMasker: window-based masker for sequenced genomes. *Bioinformatics* **22**, 134–141 (2006).
26. Treangen, T. J. & Salzberg, S. L. Repetitive DNA and next-generation sequencing: computational challenges and solutions. *Nature Rev. Genet.* **13**, 36–46 (2012).
27. Bergman, C. M. & Quesneville, H. Discovering and detecting transposable elements in genome sequences. *Brief. Bioinform.* **8**, 382–392 (2007).
28. Cordaux, R. & Batzer, M. A. The impact of retrotransposons on human genome evolution. *Nature Rev. Genet.* **10**, 691–703 (2009).
29. Witherspoon, D. J. *et al.* Alu repeats increase local recombination rates. *BMC Genomics* **10**, 530 (2009).
30. Smit, A. F., Hubley, R. & Green, P. RepeatMasker 3.0 *repeatmasker.org* [online], <http://www.repeatmasker.org/webrepeatmaskerhelp.html> (1996–2010).
31. Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J. Basic local alignment search tool. *J. Mol. Biol.* **215**, 403–410 (1990).
32. Korf, I., Yandell, M. & Bedell, J. *BLAST: an Essential Guide to the Basic Local Alignment Search Tool* 339 (O'Reilly & Associates, 2003). **Everyone involved with a genome project should be familiar with BLAST. Reference 31 is the original paper describing this tool. Reference 32 is an entire book describing BLAST and how it is used.**
33. Altschul, S. F. *et al.* Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* **25**, 3389–3402 (1997).
34. Green, P. Crossmatch: a general purpose utility for comparing any two sets of DNA sequences. *PHRAP* [online], <http://www.phrap.org/phredphrap/general.html> (1993–1996).
35. Majoros, W. H. *Methods for Computational Gene Prediction* 2 (Cambridge Univ. Press, 2007).
36. Camacho, C. *et al.* BLAST+: architecture and applications. *BMC Bioinformatics* **10**, 421 (2009).
37. Bairoch, A., Boeckmann, B., Ferro, S. & Gasteiger, E. Swiss-Prot: juggling between evolution and stability. *Brief. Bioinform.* **5**, 39–55 (2004).
38. Boeckmann, B. *et al.* Protein variety and functional diversity: Swiss-Prot annotation in its biological context. *C.R. Biol.* **328**, 882–899 (2005).
39. The UniProt Consortium. Ongoing and future developments at the Universal Protein Resource. *Nucleic Acids Res.* **39**, D214–D219 (2011).
40. Benson, D. A., Karsch-Mirzachi, I., Lipman, D. J., Ostell, J. & Sayers, E. W. GenBank. *Nucleic Acids Res.* **37**, D26–D31 (2009).
41. Sayers, E. W. *et al.* Database resources of the National Center for Biotechnology Information. *Nucleic Acids Res.* **37**, D5–D15 (2009).
42. Kent, W. J. BLAT—the BLAST-like alignment tool. *Genome Res.* **12**, 656–664 (2002).
43. Slater, G. S. & Birney, E. Automated generation of heuristics for biological sequence comparison. *BMC Bioinformatics* **6**, 31 (2005).
44. Kapustin, Y., Souvorov, A., Tatusova, T. & Lipman, D. SpliceN: algorithms for computing spliced alignments with identification of paralogs. *Biol. Direct* **3**, 20 (2008).
45. Wheeler, S. J., Church, D. M. & Ostell, J. M. Spidey: a tool for mRNA-to-genomic alignments. *Genome Res.* **11**, 1952–1957 (2001).
46. Florea, L., Hartzell, G., Zhang, Z., Rubin, G. M. & Miller, W. A computer program for aligning a cDNA sequence with a genomic DNA sequence. *Genome Res.* **8**, 967–974 (1998).
47. Garber, M., Grabherr, M. G., Guttman, M. & Trapnell, C. Computational methods for transcriptome annotation and quantification using RNA-seq. *Nature Methods* **8**, 469–477 (2011).
48. Simpson, J. T. *et al.* ABYSS: a parallel assembler for short read sequence data. *Genome Res.* **19**, 1117–1123 (2009).
49. Li, R. *et al.* De novo assembly of human genomes with massively parallel short read sequencing. *Genome Res.* **20**, 265–272 (2010).
50. Grabherr, M. G. *et al.* Full-length transcriptome assembly from RNA-seq data without a reference genome. *Nature Biotech.* **29**, 644–652 (2011). **This paper describes Trinity, a transcriptome assembler that was specifically designed for next-generation sequence data. It is required reading for anyone trying to use RNA-seq data for genome annotation.**
51. Trapnell, C., Pachter, L. & Salzberg, S. L. TopHat: discovering splice junctions with RNA-seq. *Bioinformatics* **25**, 1105–1111 (2009).
52. Wu, T. D. & Nacu, S. Fast and SNP-tolerant detection of complex variants and splicing in short reads. *Bioinformatics* **26**, 873–881 (2010).
53. Guttman, M. *et al.* Ab initio reconstruction of cell type-specific transcriptomes in mouse reveals the conserved multi-exonic structure of lincRNAs. *Nature Biotech.* **28**, 503–510 (2010).
54. Trapnell, C. *et al.* Transcript assembly and quantification by RNA-seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature Biotech.* **28**, 511–515 (2010).
55. Trapnell, C. *et al.* Differential gene and transcript expression analysis of RNA-seq experiments with TopHat and Cufflinks. *Nature Protoc.* **7**, 562–578 (2012). **This paper describes best practice approaches for combining TopHat and Cufflinks when using RNA-seq data for genome annotation.**
56. Haas, B. J. *et al.* Improving the *Arabidopsis* genome annotation using maximal transcript alignment assemblies. *Nucleic Acids Res.* **31**, 5654–5666 (2003).
57. Guigo, R., Knudsen, S., Drake, N. & Smith, T. Prediction of gene structure. *J. Mol. Biol.* **226**, 141–157 (1992).
58. Solovyev, V. V., Salamov, A. A. & Lawrence, C. B. The prediction of human exons by oligonucleotide composition and discriminant analysis of spliceable open reading frames. *Proc. Int. Conf. Intell. Syst. Mol. Biol.* **2**, 354–362 (1994).
59. Burge, C. & Karlin, S. Prediction of complete gene structures in human genomic DNA. *J. Mol. Biol.* **268**, 78–94 (1997). **This study describes the *ab initio* gene predictor GenScan. It is a classic paper that is full of informative explanations of the problems associated with eukaryotic gene prediction.**
60. Reese, M. G., Kulp, D., Tammanna, H. & Haussler, D. Genie—gene finding in *Drosophila melanogaster*. *Genome Res.* **10**, 529–538 (2000).
61. Brent, M. R. Genome annotation past, present, and future: how to define an ORF at each locus. *Genome Res.* **15**, 1777–1786 (2005).
62. Korf, I. Gene finding in novel genomes. *BMC Bioinformatics* **5**, 59 (2004). **This paper describes a gene predictor, SNAP, that is easy to use and to configure. It also clearly explains the pitfalls that are associated with using a poorly trained gene finder or one that has been trained on a different genome from the one that is being annotated.**
63. Reese, M. G. & Guigo, R. EGASP: Introduction. *Genome Biol.* **7** (Suppl. 1), 1–3 (2006). **This is the introduction to an entire issue of *Genome Biology* that is dedicated to benchmarking an entire host of eukaryotic gene finders and annotation pipelines. Anyone involved with a genome annotation project should have a look at every paper in this special supplement.**
64. Coghlan, A. *et al.* nGASP—the nematode genome annotation assessment project. *BMC Bioinformatics* **9**, 549 (2008).
65. Guigo, R. & Reese, M. G. EGASP: collaboration through competition to find human genes. *Nature Methods* **2**, 575–577 (2005).
66. Stanke, M. & Waack, S. Gene prediction with a hidden Markov model and a new intron submodel. *Bioinformatics* **19** (Suppl. 2), ii215–ii225 (2003).
67. Stanke, M., Schöffmann, O., Morgenstern, B. & Waack, S. Gene prediction in eukaryotes with a generalized hidden Markov model that uses hints from external sources. *BMC Bioinformatics* **7**, 62 (2006).
68. Lukashin, A. V. & Borodovsky, M. GeneMark.hmm: new solutions for gene finding. *Nucleic Acids Res.* **26**, 1107–1115 (1998).
69. Ter-Hovhannisyan, V., Lomsadze, A., Chernoff, Y. O. & Borodovsky, M. Gene prediction in novel fungal genomes using an *ab initio* algorithm with unsupervised training. *Genome Res.* **18**, 1979–1990 (2008).
70. Zhu, W., Lomsadze, A. & Borodovsky, M. *Ab initio* gene identification in metagenomic sequences. *Nucleic Acids Res.* **38**, e132 (2010).
71. Korf, I., Flicek, P., Duan, D. & Brent, M. R. Integrating genomic homology into gene structure prediction. *Bioinformatics* **17**, S140–S148 (2001).
72. Salamov, A. A. & Solovyev, V. V. *Ab initio* gene finding in *Drosophila* genomic DNA. *Genome Res.* **10**, 516–522 (2000).
73. Souvorov, A. *et al.* Gnomon — the NCBI eukaryotic gene prediction tool. *National Center for Biotechnology Information* [online], <http://www.ncbi.nlm.nih.gov/genome/guide/gnomon.shtml> (2010).

74. Howe, K. L., Chothia, T. & Durbin, R. GAZE: a generic framework for the integration of gene-prediction data by dynamic programming. *Genome Res.* **12**, 1418–1427 (2002).
75. Mungall, C. J. *et al.* An integrated computational pipeline and database to support whole-genome sequence annotation. *Genome Biol.* **3**, research0081 (2002).
76. Misra, S. *et al.* Annotation of the *Drosophila melanogaster* euchromatic genome: a systematic review. *Genome Biol.* **3**, research0083 (2002).
77. Yandell, M. *et al.* A computational and experimental approach to validating annotations and gene predictions in the *Drosophila melanogaster* genome. *Proc. Natl Acad. Sci. USA* **102**, 1566–1571 (2005).
78. Allen, J. E. & Salzberg, S. L. JIGSAW: integration of multiple sources of evidence for gene prediction. *Bioinformatics* **21**, 3596–3603 (2005).
79. Haas, B. J. *et al.* Automated eukaryotic gene structure annotation using EVidenceModeler and the Program to Assemble Spliced Alignments. *Genome Biol.* **9**, R7 (2008).
80. Elsik, C. G. *et al.* Creating a honey bee consensus gene set. *Genome Biol.* **8**, R13 (2007).
81. Liu, Q., Mackey, A. J., Roos, D. S. & Pereira, F. C. Evigan: a hidden variable model for integrating gene evidence for eukaryotic gene prediction. *Bioinformatics* **24**, 597–605 (2008).
82. Haas, B. J., Zeng, Q., Pearson, M. D., Cuomo, C. A. & Wortman, J. R. Approaches to fungal genome annotation. *Mycology* **2**, 118–141 (2011). **This paper provides an excellent description of the process used by the Broad Institute for fungal annotation. It is also a good resource for those seeking to learn more about PASA; for more information about PASA, see reference 56.**
83. Holt, C. & Yandell, M. MAKER2: an annotation pipeline and genome-database management tool for second-generation genome projects. *BMC Bioinformatics* **12**, 491 (2011). **This study describes the database management and annotation quality-control tools for the MAKER2 genome annotation pipeline. It also explains many of the challenges that are associated with annotating novel genomes and how to overcome them.**
84. Pearson, W. R. & Lipman, D. J. Improved tools for biological sequence comparison. *Proc. Natl Acad. Sci. USA* **85**, 2444–2448 (1988).
85. Eilbeck, K. *et al.* The Sequence Ontology: a tool for the unification of genome annotations. *Genome Biol.* **6**, R44 (2005).
86. Donlin, M. J. in *Current Protocols in Bioinformatics*. Ch. 9, Unit 9.9 (2007).
87. Skinner, M. E., Uzilov, A. V., Stein, L. D., Mungall, C. J., Holmes, I. H. JBrowse: a next-generation genome browser. *Genome Res.* **19**, 1630–1638 (2009).
88. Stajich, J. E. *et al.* The Bioperl toolkit: Perl modules for the life sciences. *Genome Res.* **12**, 1611–1618 (2002).
89. Zhou, P., Emmert, D. & Zhang, P. in *Current Protocols in Bioinformatics* Ch. 9, Unit 9.6 (2006).
90. Klimke, W. *et al.* Solving the problem: genome annotation standards before the data deluge. *Stand. Genomic Sci.* **5**, 168–193 (2011).
91. Brister, J. R. *et al.* Towards viral genome annotation standards, report from the 2010 NCBI annotation workshop. *Viruses* **2**, 2258–2268 (2010).
92. Madupu, R. *et al.* Meeting report: a workshop on best practices in genome annotation. *Database* **2010**, baq001 (2010).
93. Mulder, N. & Apweiler, R. InterPro and InterProScan: tools for protein sequence classification and comparison. *Methods Mol. Biol.* **396**, 59–70 (2007).
94. Finn, R. D. *et al.* The Pfam protein families database. *Nucleic Acids Res.* **38**, D211–D222 (2010).
95. Holt, C. *Tools and Techniques for Genome Annotation Analysis*. Ph.D. thesis, Univ. Utah (2011).
96. Eilbeck, K., Moore, B., Holt, C. & Yandell, M. Quantitative measures for the management and comparison of annotated genomes. *BMC Bioinformatics* **10**, 67 (2009). **This paper describes a number of annotation quality-control measures, including annotation edit distance (AED). It also provides some interesting meta-analyses describing the impact of curation efforts on the gene annotations of several model organism databases over a period of several years.**
97. Lewis, S. E. *et al.* Apollo: a sequence annotation editor. *Genome Biol.* **3**, research0082 (2002).
98. Engels, R. Argo Genome Browser version 1.0.31. *Broad Institute* [online], <http://www.broadinstitute.org/annotation/argo> (2010).
99. Rutherford, K. *et al.* Artemis: sequence visualization and annotation. *Bioinformatics* **16**, 944–945 (2000).
100. Hartl, D. L. Fly meets shotgun: shotgun wins. *Nature Genet.* **24**, 327–328 (2000).
101. Desk, B. H. Introduction to the standalone WWW Blast server. *National Center for Biotechnology Information* [online], <http://www.ncbi.nlm.nih.gov/blast/docs/wwwblast.html> (2002).
- This page explains how to use a suite of programs to set up a local Blast server for your local database.**
102. Stein, L. D. *et al.* The generic genome browser: a building block for a model organism system database. *Genome Res.* **12**, 1599–1610 (2002).
103. Munoz-Torres, M. C. *et al.* Hymenoptera Genome Database: integrated community resources for insect species of the order Hymenoptera. *Nucleic Acids Res.* **39**, D658–D662 (2011).
104. Smith, C. D. *et al.* Draft genome of the globally widespread and invasive Argentine ant (*Linepithema humile*). *Proc. Natl Acad. Sci. USA* **108**, 5673–5678 (2011).
105. Suen, G. *et al.* The genome sequence of the leaf-cutter ant *Atta cephalotes* reveals insights into its obligate symbiotic lifestyle. *PLoS Genet.* **7**, e1002007 (2011).
106. Nygaard, S. *et al.* The genome of the leaf-cutting ant *Acromyrmex echinatior* suggests key adaptations to advanced social life and fungus farming. *Genome Res.* **21**, 1339–1348 (2011).
107. Curwen, V. *et al.* The Ensembl automatic gene annotation system. *Genome Res.* **14**, 942–950 (2004). **This paper describes the Ensembl genome annotation pipeline; although the article is now several years old, it is still a good place to start. We would recommend reading this paper and then browsing the extensive Ensembl web site for more information.**
108. Youens-Clark, K. *et al.* Gramene database in 2010: updates and extensions. *Nucleic Acids Res.* **39**, D1085–D1094 (2011).
109. Duvick, J. *et al.* PlantGDB: a resource for comparative plant genomics. *Nucleic Acids Res.* **36**, D959–D965 (2008).
110. Goodstein, D. M. *et al.* Phytozome: a comparative platform for green plant genomics. *Nucleic Acids Res.* **40**, D1178–D1186 (2012).
111. Lawson, D. *et al.* VectorBase: a data resource for invertebrate vector genomics. *Nucleic Acids Res.* **37**, D583–D587 (2009).
112. Karro, J. E. *et al.* Pseudogene.org: a comprehensive database and comparison platform for pseudogene annotation. *Nucleic Acids Res.* **35**, D55–D60 (2007).
113. Zheng, D. *et al.* Integrated pseudogene annotation for human chromosome 22: evidence for transcription. *J. Mol. Biol.* **349**, 27–45 (2005).
114. Griffiths-Jones, S., Bateman, A., Marshall, M., Khanna, A. & Eddy, S. R. Rfam: an RNA family database. *Nucleic Acids Res.* **31**, 439–441 (2003).
115. Lagesen, K. *et al.* RNAMMER: consistent and rapid annotation of ribosomal RNA genes. *Nucleic Acids Res.* **35**, 3100–3108 (2007).
116. Dolezel, J. & Bartos, J. Plant DNA flow cytometry and estimation of nuclear genome size. *Ann. Botany* **95**, 99–110 (2005).
117. Laird, C. D. & McCarthy, B. J. Molecular characterization of the *Drosophila* genome. *Genetics* **63**, 865–882 (1969).
118. Lowe, T. M. & Eddy, S. R. tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence. *Nucleic Acids Res.* **25**, 955–964 (1997).
119. Schattner, P., Brooks, A. N. & Lowe, T. M. The tRNAscan-SE, snoscan and snoGPS web servers for the detection of tRNAs and snoRNAs. *Nucleic Acids Res.* **33**, W686–W689 (2005).
120. Lewis, B. P., Shih, I. H., Jones-Rhoades, M. W., Bartel, D. P. & Burge, C. B. Prediction of mammalian microRNA targets. *Cell* **115**, 787–798 (2003).
121. Eddy, S. R. A memory-efficient dynamic programming algorithm for optimal alignment of a sequence to an RNA secondary structure. *BMC Bioinformatics* **3**, 18 (2002).
122. Holmes, I. & Rubin, G. M. Pairwise RNA structure comparison with stochastic context-free grammars. *Pac. Symp. Biocomput.* **7**, 163–174 (2002).
123. QIAGEN. Quick-Start Protocol miRNAeasy Mini Kit. *QIAGEN* [online], <http://www.qiagen.com/products/rnastabilizationpurification/microrna/mirneasyminikit.aspx#Tabs=t2> (2011).
124. Chen, C. *et al.* Real-time quantification of microRNAs by stem-loop RT-PCR. *Nucleic Acids Res.* **33**, e179 (2005).
125. van Leeuwen, S. & Mikkers, H. Long non-coding RNAs: guardians of development. *Differentiation* **80**, 175–183 (2010).
126. Hung, T. & Chang, H. Y. Long noncoding RNA in genome regulation: prospects and mechanisms. *RNA Biol.* **7**, 582–585 (2010).
127. Tam, O. H. *et al.* Pseudogene-derived small interfering RNAs regulate gene expression in mouse oocytes. *Nature* **453**, 534–538 (2008).
128. Zhang, Z., Carriero, N. & Gerstein, M. Comparative analysis of processed pseudogenes in the mouse and human genomes. *Trends Genet.* **20**, 62–67 (2004).
129. Nawrocki, E. P., Kolbe, D. L. & Eddy, S. R. Infernal 1.0: inference of RNA alignments. *Bioinformatics* **25**, 1335–1337 (2009).
130. Burset, M. & Guigo, R. Evaluation of gene structure prediction programs. *Genomics* **34**, 353–367 (1996). **This paper provides an excellent explanation of how sensitivity and specificity measures can be used to evaluate gene finder performance. This is a classic paper in the field and should be read by anyone involved in gene annotation.**
131. Baldi, P., Brunak, S., Chauvin, Y., Andersen, C. A. & Nielsen, H. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics* **16**, 412–424 (2000).
132. Guigo, R. *et al.* EGASP: the human ENCODE Genome Annotation Assessment Project. *Genome Biol.* **7** (Suppl. 1), 1–31 (2006).
133. Schweikert, G. *et al.* mGene: accurate SVM-based gene finding with an application to nematode genomes. *Genome Res.* **19**, 2133–2143 (2009).
134. Parra, G., Blanco, E. & Guigo, R. GenEd in *Drosophila*. *Genome Res.* **10**, 511–515 (2000).
135. Yeh, R. F., Lim, L. P. & Burge, C. B. Computational inference of homologous gene structures in the human genome. *Genome Res.* **11**, 803–816 (2001).
136. DeCaprio, D. *et al.* Conrad: gene prediction using conditional random fields. *Genome Res.* **17**, 1389–1398 (2007).
137. Gross, S. S., Do, C. B., Sirota, M. & Batzoglou, S. CONTRAST: a discriminative, phylogeny-free approach to multiple informant *de novo* gene prediction. *Genome Biol.* **8**, R269 (2007).
138. Bernal, A., Crammer, K., Hatzigeorgiou, A. & Pereira, F. Global discriminative learning for higher-accuracy computational gene prediction. *PLoS Comput. Biol.* **3**, e54 (2007).
139. Usuka, J., Zhu, W. & Brendel, V. Optimal spliced alignment of homologous cDNA to a genomic DNA template. *Bioinformatics* **16**, 203–211 (2000).
140. Kiryutin, B. ProSplign. *National Center for Biotechnology Information* [online], <http://www.ncbi.nlm.nih.gov/utils/static/prosplign/prosplign.html> (2011).
141. Wang, K. *et al.* MapSplice: accurate mapping of RNA-seq reads for splice junction discovery. *Nucleic Acids Res.* **38**, e178 (2010).
142. Kitts, P. in *The NCBI Handbook* (ed. McEntyre, J. & Ostell, J.) (National Center for Biotechnology Information, 2003).
143. Robinson, J. T. *et al.* Integrative genomics viewer. *Nature Biotech.* **29**, 24–26 (2011).

Acknowledgements

The authors would like to thank P. Flieck, B. Haas, N. Jiang, D. Lipman, A. Mackey, K. Pruitt, Y. Sun and J. Stajich for reading an earlier version of this manuscript and for their many helpful suggestions. This work was supported by the US National Institutes of Health grants R01GM09939 and R01-HG004694 and by the US National Science Foundation IOS-1126998 to M.Y.

Competing interests statement

The authors declare no competing financial interests.

FURTHER INFORMATION

Mark Yandell's homepage: <http://www.yandell-lab.org>
ABySS: <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2833333/>
Apollo: <http://apollo.berkeleybop.org/current/index.html>
The Arabidopsis Information Resource (TAIR): <http://www.arabidopsis.org>
Argo: <http://www.broadinstitute.org/annotation/argo>
Artemis: <http://www.sanger.ac.uk/resources/software/artemis>
Augustus: <http://bioinf.uni-grafswald.de/augustus>
BeeBase: <http://hymenopteragenome.org/beebase>
BioPerl: http://www.bioperl.org/wiki/Main_Page
BLAST: <blast.ncbi.nlm.nih.gov/Blast.cgi>
The Brent Lab software (for TwinScan): <http://mblab.wustl.edu/software.html>
CEGMA: <http://korflab.ucdavis.edu/Datasets/cegma>
CHADO: <http://gmod.org/wiki/Chado - Getting Started>
Crossmatch: <http://www.ncbi.nlm.nih.gov/blast.cgi?CMD=CrossMatch&DB=nr>
Cufflinks: <http://cufflinks.ccb.umd.edu>
EMBL: <http://www.ebi.ac.uk/help/formats.html#EMBL>
Ensembl: <http://www.ensembl.org/index.html>
Ensembl Genome Annotation: http://www.ensembl.org/info/docs/genebuild/genome_annotation.html
EVidenceModeler: <http://evidencemodeler.sourceforge.net>
EviGan: <http://www.seas.upenn.edu/~strclrn/evigan/evigan.html>
Exonerate: <http://www.genome.iastate.edu/bioinfo/resources/manuals/exonerate>
FlyBase: <http://flybase.org>

GAZE: <http://www.sanger.ac.uk/resources/software/gaze>
GBrowse: <http://gmod.org/wiki/GBrowse>
GenBank homepage: <http://www.ncbi.nlm.nih.gov/genbank>
GenBank submission guide for eukaryotic genomes: http://www.ncbi.nlm.nih.gov/genbank/eukaryotic_genome_submission
GeneMark-ES: <http://exon.gatech.edu>
GFF3: <http://sequenceontology.org/gff3.shtml>
GLEAN: <http://sourceforge.net/projects/glean-gene>
Generic Model Organism Database (GMOD) overview: <http://gmod.org/wiki/Overview>
Gnomon: <http://www.ncbi.nlm.nih.gov/genome/guide/gnomon.shtml>
GSNAP: <http://research-pub.gene.com/gsnap>
Gramene: http://www.gramene.org/genome_browser/index.html
GTF: <http://mblab.wustl.edu/GTF22.html>
Infernal: <http://infernal.janelia.org>
JBrowse: <http://jbrowse.org>
JIGSAW: <http://www.cbcb.umd.edu/software/jigsaw>
MAKER: <http://www.yandell-lab.org/software/maker.html>
Nature Reviews Genetics article series on Study designs: <http://www.nature.com/nrg/series/studydesigns/index.html>
NCBI taxonomy browser: <http://www.ncbi.nlm.nih.gov/Taxonomy/Browser/wwwtax.cgi>
PASA: <http://pasa.sourceforge.net>
Phytozome: <http://www.phytozome.net>

PlantGDB: <http://www.plantgdb.org>
qRNA: <http://selab.janelia.org/software/#qrna>
RepeatMasker: <http://www.repeatmasker.org>
Rfam: <http://rfam.sanger.ac.uk>
Saccharomyces Genome Database: <http://www.yeastgenome.org>
Scripture: <http://www.broadinstitute.org/software/scripture>
Sequence Ontology Project: <http://www.sequenceontology.org/index.html>
sim4: <http://globin.bx.psu.edu/html/docs/sim4.html>
Spidey: <http://www.ncbi.nlm.nih.gov/spidey/spideydoc.html>
Splign: <http://www.ncbi.nlm.nih.gov/utils/splign/splign.cgi>
SNAP: <http://homepage.mac.com/jankorf/SNAP.html>
Snoscan: <http://lowelab.ucsc.edu/snoscans>
SOAPdenovo: <http://soap.genomics.org.cn/soapdenovo.html>
SoftBerry: <http://www.softberry.com>
SoftBerry products (for FGENESH): <http://linux1.softberry.com/berryphthml/topic=products>
Stemloc: <http://biowiki.org/Stemloc>
TopHat: <http://tophat.cbcb.umd.edu/index.html>
Trinity: <http://trinityrnaseq.sourceforge.net>
tRNAscan-SE: <http://lowelab.ucsc.edu/tRNAscan-SE>
UniProtKB/SwissProt: <http://www.uniprot.org>
University of California Santa Cruz (UCSC) Genome Browser: <http://genome.ucsc.edu>
VectorBase: <http://www.vectorbase.org>
WormBase: <http://www.wormbase.org>

ALL LINKS ARE ACTIVE IN THE ONLINE PDF

WEEK 4 READINGS

The seahorse genome and the evolution of its specialized morphology

Qiang Lin^{1,*§}, Shaohua Fan^{2,†*}, Yanhong Zhang^{1*}, Meng Xu^{3*}, Huixian Zhang^{1,4*}, Yulan Yang^{3*}, Alison P. Lee^{4†}, Joost M. Woltering², Vydiyanathan Ravi⁴, Helen M. Gunter^{2†}, Wei Luo¹, Zexia Gao⁵, Zhi Wei Lim^{4†}, Geng Qin^{1,6}, Ralf F. Schneider², Xin Wang^{1,6}, Peiwen Xiong², Gang Li¹, Kai Wang⁷, Jiumeng Min³, Chi Zhang³, Ying Qiu⁸, Jie Bai⁸, Weiming He³, Chao Bian⁸, Xinhui Zhang⁸, Dai Shan³, Hongyue Qu^{1,6}, Ying Sun⁸, Qiang Gao³, Liangmin Huang^{1,6}, Qiong Shi^{1,8§}, Axel Meyer^{2§} & Byrappa Venkatesh^{4,9§}

Seahorses have a specialized morphology that includes a toothless tubular mouth, a body covered with bony plates, a male brood pouch, and the absence of caudal and pelvic fins. Here we report the sequencing and *de novo* assembly of the genome of the tiger tail seahorse, *Hippocampus comes*. Comparative genomic analysis identifies higher protein and nucleotide evolutionary rates in *H. comes* compared with other teleost fish genomes. We identified an astacin metalloprotease gene family that has undergone expansion and is highly expressed in the male brood pouch. We also find that the *H. comes* genome lacks enamel matrix protein-coding proline/glutamine-rich secretory calcium-binding phosphoprotein genes, which might have led to the loss of mineralized teeth. *tbx4*, a regulator of hindlimb development, is also not found in *H. comes* genome. Knockout of *tbx4* in zebrafish showed a ‘pelvic fin-loss’ phenotype similar to that of seahorses.

Members of the teleost family Syngnathidae (seahorses, pipefishes and seadragons) (Extended Data Fig. 1), comprising approximately 300 species, display a complex array of morphological innovations and reproductive behaviours. This includes specialized morphological phenotypes such as an elongated snout with a small terminal mouth, fused jaws, absent pelvic and caudal fins, and an extended body covered with an armour of bony plates instead of scales¹ (Fig. 1a). Syngnathids are also unique among vertebrates due to their ‘male pregnancy’, whereby males nourish developing embryos in a brood pouch until hatching and parturition occurs^{2,3}. In addition, members of the sub-family Hippocampinae (seahorses) exhibit other derived features such as the lack of a caudal fin, a characteristic prehensile tail, and a vertical body axis⁴ (Fig. 1a). To understand the genetic basis of the specialized morphology and reproductive system of seahorses, we sequenced the genome of the tiger tail seahorse, *H. comes*, and carried out comparative genomic analyses with the genome sequences of other ray-finned fishes (Actinopterygii).

Genome assembly and annotation

The genome of a male *H. comes* individual was sequenced using the Illumina HiSeq 2000 platform. After filtering low-quality and duplicate reads, 132.13 Gb (approximately 190-fold coverage of the estimated 695 Mb genome) of reads from libraries with insert sizes ranging from 170 bp to 20 kb were retained for assembly. The filtered reads were assembled using SOAPdenovo (version 2.04) to yield a 501.6 Mb assembly with an N50 contig size and N50 scaffold size of 34.7 kb and 1.8 Mb, respectively. Total RNA from combined soft tissues of *H. comes* was sequenced using RNA-sequencing (RNA-seq) and assembled

de novo. The *H. comes* genome assembly is of high quality, as >99% of the *de novo* assembled transcripts (76,757 out of 77,040) could be mapped to the assembly; and 243 out of 248 core eukaryotic genes mapping approach (CEGMA) genes are complete in the assembly.

We predicted 23,458 genes in the genome of *H. comes* based on homology and by mapping the RNA-seq data of *H. comes* and a closely related species, the lined seahorse, *Hippocampus erectus*, to the genome assembly (see Methods and Supplementary Information). More than 97% of the predicted genes (22,941 genes) either have homologues in public databases (Swissprot, Trembl and the Kyoto Encyclopedia of Genes and Genomes (KEGG)) or are supported by assembled RNA-seq transcripts. Analysis of gene family evolution using a maximum likelihood framework identified an expansion of 25 gene families (261 genes; 1.11%) and contraction of 54 families (96 genes; 0.41%) in the *H. comes* lineage (Extended Data Fig. 2 and Supplementary Tables 4.1, 4.2). Transposable elements comprise around 24.8% (124.5 Mb) of the *H. comes* genome, with class II DNA transposons being the most abundant class (9%; 45 Mb). Only one wave of transposable element expansion was identified, with no evidence for a recent transposable element burst (Kimura divergence ≤ 5) (Extended Data Fig. 3).

Phylogenomics and evolutionary rate

The phylogenetic relationships between *H. comes* and other teleosts were determined using a genome-wide set of 4,122 one-to-one orthologous genes (Supplementary Note 4.2). The phylogenetic analysis (Fig. 1b) showed that *H. comes* is a sister group to other percomorph fishes analysed (stickleback, *Gasterosteus aculeatus*; medaka, *Oryzias latipes*; Nile tilapia, *Oreochromis niloticus*; fugu, *Takifugu rubripes*; and

¹CAS Key Laboratory of Tropical Marine Bio-resources and Ecology, South China Sea Institute of Oceanology, Chinese Academy of Sciences, Guangzhou 510301, China. ²Chair in Zoology and Evolutionary Biology, Department of Biology, University of Konstanz, Konstanz 78457, Germany. ³BGI-Shenzhen, Shenzhen 518083, China. ⁴Institute of Molecular and Cell Biology, A*STAR, Biopolis, Singapore 138673, Singapore. ⁵College of Fisheries, Huazhong Agricultural University, Wuhan 430070, China. ⁶University of Chinese Academy of Science, Beijing 100049, China. ⁷School of Agriculture, Ludong University, Yantai 264025, China. ⁸Shenzhen Key Lab of Marine Genomics, Guangdong Provincial Key Lab of Molecular Breeding in Marine Economic Animals, BGI, Shenzhen 518083, China. ⁹Department of Paediatrics, Yong Loo Lin School of Medicine, National University of Singapore, Singapore 119228, Singapore. [†]Present addresses: Department of Genetics, University of Pennsylvania, Pennsylvania 19104, USA (S.F.); Bioprocessing Technology Institute, Biopolis, Singapore 138668, Singapore (A.P.L.); Institute of Evolutionary Biology, the University of Edinburgh EH9 3FL, UK (H.M.G.); School of Material Science and Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798, Singapore (Z.W.L.).

*These authors contributed equally to this work.

§These authors jointly supervised this work.

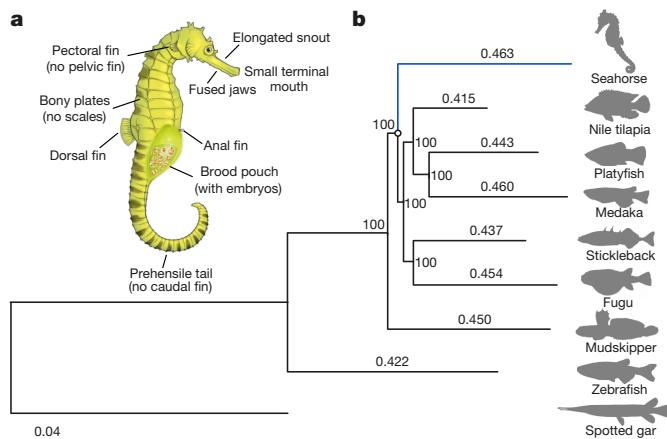


Figure 1 | Adaptations and evolutionary rate of *H. comes*. **a**, Schematic diagram of a pregnant male seahorse. **b**, The phylogenetic tree generated using protein sequences. The values on the branches are the distances (number of substitutions per site) between each of the teleost fishes and the spotted gar (outgroup). Spotted gar, *Lepisosteus oculatus*; zebrafish, *Danio rerio*.

platyfish, *Xiphophorus maculatus*) with the exception of blue-spotted mudskipper (*Boleophthalmus pectinirostris*), a member of the family Gobiidae. Our inference, which placed the mudskipper as the outgroup, differs from that of a previous phylogenetic analysis based on fewer protein-coding genes that had placed syngnathids as an outgroup⁵. Estimated divergence times of *H. comes* and other teleosts calculated using MCMCTree suggest that *H. comes* diverged from the other percomorphs approximately 103.8 million years ago, during the Cretaceous period (Extended Data Fig. 2). Interestingly, the branch length of *H. comes* is longer than that of other teleosts, suggesting a higher protein evolutionary rate compared to other teleosts analysed in this study (Fig. 1b). This result was found to be statistically significant by both relative rate test⁶ and two cluster analysis⁷ (Supplementary Tables 4.3 and 4.4). To determine whether the neutral nucleotide substitution rate of *H. comes* is also higher, we generated a neutral tree on the basis of fourfold degenerate sites and calculated the pairwise distance of each teleost to the spotted gar (an outgroup) (Supplementary Fig. 4.4). The pairwise distance of *H. comes* was again higher compared with other teleosts, indicating that the neutral evolutionary rate of *H. comes* is also higher than that of other teleosts. The reasons for this higher molecular evolutionary rate in *H. comes* are unclear.

Gene loss

Gene loss or loss of function can contribute to evolutionary novelties and can be positively selected for^{8,9}. We identified several genes that are not found in the *H. comes* genome but are found in other sequenced teleost genomes.

Secretory calcium-binding phosphoprotein (SCPP) genes encode extracellular matrix proteins that are involved in the formation of mineralized tissues such as bone, dentin, enamel and enameloid. Bony vertebrate genomes encode multiple SCPP genes that can be divided into two groups, the acidic and the proline/glutamine (P/Q)-rich SCPP genes. Acidic SCPPs regulate the mineralization of collagen scaffolds in bone and dentin whereas the P/Q-rich SCPPs are primarily involved in enamel or enameloid formation¹⁰. Analysis of the *H. comes* genome and the transcriptomes of *H. comes* and *H. erectus* showed that both contain two acidic SCPP genes, *scpp1* and *spp1* (Extended Data Fig. 4). However, no intact P/Q-rich gene could be identified. The only P/Q-rich gene present in the *H. comes* genome assembly, *scpp5*, is represented by only three out of ten exons, indicating that it has become a pseudogene. Seahorses and pipefish (family Syngnathidae) are toothless, a phenomenon known as edentulism. Besides syngnathids, edentulism has occurred convergently in several other vertebrate

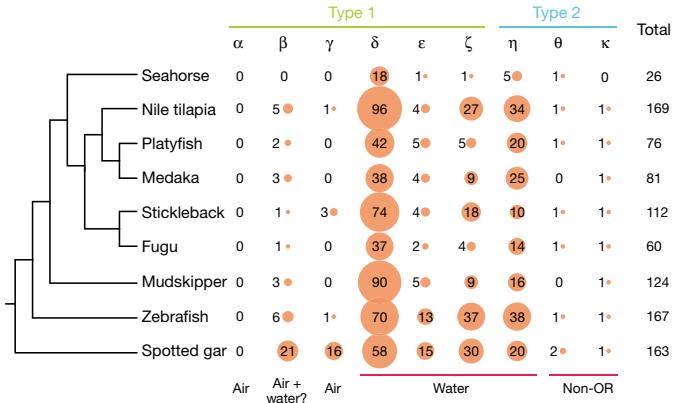


Figure 2 | OR genes in *H. comes* and other ray-finned fishes. ‘Air’ and ‘water’ refer to the detection of airborne and water-soluble odorants, respectively. The sizes of the orange circles represent the number of OR genes of a particular category.

lineages¹¹, the most notable ones being birds¹², turtles, and some mammals such as baleen whales, pangolins and anteaters¹³. The loss of teeth in birds, turtles and mammals has been attributed to inactivating mutations in one or more P/Q-rich enamel-specific SCPP genes such as *Enam*, *Amel*, *Amnb* and *Amtn*, and the dentin-specific gene, *Dspp*^{12,14}. In the case of *H. comes*, the complete loss of functional P/Q-rich SCPP genes may explain the loss of mineralized teeth.

Animals use their sense of smell, or olfaction, for finding food, mates and avoiding predators. Olfaction is mediated by olfactory receptors (ORs), which constitute the largest family of G-protein-coupled receptors. We were able to identify in the *H. comes* genome a significantly smaller repertoire of OR genes than in other teleosts (*P* value < 0.05, Wilcoxon rank-sum test). Our sensitive search pipeline (based on TblastN and Genewise) and manual inspection identified only 26 OR genes in the *H. comes* genome—the smallest OR repertoire identified in any ray-finned fish genome analysed so far (60 to 169 OR genes) (Fig. 2 and Extended Data Fig. 5).

A derived phenotype of seahorse and other syngnathids is the complete lack of pelvic fins^{15,16}. Pelvic fins are homologous to tetrapod hindlimbs and primarily serve a role in body trim and subtle swimming manoeuvres during teleost locomotion^{17–19}. In addition, pelvic spines have an important role in protection against predators¹⁵. Pelvic fin loss has occurred independently in several teleost lineages, including Tetraodontidae (for example, pufferfishes), Anguillidae (eels) and Gasterosteidae (some populations of sticklebacks), and is frequently associated with a reduced pressure from predators and/or the evolution of an elongated body plan¹⁵. In pufferfish (fugu), pelvic fin loss is associated with a change in the expression pattern of *hoxd9a*²⁰. In freshwater populations of stickleback, the loss of pelvic fins has been demonstrated to be due to deletions in the pelvic fin-specific enhancer of *pitx1* (ref. 21).

Analysis of the *H. comes* genome and the transcriptomes of *H. comes* and *H. erectus* (see Supplementary Information, section 2), suggested that *tbx4*, a transcription factor conserved in jawed vertebrates, is not present in the seahorse genome (Fig. 3a) (Supplementary Information, section 9). To verify this, we carried out degenerate polymerase chain reaction (PCR) using genomic DNA from *H. comes* and several other species of syngnathids and some non-syngnathids. While the degenerate primers amplified a fragment of *tbx4* from non-syngnathids, they failed to amplify a *tbx4* fragment from syngnathid fishes (see Supplementary Information, section 9). *Tbx4* is a T-box DNA-binding domain-containing transcription factor that acts as a regulator of hindlimb formation in mammals^{22–24}. Loss of function of this gene in mouse leads to a failure of hindlimb formation^{22,23} as well as strong pleiotropic defects in lung²⁵ and placental development²². Expression of zebrafish *tbx4* specifically in pelvic fins suggests a similar role in appendage patterning in fishes²⁴. Given the major role of *tbx4* in

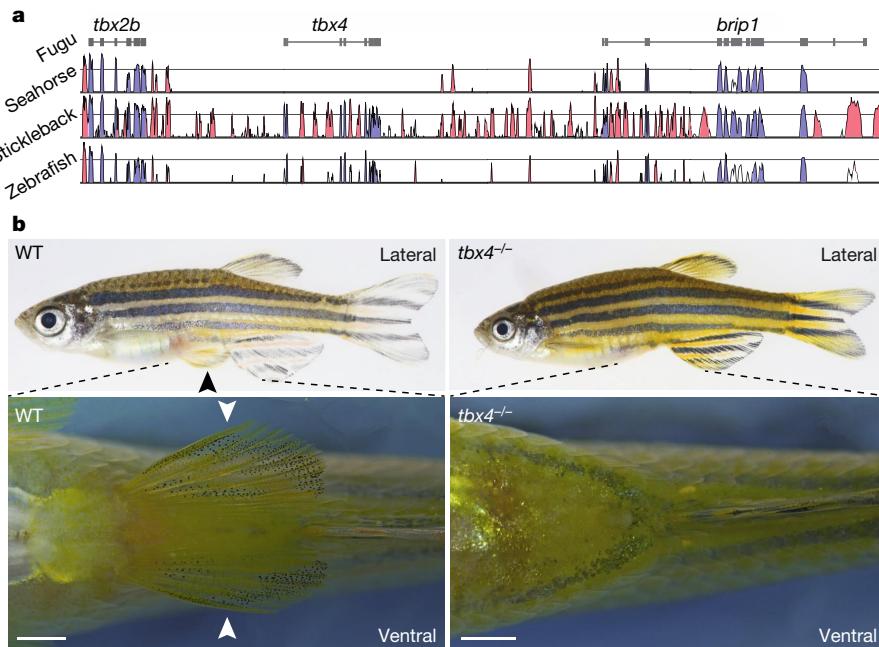


Figure 3 | Pelvic fin loss in *H. comes* is associated with loss of *tbx4*. **a**, Vista plot of conserved elements in the *tbx2b*-*tbx4*-*brip1* syntenic region in fugu (reference genome), seahorse (*H. comes*), stickleback and zebrafish showing that *tbx4* is missing from this locus in seahorse. The blue and red peaks represent conserved exonic and non-coding sequences, respectively. **b**, Lateral (top) and ventral view (bottom) of wild-type (WT) and a representative (one out of five) F3 homozygous *tbx4*-null mutant (*tbx4*^{-/-}) zebrafish. Bottom panel shows a close-up of the pelvic region (dashed lines indicate the approximate zoom region). Scale bar, 1 mm. Pelvic fins are indicated with black or white arrowheads in the wild-type fish. Homozygous *tbx4*-null mutants entirely lack pelvic fins without showing any other gross morphological defects.

hindlimb formation in mammals, we hypothesized that its absence in *H. comes* might be associated with the loss of pelvic fins. To test this hypothesis, we generated a CRISPR-Cas9 *tbx4*-knockout mutant zebrafish line. Interestingly, unlike homozygous mouse *Tbx4* mutants, which fail to develop a functional allantois²², the homozygous zebrafish mutants are viable but completely lack pelvic fins without exhibiting any other gross morphological abnormalities in pectoral or median fins (Fig. 3c and Extended Data Fig. 6; see also Supplementary Information, section 9.3, in particular Supplementary Fig. 9.6 for additional phenotype analysis). This finding is consistent with the results of a recent study that showed that mutations in *tbx4* are associated with the loss of pelvic fins in a naturally occurring zebrafish strain called *pelvic finless*²⁶ (see also Supplementary Information, section 9.3). These results show that *tbx4* has a role in pelvic fin formation in teleosts and suggests that the loss of pelvic fins in *H. comes* may be related to the loss of *tbx4*.

Expansion of the *patristacin* gene family

Male pregnancy is an evolutionary innovation unique to syngnathids. In teleosts, the C6AST subfamily of astacin metalloproteases—such as high choriolytic enzyme (HCE) and low choriolytic enzyme (LCE)—are involved in lysing the chorion surrounding the egg, leading to hatching of embryos²⁷. A member of this subfamily, *patristacin* (*pastn*), was found to be highly expressed in the brood pouch of pregnant males of the Gulf pipefish, *Syngnathus scovelli*, leading to the suggestion that this gene may have a role in the evolution of male pregnancy²⁸. A *pastn* gene was also found to be highly expressed in the brood pouch of the male big belly seahorse, *H. abdominalis*, during mid- and late pregnancy²⁹, suggesting a shared role for this gene in male pregnancy in syngnathids.

The *H. comes* genome contains six *pastn* genes (*pastn1* to *pastn6*; Fig. 4a) organized in a cluster. To examine their expression patterns in the brood pouch, we carried out RNA-seq analysis at different stages of brood pouch development (see Supplementary Information, section 2) in *H. erectus*, as this species is easy to obtain and breed in the laboratory. *H. comes* and *H. erectus* exhibit very similar reproductive cycles and their coding sequences are highly similar (average identity of 93.3%; determined by aligning *H. erectus* RNA-seq transcripts to the *H. comes* genome assembly). We identified orthologues for five of the *H. comes* *pastn* genes (*pastn1*, *pastn2*, *pastn3*, *pastn5* and *pastn6*) in the RNA-seq transcripts of *H. erectus* (Supplementary Fig. 2). Quantitative reverse transcription PCR (qRT-PCR) analysis of these

genes showed that some of them are expressed at significantly higher levels in early- and late-pregnant stages (Fig. 4c). For example, *pastn2* is expressed at significantly higher levels in early- and late-pregnant stages compared to the non-pregnant stage, whereas *pastn1* and *pastn3* are expressed at significantly higher levels during the late-pregnant stage compared to non-pregnant stage (Fig. 4c). This expression pattern suggests a role for these *pastn* genes in brood pouch development and/or hatching of embryos within the brood pouch prior to parturition.

Interestingly, the platyfish (*X. maculatus*), in which fertilization and hatching of eggs occur within the maternal body (ovoviparity), contains a cluster of six *c6ast* genes (Fig. 4a), with potential hatching enzyme-like activity³⁰. Phylogenetic analysis of *c6ast* family genes in *H. comes*, platyfish and other fishes showed that *H. comes* *pastn* genes and platyfish *c6ast* genes form separate clades (Fig. 4b), indicating that they have expanded independently in the two lineages. Thus, this is an interesting instance of a gene family (C6AST subfamily of astacin metalloproteases) that has undergone expansion independently in different teleost lineages and shows new expression patterns and functions associated with similar evolutionary innovations (that is, ovoviparity in female platyfish and male pregnancy in seahorse).

Loss of conserved noncoding elements

Vertebrate genomes contain thousands of noncoding elements that are under purifying selection^{31–33}. Many of these conserved noncoding elements (CNEs) function as *cis*-regulatory elements such as enhancers, repressors and insulators^{34,35}. Evolutionary loss of CNEs has important roles in phenotypic differences and morphological innovations^{21,36,37}. To determine the extent of loss of CNEs in seahorse, we predicted genome-wide CNEs in *H. comes* and four other percomorph fishes (stickleback, fugu, medaka and Nile tilapia) using zebrafish as the reference genome (see Supplementary Information). We identified 239,976 CNEs (average size of 168 bp) that are conserved in zebrafish and at least one of the five percomorph fishes (Supplementary Table 6.1). To determine the extent to which CNEs are lost in *H. comes*, we searched for CNEs that are uniquely lost in each of the percomorph fishes. We restricted our analyses to a high-confidence set of CNEs situated in gap-free syntenic intervals (Supplementary Table 6.5). Interestingly, *H. comes* was found to have lost a substantially higher number of CNEs (1,612 CNEs) compared to other percomorphs (fugu, 1,050 CNEs; stickleback, 843 CNEs; medaka, 335 CNEs; Nile tilapia, 281 CNEs) (Supplementary Table 6.6).

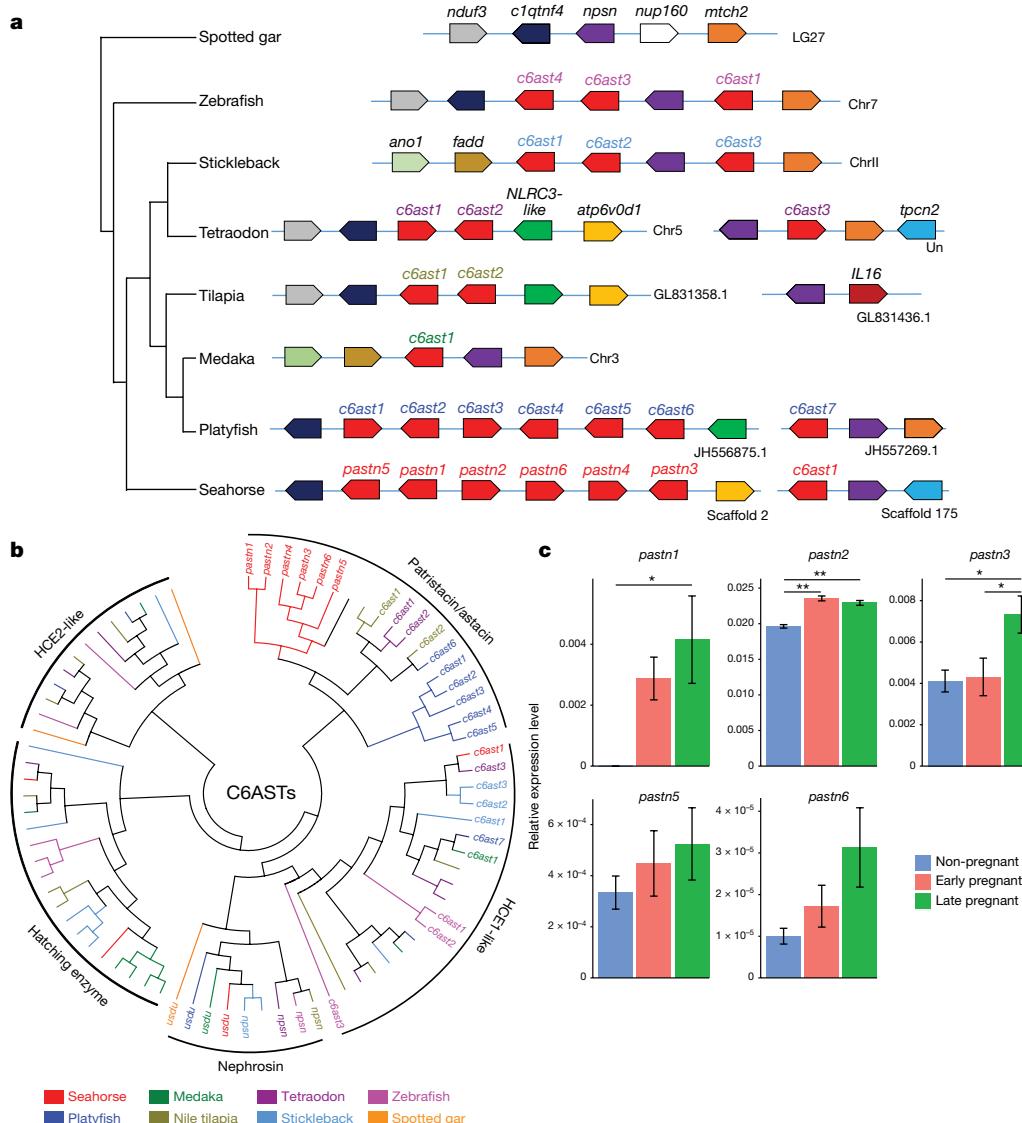


Figure 4 | Astacin metalloproteinase gene family in ray-finned fishes. **a**, Astacin gene loci in various ray-finned fish genomes showing expansion of *pastn* genes in seahorse (*H. comes*) and *c6ast* genes in platyfish. Chr, chromosome. **b**, The phylogeny of the astacin gene family in ray-finned fishes. Only *pastn* or *c6ast* genes shown in **a** are labelled. Supplementary Fig. 10.1 shows an expanded version of the tree with all the genes labelled. **c**, Expression patterns of *pastn* genes in relation to 18S ribosomal RNA genes in the brood pouch of male *H. erectus* determined by qRT-PCR. All data are expressed as mean \pm standard error of mean ($n=5$) and evaluated by one-way analysis of variance (ANOVA) followed by Tukey's honestly significant difference test for adjusting *P* values from multiple comparisons (see Methods and Supplementary Information for details of methods). The average duration of pregnancy (from fertilization to parturition) is 17 days⁴¹. The *y* axis represents expression level in relation to 18S rRNA genes. *pastn1* is expressed at low levels at the non-pregnant stage, which is not clearly visible in the figure due to the large scale used. Non-pregnant: no embryos in the brood pouch; early pregnant: 2–4 days post-fertilization; late pregnant: 12–14 days post-fertilization. **P* < 0.05, ***P* < 0.01. Note that *pastn4* is not expressed in these stages of brood pouch.

Analysis of zebrafish CNEs that are lost in *H. comes* indicated that they are present in the neighbourhood of 728 genes enriched in functions such as regulation of transcription, regulation of the fibroblast growth factor receptor signalling pathway, embryonic pectoral fin morphogenesis, steroid hormone receptor activity and O-acetyltransferase activity (Supplementary Tables 6.8 and 6.9). The top 20 genes adjacent to regions with the highest number of CNEs lost in *H. comes* include *sall1a*, *shox* and *irx5a* (Supplementary Tables 6.10 and 6.11), which are involved in the development of the limbs, nervous system, kidney, heart and skeletal system. Altered expression patterns of these genes can potentially lead to altered morphological phenotypes. For example, loss of regulatory regions of the human *SHOX* gene is the cause of Leri–Weill dyschondrosteosis, a dominantly inherited skeletal dysplasia that is characterized by moderate short stature caused by short mesomelic limb segments^{38,39}.

To verify the potential *cis*-regulatory functions of CNEs that were absent in *H. comes* but present in other teleost genomes, we assayed the function of seven selected zebrafish CNEs that were uniquely absent in *H. comes*. Of the seven CNEs assayed in transgenic zebrafish, four CNEs drove reproducible patterns of reporter gene expression in F1 embryos (Extended Data Fig. 7 and Supplementary Table 6.12). Thus, our transgenic assay indicates that some of the CNEs absent in *H. comes* may function as *cis*-regulatory elements in other teleosts. Further studies are required to examine whether the loss of CNEs may have played a role in the evolution of seahorse morphology.

Summary

Seahorses possess one of the most highly specialized morphologies and reproductive behaviours. We sequenced the genome of the tiger tail seahorse and performed comparative analysis with other teleost fishes. Our genome-wide analysis highlights several aspects that may have contributed to the highly specialized body plan and male pregnancy of seahorses. These include a higher protein and nucleotide evolutionary rate, loss of genes and expansion of gene families, with duplicated genes exhibiting new expression patterns, and loss of a selection of potential *cis*-regulatory elements. It is becoming recognized that evolutionary changes in *cis*-regulatory elements, particularly the loss and gain of enhancers, might play a major part in the evolution of morphological innovations and phenotypic changes across species^{21,36,37,40}.

Male pregnancy is a unique developmental feature of seahorses and pipefishes (family Syngnathidae, comprising 57 genera and approximately 300 species). In the seahorse genome, the astacin subfamily of *c6ast* metalloprotease genes has undergone tandem duplications giving rise to six genes. This subfamily of metalloprotease includes the hatching enzyme (also known as chorionysin), HCE-like and HCE2-like enzymes that are responsible for hatching of embryos in fishes²⁷. Of the six duplicated genes in seahorse, five are highly expressed in the male brood pouch, suggesting that they may be involved in male pregnancy, possibly through rewiring of their regulatory network. The loss of pelvic fins in seahorse is associated with the evolution of an armour-like covering of its body and gain of an

elongated, flexible, substrate-gripping tail. By combining comparative genomics and gene-knockout experiments in zebrafish, we suggest that loss of *tbx4* may have a role in this phenotype in seahorse. The loss of mineralized teeth in seahorse is associated with the fusion of the jaws into a tube-like snout and a small mouth, which is extremely efficient in sucking small food items that are abundant in the benthic environment. In teleosts, P/Q-rich SCPP genes are involved in the mineralization of enameloid, which is the equivalent of enamel in tetrapods¹⁰. The seahorse genome does not contain any intact P/Q-rich SCPP genes that code for enamel matrix proteins, suggesting that the loss of these genes could have played a part in the loss of its mineralized teeth. Our analyses of the *H. comes* genome sequence and comparative genomics with other teleosts highlighted several genetic changes that may be involved in the evolution of the unique morphology of seahorses.

Online Content Methods, along with any additional Extended Data display items and Source Data, are available in the online version of the paper; references unique to these sections appear only in the online paper.

Received 18 March; accepted 2 November 2016.

- Leysen, H. et al. Musculoskeletal structure of the feeding system and implications of snout elongation in *Hippocampus reidi* and *Dunckerocampus dactyliophorus*. *J. Fish Biol.* **78**, 1799–1823 (2011).
- Stöting, K. N. & Wilson, A. B. Male pregnancy in seahorses and pipefish: beyond the mammalian model. *BioEssays* **29**, 884–896 (2007).
- Wilson, A. B., Vincent, A., Ahnesjö, I. & Meyer, A. Male pregnancy in seahorses and pipefishes (family Syngnathidae): rapid diversification of paternal brood pouch morphology inferred from a molecular phylogeny. *J. Hered.* **92**, 159–166 (2001).
- Teske, P. R., Cherry, M. I. & Matthee, C. A. The evolutionary history of seahorses (Syngnathidae: *Hippocampus*): molecular data suggest a West Pacific origin and two invasions of the Atlantic Ocean. *Mol. Phylogenet. Evol.* **30**, 273–286 (2004).
- Near, T. J. et al. Phylogeny and tempo of diversification in the superradiation of spiny-rayed fishes. *Proc. Natl Acad. Sci. USA* **110**, 12738–12743 (2013).
- Tajima, F. Statistical method for testing the neutral mutation hypothesis by DNA polymorphism. *Genetics* **123**, 585–595 (1989).
- Nei, M. & Kumar, S. *Molecular Evolution and Phylogenetics* (Oxford Univ. Press, 2000).
- Bailly, X. et al. The loss of the hemoglobin H2S-binding function in annelids from sulfide-free habitats reveals molecular adaptation driven by Darwinian positive selection. *Proc. Natl Acad. Sci. USA* **100**, 5885–5890 (2003).
- MacArthur, D. G. et al. Loss of *ACTN3* gene function alters mouse muscle metabolism and shows evidence of positive selection in humans. *Nature Genet.* **39**, 1261–1265 (2007).
- Kawasaki, K. The SCPP gene family and the complexity of hard tissues in vertebrates. *Cells Tissues Organs* **194**, 108–112 (2011).
- Louchart, A. & Viriot, L. From snout to beak: the loss of teeth in birds. *Trends Ecol. Evol.* **26**, 663–673 (2011).
- Meredith, R. W., Zhang, G., Gilbert, M. T., Jarvis, E. D. & Springer, M. S. Evidence for a single loss of mineralized teeth in the common avian ancestor. *Science* **346**, 1254390 (2014).
- Deméré, T. A., McGowen, M. R., Berta, A. & Gatesy, J. Morphological and molecular evidence for a stepwise evolutionary transition from teeth to baleen in mysticete whales. *Syst. Biol.* **57**, 15–37 (2008).
- Zhang, G. et al. Comparative genomics reveals insights into avian genome evolution and adaptation. *Science* **346**, 1311–1320 (2014).
- Yamanoue, Y., Setiamarga, D. H. & Matsuura, K. Pelvic fins in teleosts: structure, function and evolution. *J. Fish Biol.* **77**, 1173–1208 (2010).
- Kuiter, R. H. *Seahorses and their Relatives* (Aquatic Photographics, 2009).
- Harris, J. E. The role of the fins in the equilibrium of the swimming fish. II. The role of the pelvic fins. *J. Exp. Biol.* **15**, 32–47 (1938).
- Gosline, W. A. The evolution of some structural systems with reference to the interrelationships of modern lower teleostean fish groups. *Jpn. J. Ichthyol.* **27**, 1–28 (1980).
- Standen, E. M. Pelvic fin locomotor function in fishes: three-dimensional kinematics in rainbow trout (*Oncorhynchus mykiss*). *J. Exp. Biol.* **211**, 2931–2942 (2008).
- Tanaka, M. et al. Developmental genetic basis for the evolution of pelvic fin loss in the pufferfish *Takifugu rubripes*. *Dev. Biol.* **281**, 227–239 (2005).
- Chan, Y. F. et al. Adaptive evolution of pelvic reduction in sticklebacks by recurrent deletion of a *Pitx1* enhancer. *Science* **327**, 302–305 (2010).
- Naiche, L. A. & Papaioannou, V. E. Loss of *Tbx4* blocks hindlimb development and affects vascularization and fusion of the allantois. *Development* **130**, 2681–2693 (2003).
- Rodríguez-Esteban, C. et al. The T-box genes *Tbx4* and *Tbx5* regulate limb outgrowth and identity. *Nature* **398**, 814–818 (1999).
- Tamura, K., Yonei-Tamura, S. & Izpisúa Belmonte, J. C. Differential expression of *Tbx4* and *Tbx5* in zebrafish fin buds. *Mech. Dev.* **87**, 181–184 (1999).
- Arora, R., Metzger, R. J. & Papaioannou, V. E. Multiple roles and interactions of *Tbx4* and *Tbx5* in development of the respiratory system. *PLoS Genet.* **8**, e1002866 (2012).
- Don, E. K. et al. Genetic basis of hindlimb loss in a naturally occurring vertebrate model. *Biol. Open* **5**, 359–366 (2016).
- Kawaguchi, M. et al. Evolution of teleostean hatching enzyme genes and their paralogous genes. *Dev. Genes Evol.* **216**, 769–784 (2006).
- Harlin-Cognato, A., Hoffman, E. A. & Jones, A. G. Gene cooption without duplication during the evolution of a male-pregnancy gene in pipefish. *Proc. Natl Acad. Sci. USA* **103**, 19407–19412 (2006).
- Whittington, C. M., Griffith, O. W., Qi, W., Thompson, M. B. & Wilson, A. B. Seahorse brood pouch transcriptome reveals common genes associated with vertebrate pregnancy. *Mol. Biol. Evol.* **32**, 3114–3131 (2015).
- Kawaguchi, M., Tornita, K., Sano, K. & Kaneko, T. Molecular events in adaptive evolution of the hatching strategy of ooviparous fishes. *J. Exp. Zool. B Mol. Dev. Evol.* **324**, 41–50 (2015).
- Bejerano, G. et al. Ultraconserved elements in the human genome. *Science* **304**, 1321–1325 (2004).
- Lindblad-Toh, K. et al. A high-resolution map of human evolutionary constraint using 29 mammals. *Nature* **478**, 476–482 (2011).
- Venkatesh, B. et al. Ancient noncoding elements conserved in the human genome. *Science* **314**, 1892 (2006).
- Navratilova, P. et al. Systematic human/zebrafish comparative identification of *cis*-regulatory activity around vertebrate developmental transcription factor genes. *Dev. Biol.* **327**, 526–540 (2009).
- Visel, A. et al. Ultraconservation identifies a small subset of extremely constrained developmental enhancers. *Nature Genet.* **40**, 158–160 (2008).
- Attanasio, C. et al. Fine tuning of craniofacial morphology by distant-acting enhancers. *Science* **342**, 1241006 (2013).
- McLean, C. Y. et al. Human-specific loss of regulatory DNA and the evolution of human-specific traits. *Nature* **471**, 216–219 (2011).
- Sabherwal, N. et al. Long-range conserved non-coding SHOX sequences regulate expression in developing chicken limb and are associated with short stature phenotypes in human patients. *Hum. Mol. Genet.* **16**, 210–222 (2007).
- Shears, D. J. et al. Mutation and deletion of the pseudoautosomal gene *SHOX* cause Leri–Weill dyschondrosteosis. *Nature Genet.* **19**, 70–73 (1998).
- Indjeian, V. B. et al. Evolving new skeletal traits by *cis*-regulatory changes in bone morphogenetic proteins. *Cell* **164**, 45–56 (2016).
- Lin, Q., Lin, J. & Zhang, D. Breeding and juvenile culture of the lined seahorse, *Hippocampus erectus* Perry, 1810. *Aquaculture* **277**, 287–292 (2008).

Supplementary Information is available in the online version of the paper.

Acknowledgements We thank the Laboratory Animal Center of Sun Yat-Sen University for providing experimental facilities and the Agency for Science, Technology and Research (A*STAR) Computational Resource Centre for use of its high-performance computing facilities. This research was supported by the National Science Fund for Excellent Young Scholars (41322038), the Strategic Priority Research Program of the Chinese Academy of Sciences (XDA13020103), the Youth Foundation of National High Technology Research and Development Program (863 Program) (2015AA020909), the Outstanding Youth Foundation in Guangdong Province (S2013050014802), the National Natural Science Foundation of China (41576145), the National Key Basic Research Program of China (2015CB452904), the Special Project on the Integration of Industry, Education and Research of Guangdong Province (no. 2013B090800017) and the Biomedical Research Council of A*STAR, Singapore.

Author Contributions Q.L., A.M. and B.V. designed the scientific objectives. Q.L., B.V., A.M., Q.S. and Y.Z. oversaw the project. H.Z., G.Q., B.V. and Y.Z. collected samples for sequencing DNA and RNA. M.X., Y.Y., J.M. and Q.G. performed genome sequencing, assembly and annotation. S.F., J.M. and Y.Y. performed phylogenomic analysis and molecular evolutionary rate analysis. S.F. and M.X. characterized repetitive sequences and GC content. S.F., R.F.S., P.X., V.R. and B.V. annotated and analysed Hox clusters. V.R. and B.V. annotated and analysed SCPP genes. A.P.L., V.R., Z.W.L. and B.V. performed CNE analysis and functional assay of zebrafish CNEs. Y.Z., M.X., C.Z. and D.S. assembled and annotated RNA-seq data. H.M.G. and S.F. interpreted RNA-seq results and designed the qRT-PCR experiment. Y.Z., H.Z. and X.W. performed qRT-PCR to validate the expression levels of transcripts. Y.Z., M.X., H.Z. and V.R. analysed the *patristacin* gene family. Y.Z., Q.L., J.M.W., R.F.S. and A.M. performed *tbx4* knockout analysis. Y.Q., J.B., C.B., Y.S. and X.Z. were involved in data analysis. L.H., G.L., W.L., Z.G., K.W. and H.Q. participated in the discussions related to data analysis. Q.L., Y.Z., S.F., H.M.G., A.M. and B.V. wrote the manuscript with input from all other authors.

Author Information Reprints and permissions information is available at www.nature.com/reprints. The authors declare no competing financial interests. Readers are welcome to comment on the online version of the paper. Correspondence and requests for materials should be addressed to Q.S. (shiqiong@genomics.cn), A.M. (axel.meyer@uni-konstanz.de) or B.V. (mcbbv@imcb.a-star.edu.sg).

Reviewer Information *Nature* thanks C. Amemiya, S. Burgess, K. Worley and the other anonymous reviewer(s) for their contribution to the peer review of this work.

 This work is licensed under a Creative Commons Attribution 4.0 International (CC BY 4.0) licence. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in the credit line; if the material is not included under the Creative Commons licence, users will need to obtain permission from the licence holder to reproduce the material. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

METHODS

Genome sequencing and assembly. Genomic DNA of a single male *H. comes* was used to construct eleven libraries including short-insert (170 bp, 500 bp, 800 bp) and mate-paired (2 kb, 5 kb, 10 kb, 20 kb) libraries and sequenced on the Illumina HiSeq 2000 sequencing platform. In total, we obtained around 218 Gb of raw sequence data (Supplementary Table 1.1). The genome was assembled using SOAPdenovo2.04 (ref. 42) with default parameters. No statistical methods were used to predetermine sample size. The experiments were not randomized. The investigators were not blinded to allocation during experiments and outcome assessment.

RNA sequencing and analysis. In total, 19 RNA-seq libraries were constructed, including two libraries from combined soft tissues (brain, gills, intestine, liver and muscle) from a male and a female *H. comes* (Supplementary Table 2.1); and 17 libraries of five developmental stages of embryos and different stages of brood pouch development such as the juvenile stage, rudimentary stage, pre-pregnancy stage, pregnancy stage, and post pregnancy stage, using RNA from the lined seahorse (*Hippocampus erectus*) (Supplementary Information, section 2). All libraries were prepared using Illumina TruSeq RNA sample preparation kit according to the manufacturer's instructions (Illumina, San Diego, CA, USA) and sequenced using Illumina HiSeq 2000 platform. The RNA-seq reads were either *de novo* assembled using Trinity⁴³ or mapped to the *H. comes* genome using TopHat⁴⁴ with default parameters, and subsequently analysed using in-house Perl scripts. The differential expression of genes at different stages of brood pouch development was determined using the method developed previously⁴⁵. The RNA-seq results were validated using qRT-PCR, with five biological replicates for each stage. All data were expressed as mean \pm standard error of mean and were evaluated by one-way ANOVA followed by Tukey's honestly significant difference test for adjusting *P* values from multiple comparisons. Results were considered to be statistically significant for *P* values < 0.05 .

Genome annotation. Annotation of the *H. comes* genome was carried out using the Ensembl gene annotation pipeline which integrated *ab initio* gene predictions and evidence-based gene models. Briefly, protein sequences of *D. rerio*, *G. aculeatus*, *O. latipes*, *T. rubripes* and *T. nigroviridis* were downloaded from Ensembl (release 75) and mapped to the genome using TblastN⁴⁶ with the parameter “-evalue 1E-5”. Second, high scoring segment pairs (HSPs) from blast were concatenated using Solar (in-house software, version 0.9.6). Third, the concatenated segments were aligned using GeneWise⁴⁷ to refine the gene models. Finally, we filtered the alignments that showed alignment rates less than 50% of the full-length copies and filtered redundant alignments based on the GeneWise score. In addition, *H. comes* transcripts (female_transcript and male_transcript) and *H. erectus* transcripts (Juv_brain, Juv_body, Rud_testis and PreP_pouch) were used to assist in the gene model prediction. We annotated the predicted gene models using Swiss-Prot, TrEMBL, NCBI NR database, and KEGG databases (Supplementary Table 3.4).

Expansion and contraction of gene families. We used CAFE (version 2.1), a program for analysing gene family expansion and contraction under maximum likelihood framework. The gene family results from TreeFam pipeline and the estimated divergence time between species were used as inputs. We used the parameters “-p 0.01, -r 10000, -s” to search the birth and death parameter (λ) of genes, calculated the probability of each gene family with observed sizes using 10,000 Monte Carlo random samplings, and reported birth and death parameters in gene families with probabilities less than 0.01. For the gene family expansion and contraction analysis in *H. comes*, we first filtered out gene families without homology in the SWISS-PROT database to reduce the potential false positive expansions or contractions caused by gene prediction. The families that contained sequences that have multiple functional annotations were also removed (Supplementary Tables 4.1 and 4.2).

Phylogenetic analysis. We obtained 4,122 one-to-one orthologous genes from the gene family analysis (Supplementary Information, section 4.1). The protein sequences of one-to-one orthologous genes were aligned using MUSCLE⁴⁸ with the default parameters. We then filtered the saturated sites and poorly aligned regions using trimAl (ref. 49) with the parameters “-gt 0.8 -st 0.001 -cons 60”. After trimming the saturated sites and poorly aligned regions in the concatenated alignment, 2,128,000 amino acids were used for the phylogenomic analysis. The trimmed protein alignments were used as a guide to align corresponding coding sequences (CDSs). The aligned protein and the fourfold degenerate sites in the CDSs were each concatenated into a super gene using an in-house Perl script.

The phylogenomic tree was reconstructed using RAxML version 8.1.19 (ref. 50) based on concatenated protein sequences. Specifically, we used the PROTGAMMAAUTO parameter to select the optimal amino acid substitution model, specified spotted gar as the outgroup, and evaluated the robustness of the result using 100 bootstraps. To compare the neutral mutation rate of different species, we also generated a phylogeny based on fourfold degenerate sites. The phylogenomic topology was used as input and the “-f e” option in RAxML was used to optimize the branch lengths of the input tree using the alignment of fourfold degenerate sites under the general time reversible (GTR) model as suggested by ModelGenerator

version 0.85 (ref. 51). We calculated the pairwise distances to the outgroup (spotted gar) based on the optimized branch length of the neutral tree using the cophenetic.phylo module in the R-package APE⁵². The Bayesian relaxed-molecular clock (BRMC) method, implemented in the MCMCTree program⁵³, was used to estimate the divergence time between different species. The concatenated CDS of one-to-one orthologous genes and the phylogenomics topology were used as inputs. Two calibration time points based on fossil records, *O. latipes*-*T. nigroviridis* (~96.9–150.9 million years ago (Mya)), and *D. rerio*-*G. aculeatus* (~149.85–165.2 Mya) (<http://www.fossilrecord.net/dateclade/index.html>), were used as constraints in the MCMCTree estimation. Specifically, we used the correlated molecular clock and REV substitution model in our calculation. The MCMC process was run for 5,000,000 steps and sampled every 5,000 steps. MCMCTree suggested that *H. comes* diverged from the common ancestor of stickleback, Nile tilapia, platyfish, fugu, and medaka approximately 103.8 Mya, which corresponds to the Cretaceous period.

Analysis of OR genes. We downloaded protein sequences of 1,417 OR gene family members from NCBI and mapped them to *H. comes* genome using Tblastn with “E-value $\leq 1e-10$ ” and “alignment rate ≥ 0.5 ”. Solar (in-house software, version 0.9.6) was used to join high-scoring segment pairs (HSPs) between each pair of protein mapping results. We retained alignments with an alignment rate of more than 70% and a mapping identity of more than 40%. Subsequently, the protein sequences were mapped to the genome using GeneWise and extended 280 bp upstream and downstream to define integrated gene models. For phylogenetic analysis, protein sequences were aligned using MUSCLE and a JTT+gamma model was used in a maximum-likelihood analysis using PhyML to construct a phylogenetic tree.

Evidence for loss of *tbx4* in *H. comes*. The synteny analysis of *tbx2b*-*tbx4*-*brip1* region of *H. comes*, stickleback, fugu and zebrafish using Vista shows that *tbx4* was lost in *H. comes* (Fig. 3). To exclude the scenario that the absence of *tbx4* in the *H. comes* genome sequence is due to an assembly error, we first validated the micro-synteny region of *tbx2b*-*tbx4*-*brip1* region in *H. comes* using a PCR-based genomic walk strategy. Briefly, 28 primer pairs (Supplementary Table 9.1) were designed for overlapping amplicons to ‘walk’ from the end of *tbx2b* to the start of *brip1*. Amplicon size and partial end sequencing of these products did not indicate any anomalies in the assembly of the *H. comes* *tbx4* ‘ghost locus’.

In addition, we carried out the following analyses: (1) searched the *H. comes* genome (TblastN) using Tbx4 protein from zebrafish and Nile tilapia and were unable to find a *tbx4* gene; (2) searched the *H. comes* genome using only the domain sequence of Tbx4 protein but were unable to find a *tbx4* gene; (3) searched *H. comes* and *H. erectus* transcriptome data for *tbx4* (TblastN) using Tbx4 protein from zebrafish and Nile tilapia but were unable to find any matching transcript; (4) searched *H. comes* and *H. erectus* transcriptome data with the domain sequence as well and did not find any remnant of a *tbx4* gene; and (5) predicted CNEs in the ‘ghost’ *tbx4* locus of *H. comes* using the fugu *tbx4* locus as the reference (base) (Supplementary Fig. 9.3). We used the CNEs present in the other fish genome loci (that were absent in *H. comes*) to search the *H. comes* genome to rule out the possibility that they may be present elsewhere in the genome. We were unable to find any of these CNEs in the *H. comes* genome. Finally, we conducted degenerate PCR experiments to ascertain if the *tbx4* gene is missing in *H. comes*. Using a combination of four forward and two reverse primers (Supplementary Table 9.1), we checked for the presence of *tbx4* in seven species of *Hippocampus* (including *H. comes* and *H. erectus*), five species of pipefish (four from the genus *Syngnathus* and one species of *Corythoichthys*) (all from the family Syngnathidae that lack pelvic fins); ghost pipefish (*Solenostomus*) and the trumpetfish (*Aulostomidae*) which are closely related to the Syngnathidae but possess pelvic fins; and five other teleost species that possess pelvic fins (Supplementary Figs 9.1 and 9.2).

Generation of mutant *tbx4* zebrafish. We used a CRISPR-Cas9 strategy to generate a *tbx4* mutant zebrafish line. Two guide RNAs (gRNAs) were designed targeting zebrafish *tbx4* in the 5' end of the sequence that is upstream of or within the DNA-binding TBOX domain (Supplementary Fig. 9.4). gRNAs were cloned using synthesized oligonucleotides into the pT7gRNA vector as described previously⁵⁴ (oligonucleotide sequences given in Supplementary Table 9.2). gRNAs were synthesized from this vector after linearization with BamH1-HF (NEB R3136T), transcribed using the MEGAscript T7 Transcription Kit (Thermo Fischer Scientific AM1334) and purified using the mirVana miRNA isolation kit (Thermo Fischer Scientific AM1560). Cas9 mRNA was synthesized from the Cs2+Cas9 vector using the mMessage mMachine Sp6 Transcription Kit (Thermo Fischer Scientific AM1340) and purified using the RNA cleanup protocol from the RNAeasy mini kit (Qiagen 74104).

Zebrafish from a wild caught strain were injected at the one-cell stage with ~ 50 ng gRNA and ~ 90 ng Cas9 RNA. These F0 fish were raised to maturity and genotyped using fin clipping, DNA isolation and PCR spanning the target site (genotyping primers given in Supplementary Table 9.2). PCR products were analysed for mutations as described previously⁵⁴ using T7 endonuclease (NEB M0302L). Mosaic mutant F0 fish were outcrossed to AB wild-type fish and embryos were batch genotyped for transmission of the mutation using PCR and T7 endonuclease. Mutant PCR products were cloned into the pGEM-T vector

(Promega, Madison, WI) and sequenced to identify carrier fish transmitting a frameshift mutation. These carrier fish were crossed again to AB wild type and the resulting F1 fish were raised to maturity. The F1 were genotyped using fin clipping, DNA isolation, PCR, T7 endonuclease to identify heterozygous mutant fish followed by cloning and sequencing of the mutant PCR products to validate presence of the frameshift allele. The CRISPR–Cas9 mutation strategy is schematically shown in Extended Data Fig. 5.

In the F0 mutant *tbx4* fish we observed pelvic fin loss at low frequency, gRNA#1 gave 3/42 fish with either double- or single-sided pelvic fin loss whereas 1/34 had single-sided pelvic fin loss for gRNA#2 (Extended Data Fig. 5). We observed mutant allele transmission for both gRNA#1 and gRNA#2 but failed to identify a deletion leading to a frameshift mutation for gRNA#2 so no stable line was generated for this CRISPR. For gRNA#1 we identified several frameshift mutants, one of which was further analysed. This mutant has a deletion/replacement mutation in which eight nucleotides are replaced by three nucleotides, leading to an effective 5 bp deletion and the introduction of a frameshift mutation (Extended Data Fig. 5). This mutation introduces a downstream STOP codon leading to a severely truncated protein lacking the DNA binding domain (Supplementary Figs 9.4 and 9.5). The mutant line is maintained on an AB wild-type background. **Loss of CNEs.** Using zebrafish as the reference genome, whole-genome alignments of six teleost fishes were generated. The soft-masked genome sequence for zebrafish (Zv9, April 2010) was downloaded from the Ensembl release-75 FTP site. The following soft-masked genome sequences were downloaded from the UCSC Genome Browser: stickleback (gasAcu1, February 2006), fugu (fr3, October 2011), medaka (oryLat2, October 2005), Nile tilapia (oreNil2, February 2012). The *H. comes* genome sequence (hipCom0) was repeat-masked using WindowMasker (from NCBI BLAST+ package v.2.2.28) with additional parameter “-dust true”. About 32% (158.1/501.6 Mb) of the *H. comes* genome was masked using this method.

Only chromosome sequences of zebrafish were aligned while unplaced scaffolds were excluded. The reference (zebrafish) genome was split into 21 Mb sequences with 10-kb overlap, while the percomorph fish genomes (*H. comes*, stickleback, fugu, medaka and Nile tilapia) were split into 10 Mb sequences with no overlap. Pairwise alignments were carried out using Lastz v.1.03.54 (ref. 55) with the following parameters: -strand = both –seed = 12of19 –notransition –chain –gapped-gap = 400,30 –hsptthresh = 3000 –gappedthresh = 3000 –inner = 2000 –masking = 50 –ydrop = 9400 –scores = HoxD55.q –format = axt. Coordinates of split sequences were restored to genome coordinates using an in-house Perl script. The alignments were reduced to single coverage with respect to the reference genome using UCSC Genome Browser tools ‘axtChain’ and ‘chainNet’. Multiple alignments were generated using Multiz.v11.2.roast.v3 (ref. 56) with the tree topology “(Zv9 (hipCom0 ((fr3 gasAcu1) (oryLat2 oreNil2))))”.

Fourfold degenerate (4D) sites of zebrafish genes (Ensembl release-75) were extracted from the multiple alignments. These 4D sites were used to build a neutral model using PhyloFit in the rphast v.1.5 package⁵⁷ (general reversible “REV” substitution model). PhastCons was then run in rho-estimation mode on each of the zebrafish chromosomal alignments to obtain a conserved model for each chromosome. These conserved models were averaged into one model using PhyloBoot. Subsequently, conserved elements were predicted in the multiple alignments using PhastCons with the following inputs and parameters: the neutral and conserved models, target coverage of input alignments = 0.3 and average length of conserved sequence = 45 bp. To assess the sensitivity of this approach in identifying functional elements, the PhastCons elements were compared against zebrafish protein-coding genes. Eighty per cent of protein-coding exons (197,508/245,556 exons) were overlapped by a conserved element (minimum coverage 10%), indicating that the identification method was fairly sensitive.

A CNE was considered present in a percomorph genome if it showed coverage of at least 30% with a zebrafish CNE in Multiz alignment. To identify CNEs that could have been missed in the Multiz alignments due to rearrangements in the genomes, or due to partitioning of the CNEs among teleost fish duplicate genes, we searched the zebrafish CNEs against the genome of the percomorph using BLASTN ($E < 1 \times 10^{-10}$; $\geq 80\%$ identity; $\geq 30\%$ coverage). Those CNEs that had no significant match in a percomorph genome were considered as missing in that genome. To account for CNEs that might have been missed due to sequencing gaps, we identified gap-free synteny intervals in zebrafish and the percomorph genomes, and generated a set of CNEs that were missing from these intervals. These CNEs represent a high-confidence set of CNEs missing in the percomorph fishes and thus were used for further analysis. Functional enrichment of genes associated with CNEs was carried out using the GREAT software⁵⁸ with each CNE assigned to the genes with the nearest transcription start site and within 1 Mb in the zebrafish genome, and significantly enriched functional categories identified based on a hypergeometric test of genomic regions (false discovery rate (FDR) q value < 0.05). We identified the statistically significant gene ontology biological process terms, molecular function terms and zebrafish phenotype descriptions of the genes that are associated with CNEs.

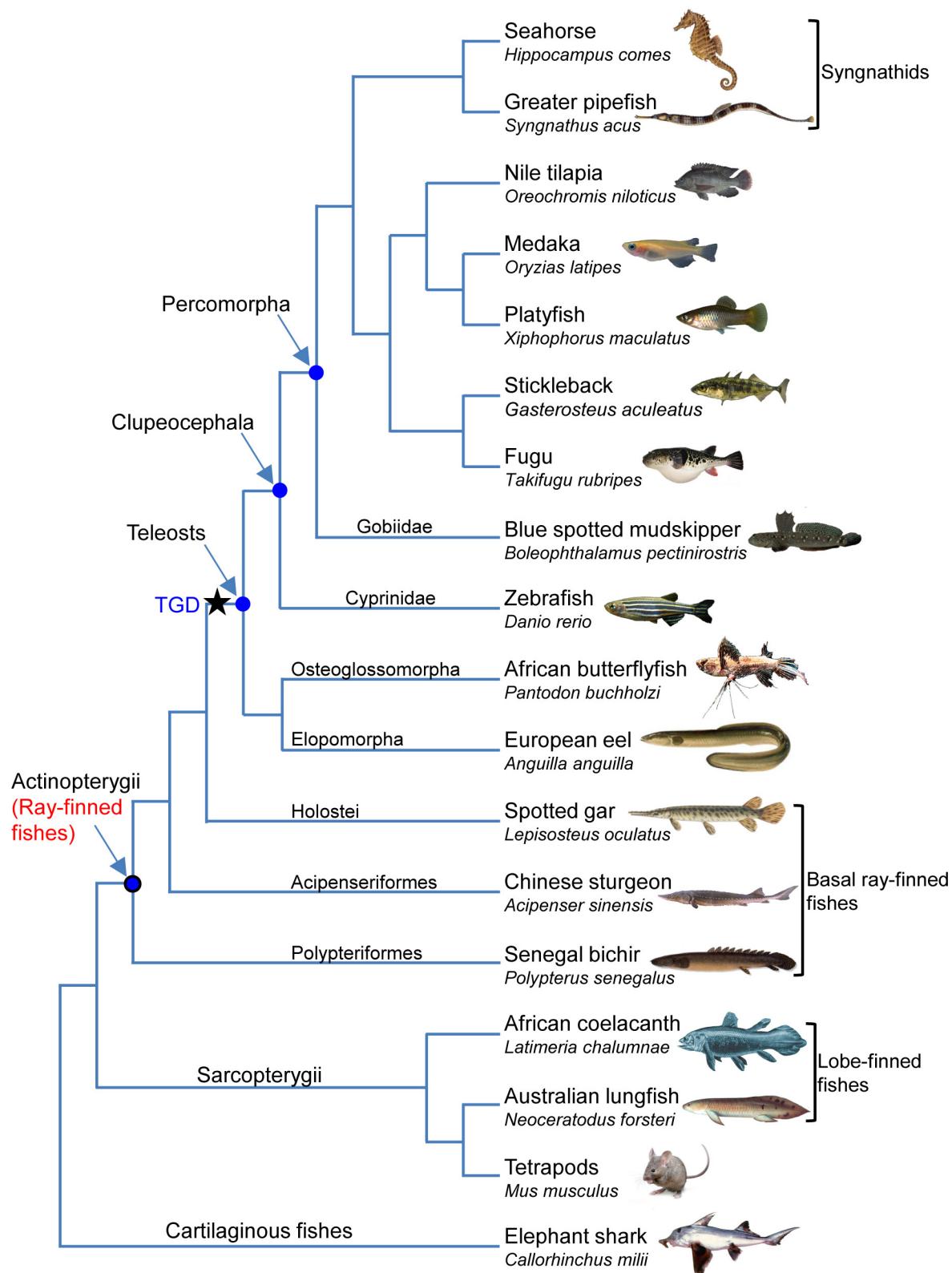
We also predicted CNEs in the Hox clusters of *H. comes* and other representative teleost fishes using the global alignment program MLAGAN. Orthologous Hox clusters were aligned using MLAGAN with zebrafish as the reference sequence and CNEs were predicted using VISTA.

Functional assay of CNEs. Seven representative zebrafish CNEs that have been lost in *H. comes* (the largest among the lost CNEs) were assayed for enhancer activity in transgenic zebrafish using GFP as the reporter gene. The CNEs were amplified by PCR using zebrafish genomic DNA as template. The products were cloned into a miniTol2 transposon donor plasmid linked to the mouse *cFos* (McFos) basal promoter and the coding sequence of GFP. Transposase mRNA was generated by transcribing cDNA *in vitro* using the mMESSAGE mMACHINE T7 kit (Ambion; Life Technologies). The CNE-containing McFos-miniTol2 construct and transposase mRNA were co-injected into the yolk of zebrafish embryos at the one to two-cell stage. Each CNE construct was injected into 250–350 embryos and the injections were repeated on two days. The embryos were reared at 28 °C, and GFP was observed at 24, 48 and 72 h post-fertilization (hpf). The survival rate of the embryos post-injection was 70–80%. Consistent GFP expression in at least 20% of F0 embryos was considered as specific expression driven by a CNE. Such embryos were reared to maturity and mated with wild type zebrafish to produce F1 lines. The expression of GFP in F1 embryos was observed under a compound microscope fitted for epifluorescence (Axio imager M2; Carl Zeiss, Germany) and photographed using an attached digital microscope camera (Axiocam; Carl Zeiss, Germany). Pigmentation was inhibited by maintaining zebrafish embryos in 0.003% N-phenylthiourea (Sigma-Aldrich, Sweden) from 8 hpf onwards. Consistent GFP expression observed in at least three lines of F1 fishes was considered as the specific expression driven by a CNE.

All animals were cared for in strict accordance with National Institutes of Health (USA) guidelines. The zebrafish gene knockout protocol was approved by the Institutional Animal Care and Use Committee of Sun Yat-Sen University. The zebrafish transgenic assay protocol was approved by the Institutional Animal Care and Use Committee of Biological Resource Centre, A*STAR, Singapore.

Data availability statement. The tiger tail seahorse (*H. comes*) whole-genome sequence has been deposited in the DDBJ/EMBL/GenBank database under accession number LVHJ00000000. RNA-seq reads for *H. erectus* and *H. comes* have been deposited in the NCBI Sequence Read Archive under accession numbers SRA392578 and SRA392580, respectively.

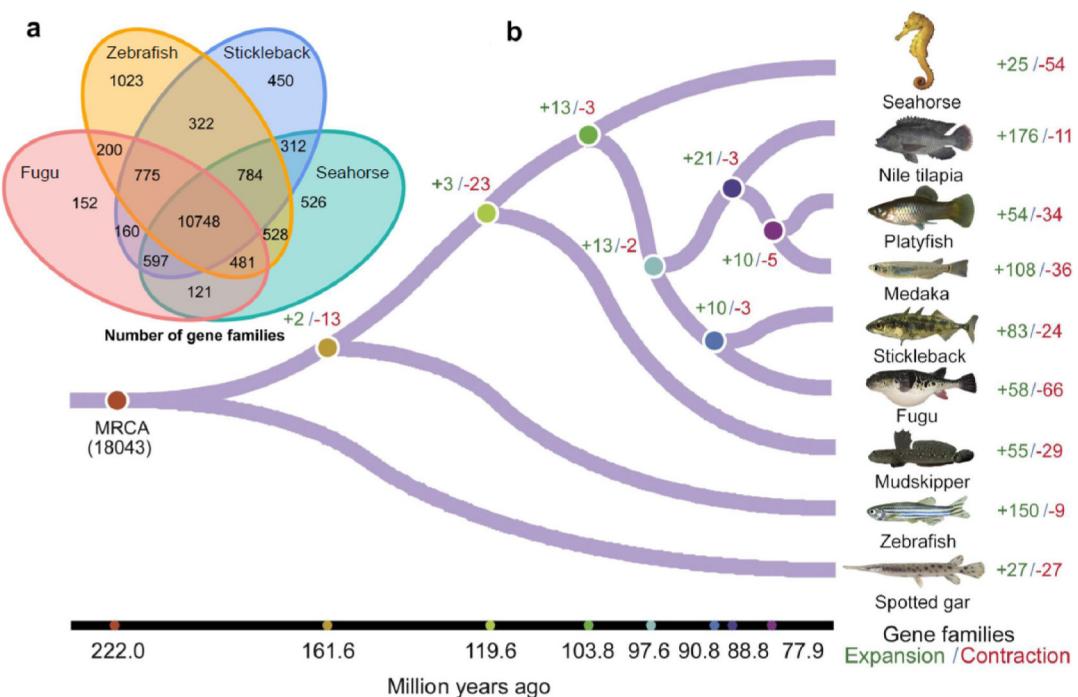
42. Luo, R. et al. SOAPdenovo2: an empirically improved memory-efficient short-read *de novo* assembler. *Gigascience* **1**, 18 (2012).
43. Grabherr, M. G. et al. Full-length transcriptome assembly from RNA-seq data without a reference genome. *Nature Biotechnol.* **29**, 644–652 (2011).
44. Trapnell, C., Pachter, L. & Salzberg, S. L. TopHat: discovering splice junctions with RNA-seq. *Bioinformatics* **25**, 1105–1111 (2009).
45. Yu, X., Lin, J., Zack, D. J. & Qian, J. Computational analysis of tissue-specific combinatorial gene regulation: predicting interaction between transcription factors in human tissues. *Nucleic Acids Res.* **34**, 4925–4936 (2006).
46. Kent, W. J. BLAT—the BLAST-like alignment tool. *Genome Res.* **12**, 656–664 (2002).
47. Birney, E., Clamp, M. & Durbin, R. GeneWise and Genomewise. *Genome Res.* **14**, 988–995 (2004).
48. Edgar, R. C. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res.* **32**, 1792–1797 (2004).
49. Capella-Gutiérrez, S., Silla-Martínez, J. M. & Gabaldón, T. trimAI: a tool for automated alignment trimming in large-scale phylogenetic analyses. *Bioinformatics* **25**, 1972–1973 (2009).
50. Stamatakis, A. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics* **22**, 2688–2690 (2006).
51. Stamatakis, A., Hoover, P. & Rougemont, J. A rapid bootstrap algorithm for the RAxML Web servers. *Syst. Biol.* **57**, 758–771 (2008).
52. Paradis, E., Claude, J. & Strimmer, K. APE: Analyses of phylogenetics and evolution in R language. *Bioinformatics* **20**, 289–290 (2004).
53. Yang, Z. PAML 4: phylogenetic analysis by maximum likelihood. *Mol. Biol. Evol.* **24**, 1586–1591 (2007).
54. Jao, L. E., Wente, S. R. & Chen, W. Efficient multiplex biallelic zebrafish genome editing using a CRISPR nuclease system. *Proc. Natl Acad. Sci. USA* **110**, 13904–13909 (2013).
55. Harris, R. S. *Improved Pairwise Alignment of Genomic DNA*. PhD thesis, Pennsylvania State Univ. (2007).
56. Blanchette, M. et al. Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Res.* **14**, 708–715 (2004).
57. Hubisz, M. J., Pollard, K. S. & Siepel, A. PHAST and RPHAST: phylogenetic analysis with space/time models. *Brief. Bioinform.* **12**, 41–51 (2011).
58. McLean, C. Y. et al. GREAT improves functional interpretation of cis-regulatory regions. *Nature Biotechnol.* **28**, 495–501 (2010).
59. Bian, C. et al. The Asian arowana (*Scleropages formosus*) genome provides new insights into the evolution of an early lineage of teleosts. *Sci. Rep.* **6**, 24501 (2016).
60. Kawasaki, K. & Amemiya, C. T. SCPP genes in the coelacanth: tissue mineralization genes shared by sarcopterygians. *J. Exp. Zool. B Mol. Dev. Evol.* **322**, 390–402 (2014).
61. Venkatesh, B. et al. Elephant shark genome provides unique insights into gnathostome evolution. *Nature* **505**, 174–179 (2014).



Extended Data Figure 1 | Phylogenetic relationships of ray-finned fishes discussed in this study. Phylogenetic relationships of ray-finned fishes depicted here are based on the current study and ref. 59.

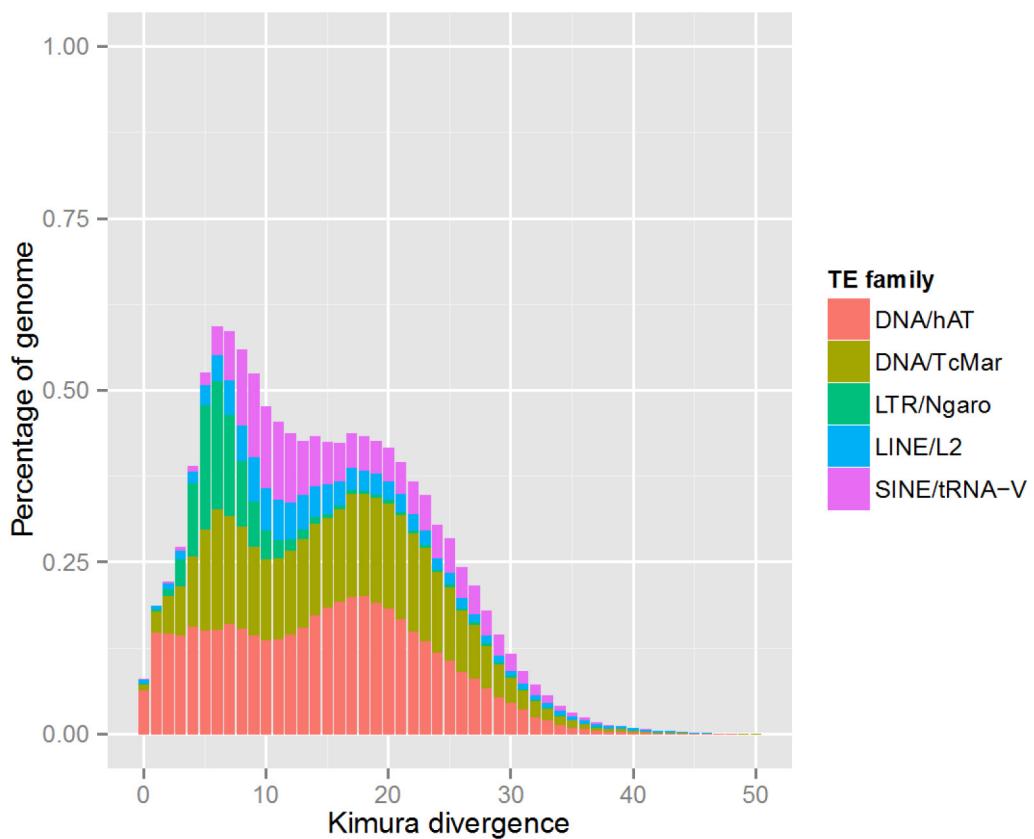
Ray-finned fishes (Actinopterygii) are divided into basal ray-finned fishes (Polypteriformes, Acipenseriformes and Holostei) and teleosts.

The latter comprise ~99% of the extant ray-finned fishes. The star represents the teleost-specific genome duplication (TGD) event that occurred in the common ancestor of all teleost fishes. Syngnathids (seahorse and pipefish) display the unique phenomenon of 'male pregnancy'.

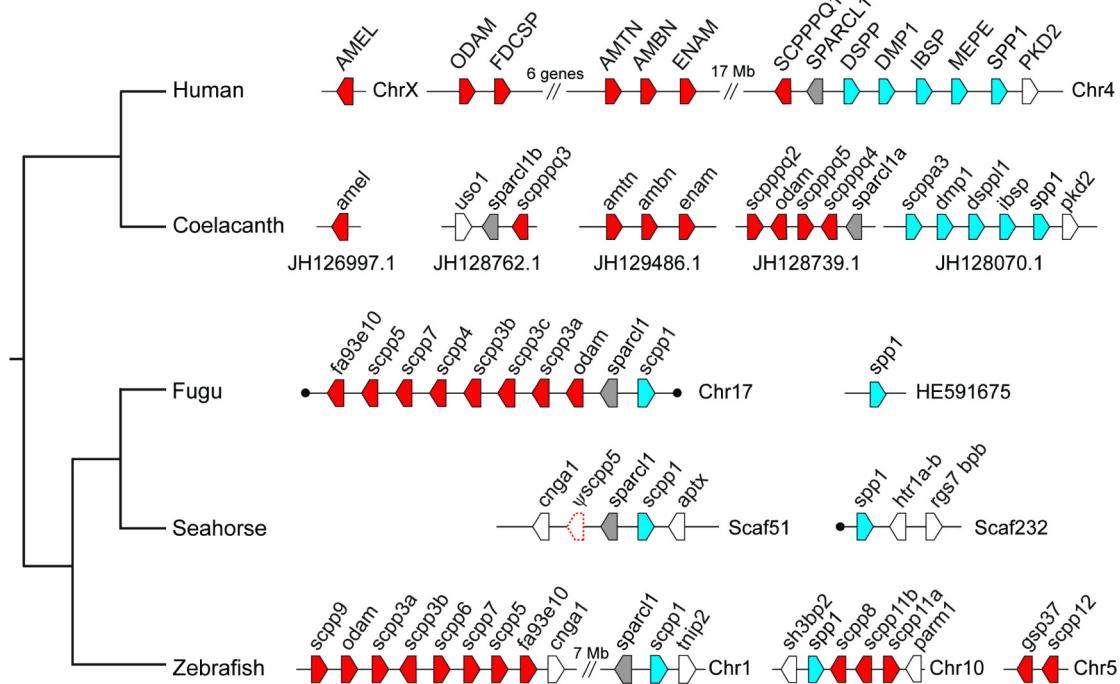


Extended Data Figure 2 | Number of gene families in various teleosts and the spotted gar. **a**, Venn diagram of shared orthologous gene families in seahorse (*H. comes*), fugu, zebrafish and stickleback. **b**, The phylogeny and divergence times of seahorse and other teleost fishes based on analysis

of genome-wide one-to-one orthologous protein sequences. The numbers at nodes indicate the number of gene families expanded and contracted at different evolutionary time points.

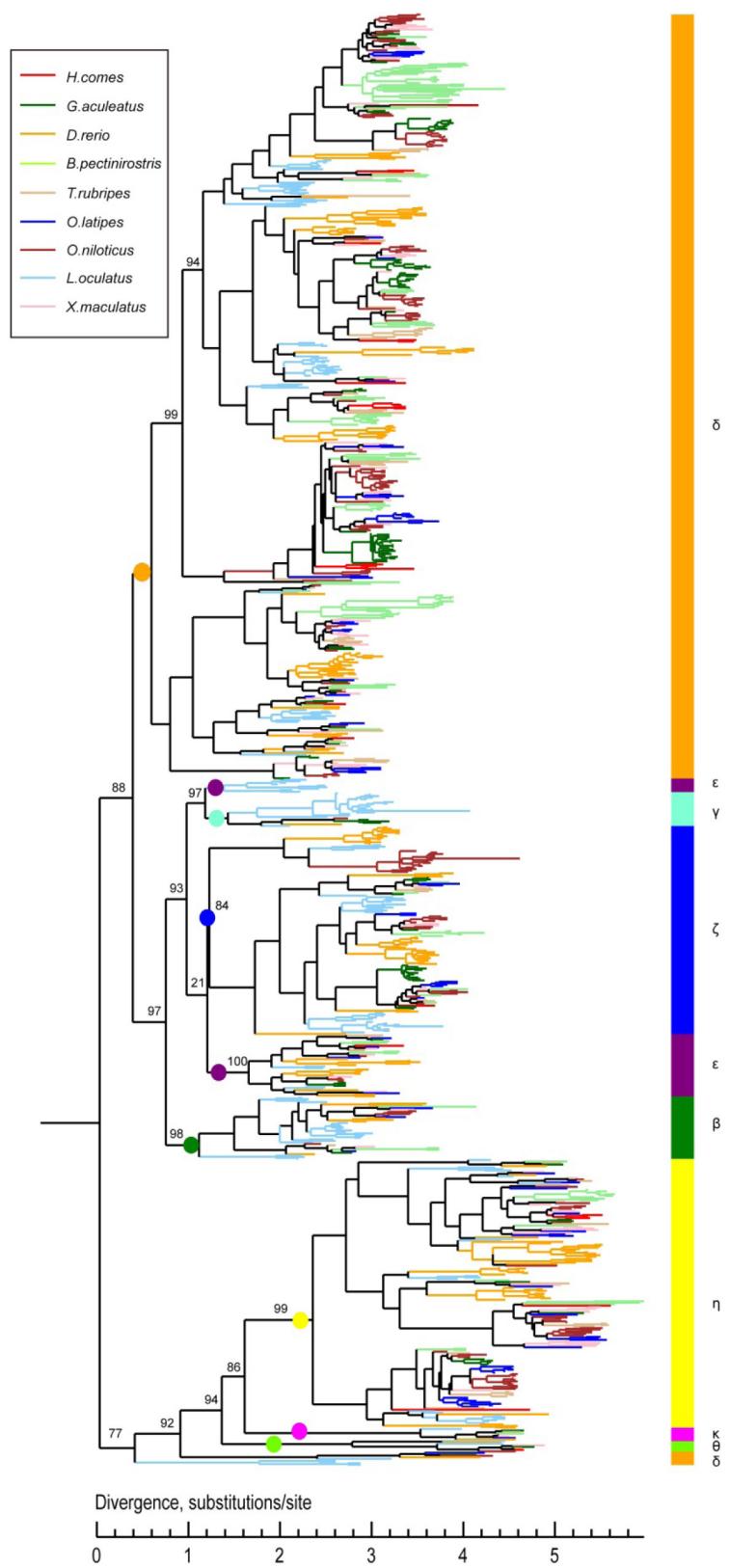


Extended Data Figure 3 | Divergence distribution of transposable elements compared to consensus in the transposable element library.
The divergence rate was calculated between the identified transposable elements (TEs) in the *H. comes* genome and the consensus sequence in the transposable element library.

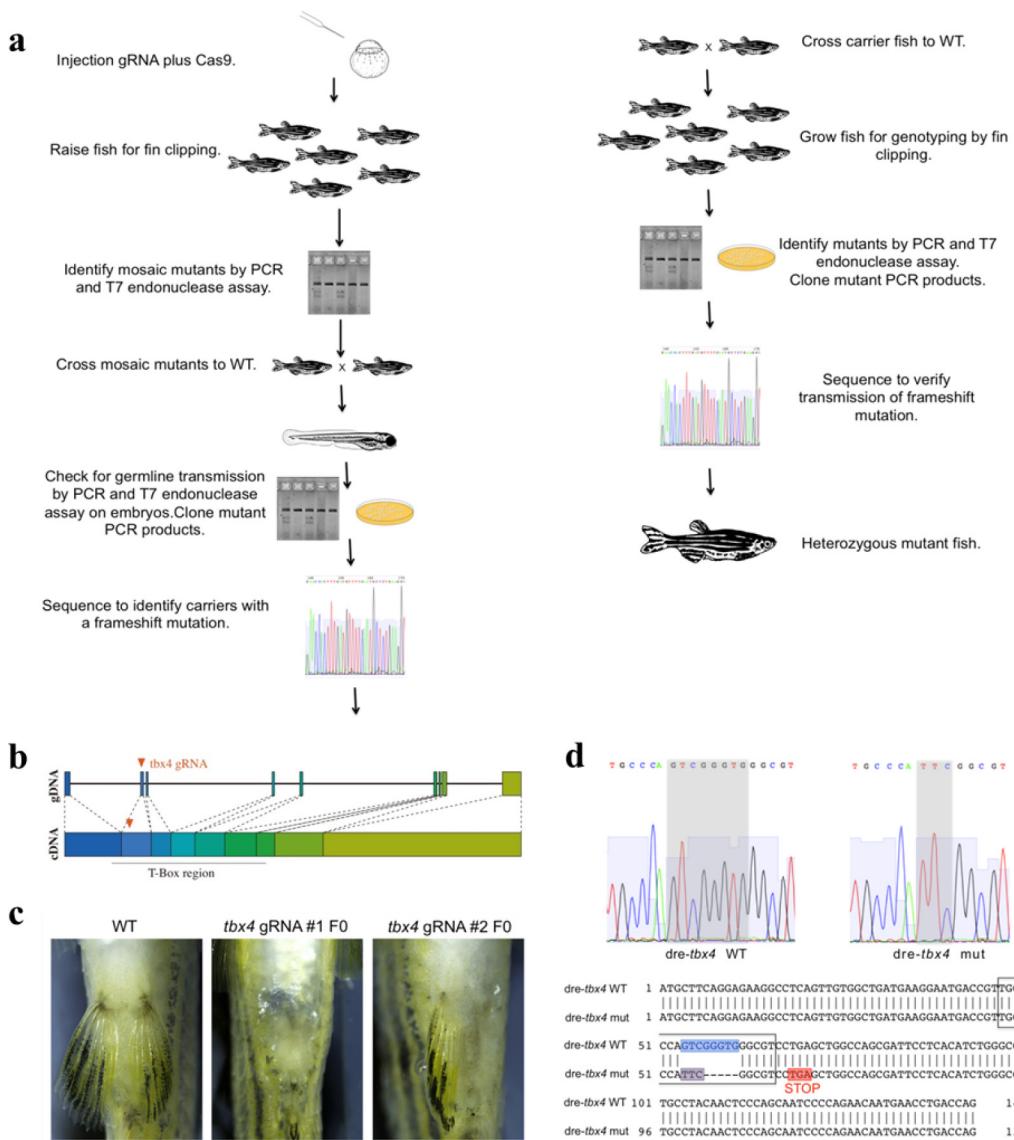


Extended Data Figure 4 | SCPP genes in *H. comes* and other jawed vertebrates. Gene loci for human, coelacanth and zebrafish were adapted from other publications^{60,61}. *sparcl1*, which is the ancestral gene that gave rise to SCPP genes is shown in grey; P/Q-rich SCPP genes are shown

in red; acidic SCPP genes are shown in blue. In seahorse, *scpp5* is a pseudogene and is denoted by ψ. Owing to space constraints, the P/Q-rich SCPP genes encoding milk casein and salivary proteins in human have been omitted. Black circles mark the ends of scaffolds.

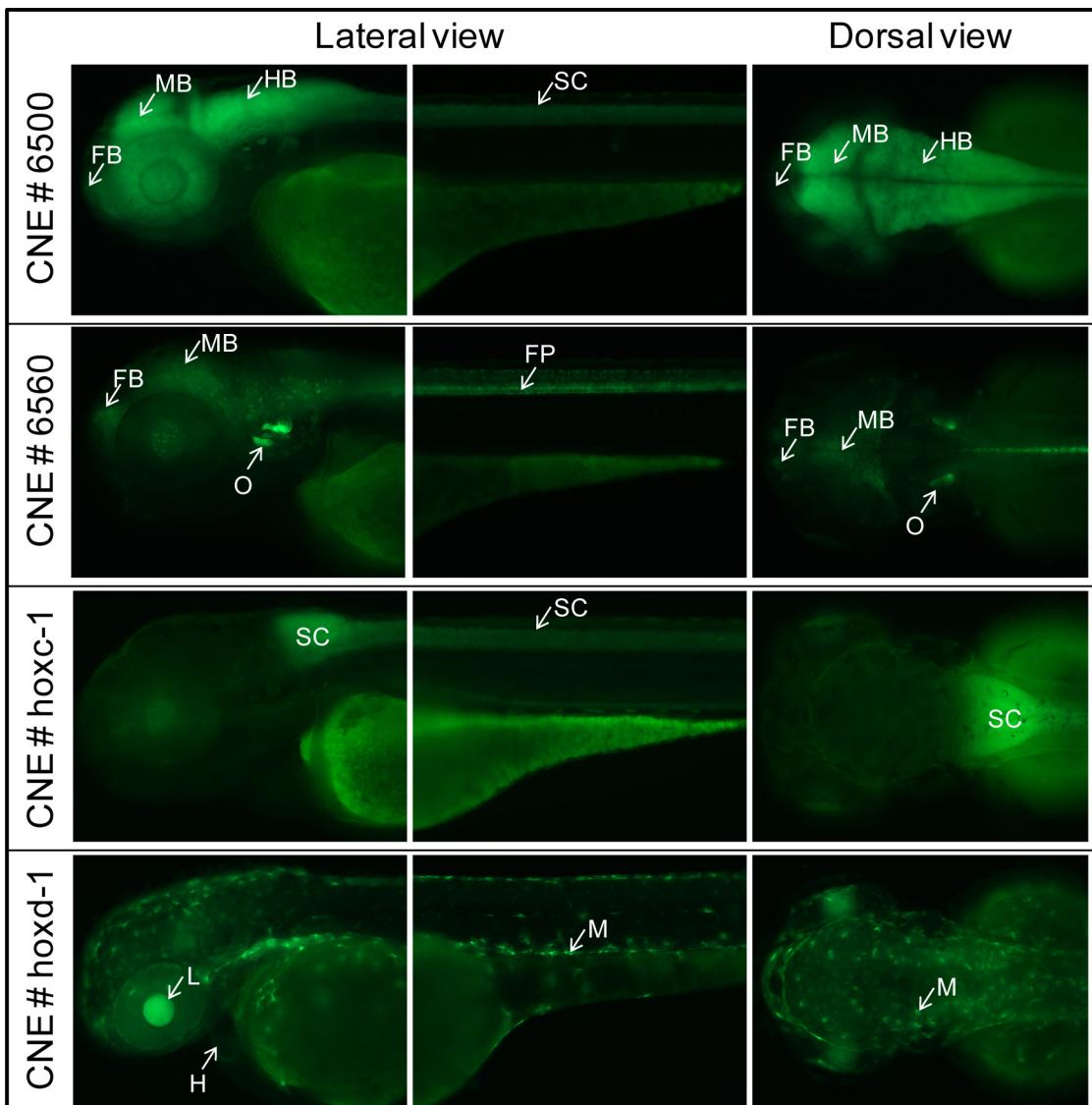


Extended Data Figure 5 | Maximum-likelihood phylogenetic tree of OR genes in *H. comes* and other ray-finned fishes.



Extended Data Figure 6 | CRISPR–Cas9 mediated knockdown of *tbx4* in zebrafish. a, CRISPR–Cas9 mutagenesis strategy. **b,** CRISPR–Cas9 sites targeted in zebrafish *tbx4* gene. **c,** Loss of function *tbx4* phenotypes in F0 mosaic mutants. Pelvic fin loss was observed with low frequency in F0 mosaic mutant fish. Frequency of animals with either single- or double-sided loss of pelvic fins was 3/42 for gRNA#1 and 1/34 for gRNA#2. **d,** Identification of zebrafish *tbx4* mutant line. Top shows

sequencing chromatograms of wild-type (left) and mutant (right) alleles. Bottom shows alignment of *tbx4* exon 2 from wild-type and mutant. The region for which the chromatograms are shown is indicated with a box. In the mutant a deletion (indicated in blue in the wild-type sequence)/substitution (indicated in lilac in the mutant sequence) was identified. The deletion/substitution area is indicated with a grey box in the chromatograms.



Extended Data Figure 7 | Reporter gene expression pattern driven by zebrafish CNEs that are lost in *H. comes*. Lateral and dorsal views of 72 h post-fertilization F1 transgenic zebrafish embryos. The lost CNEs (#6500, #6560, #hoxc-1 and #hoxd-1) were assayed for their reporter gene

expression potential in transgenic zebrafish. FB, forebrain; FP, floor plate; H, heart; HB, hindbrain; L, lens; M, melanocytes; MB, midbrain; O, otic vesicle; SC, spinal cord.

Pfam: the protein families database

Robert D. Finn^{1,*}, Alex Bateman², Jody Clements¹, Penelope Coggill^{2,3}, Ruth Y. Eberhardt^{2,3}, Sean R. Eddy¹, Andreas Heger⁴, Kirstie Hetherington³, Liisa Holm⁵, Jaina Mistry², Erik L. L. Sonnhammer⁶, John Tate^{2,3} and Marco Punta^{2,3}

¹HHMI Janelia Farm Research Campus, 19700 Helix Drive, Ashburn, VA 20147 USA, ²European Molecular Biology Laboratory, European Bioinformatics Institute (EMBL-EBI), Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SD, UK, ³Wellcome Trust Sanger Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SA, UK, ⁴MRC Functional Genomics Unit, Department of Physiology, Anatomy and Genetics, University of Oxford, Oxford, OX1 3QX, UK, ⁵Institute of Biotechnology and Department of Biological and Environmental Sciences, University of Helsinki, PO Box 56 (Viikinkaari 5), 00014 Helsinki, Finland and ⁶Stockholm Bioinformatics Center, Swedish eScience Research Center, Department of Biochemistry and Biophysics, Science for Life Laboratory, Stockholm University, PO Box 1031, SE-17121 Solna, Sweden

Received September 26, 2013; Revised November 4, 2013; Accepted November 5, 2013

ABSTRACT

Pfam, available via servers in the UK (<http://pfam.sanger.ac.uk/>) and the USA (<http://pfam.janelia.org/>), is a widely used database of protein families, containing 14 831 manually curated entries in the current release, version 27.0. Since the last update article 2 years ago, we have generated 1182 new families and maintained sequence coverage of the UniProt Knowledgebase (UniProtKB) at nearly 80%, despite a 50% increase in the size of the underlying sequence database. Since our 2012 article describing Pfam, we have also undertaken a comprehensive review of the features that are provided by Pfam over and above the basic family data. For each feature, we determined the relevance, computational burden, usage statistics and the functionality of the feature in a website context. As a consequence of this review, we have removed some features, enhanced others and developed new ones to meet the changing demands of computational biology. Here, we describe the changes to Pfam content. Notably, we now provide family alignments based on four different representative proteome sequence data sets and a new interactive DNA search interface. We also discuss the mapping between Pfam and known 3D structures.

INTRODUCTION

Pfam is a database of curated protein families, each of which is defined by two alignments and a profile hidden Markov model (HMM). Profile HMMs are probabilistic

models used for the statistical inference of homology (1,2) built from an aligned set of curator-defined family-representative sequences. A high-quality seed alignment is essential, as it provides the basis for the position-specific amino-acid frequencies, gap and length parameters in the profile HMM. In Pfam, the profile HMM is searched against a large sequence collection, based on UniProt Knowledgebase (UniProtKB) (3), to find all instances of the family. Sequence regions that score above the curated threshold that is set for each family to eliminate false positives (the so-called gathering threshold) are aligned to the profile HMM to produce the full alignment. Curated entries are referred to as Pfam-A entries. The profile HMMs are built and searched using the HHMER software suite (<http://hmmer.janelia.org>) (4,5).

Sometimes, a single profile HMM cannot detect all homologues of a diverse superfamily, so multiple entries may be built to represent different sequence families in the superfamily. Such related Pfam-A entries are grouped into clans (6). In an effort to be comprehensive, automatically generated entries, called Pfam-B, are built from sequence clusters not currently covered by Pfam-A entries.

Pfam data are available in a variety of formats, which include flatfiles (derived from the MySQL database) and relational table dumps, both of which can be downloaded from the FTP site (<ftp://ftp.sanger.ac.uk/pub/databases/Pfam>). The Pfam website (available at <http://pfam.sanger.ac.uk/> and <http://pfam.janelia.org/>) provides different ways to access the database content, providing both graphical representations of and interactive access to the data.

In the 2012 article (7), much of the content was focused on curation details. In this article, we focus on describing the new and updated data features provided in the database and by the website. Besides adding new

*To whom correspondence should be addressed. Tel: +44 571 209 4316; Fax: +44 571 209 4095; Email: finnr@janelia.hhmi.org

features, it is also important to indicate those that are no longer available, many of which have been removed due to our drive to scale with the growing influx of new sequences.

PFAM STATISTICS

The current release of Pfam, version 27.0, contains 14 831 Pfam-A families. Of these families, 4563 have been classified into 515 clans. Compared with Pfam 26.0, there has been an increase of 1159 families (1182 new entries have been added and 22 entries have been removed) and 16 new clans, with an additional 320 families having been classified into clans. The Pfam-A families in release 27.0 match 79.9% of the 23.2 million sequences and 58% of the 7.6 billion residues in the underlying sequence database. This corresponds to a negligible percentage increase in sequence and residue coverage (<0.5%), but reflects a significant amount of curation effort. These statistics mask the fact that the underlying sequence database has increased by 7.3 million sequences, a number greater than the entire sequence database of Pfam 23.0, which contained 5.3 million sequences.

Two of the main sources for generating the new families added to release 27.0 were Protein Data Bank (PDB) structures (8) and human sequences. We have made a concerted effort to build families from CATH domains (<http://www.cathdb.info/>) (9) that did not match a Pfam family in Pfam 26.0. To do so, we used jackhmmer, a program within the HMMER3 software that allows a sequence to be iteratively searched against a sequence database. One hundred new Pfam-A families were built using the sequence of a CATH domain to initiate a jackhmmer search against our underlying sequence database (three iterations were run using an E-value threshold of 0.001). Our curators then used the output from the last iteration of the jackhmmer program as the basis for generating the seed alignment of a new Pfam-A entry. We have also built families for *Homo sapiens* sequences that did not have a match in Pfam 26.0. By taking the Swiss-Prot collection of human sequences (~20 000 sequences) and excluding those sequences matched by a Pfam-A entry, each remaining sequence was used to initiate a jackhmmer search. Again, Pfam-A entries were built from the jackhmmer output. By building families in this way, we have increased the sequence coverage of the Swiss-Prot set of human sequences by almost 5% and the residue coverage by 2.2%. The Pfam 27.0 sequence coverage of Swiss-Prot human sequences is now 90.5% and the residue coverage is 45.1%. We will continue to work on incorporating more human regions into Pfam-A, as there is still much to be gained at the residue level. However, attaining high residue coverage in human is complicated by the large fraction of intrinsic disorder found in the regions that are not currently covered by Pfam-A families [discussed further in (10)]. In addition to using CATH domains and human sequences as starting points for new Pfam families, we continue to add families built from Pfam-B entries, as well as from community submissions received via our

helpdesk. We have received 135 direct submissions from our seven registered external contributors, who have our database curation tools installed locally to facilitate automated deposition.

In 2012, we described the introduction of Wikipedia as a platform for community-based functional annotation (7). Since release 26.0, the first to include links to Wikipedia articles, we have tried to link as many Pfam-A families as possible to those articles that best describe their biology. The number of families linking to a Wikipedia article increased from 4942 in 26.0 to 5663 families in release 27.0, an increase of 721. Of these 721 new links, 391 were added to old families and 330 were added to new families in Pfam 27.0. Some articles may be linked to many Pfam-A families, but the number of unique Wikipedia articles also rose by 311, from 1016 in 26.0 to 1327 in 27.0. As described previously, we operate a manual approval system that allows us to view all changes to our linked articles. Although the number of newly linked articles has increased, we have also observed a steady stream of edits to many of the linked articles. Most edits are simple format or typographic improvements, but many have also provided valuable scientific content, including significant improvements to and expansion of important articles. For example the Wikipedia article on EGF-like domains was significantly expanded in October 2012.

RECENT CHANGES TO THE DATABASE CONTENT

Removing dubious sequences from the underlying database

Each Pfam release is calculated against a fixed sequence database, called pfamseq, which is derived from UniProtKB (3). At the beginning of a release cycle, we take a copy of the current version of UniProtKB and process it in two ways, the second of which is a novel addition for release 27.0. First, we remove sequences that contain non-consecutive regions. The linear sequence-information in these proteins will be inaccurate, as adjacent residues in the sequence can flank an intervening number of unsequenced residues. There are currently <1000 UniProt entries that contain non-consecutive sequence regions. The second, new processing step is the removal of sequences derived from spurious open reading frames, which are identified by searching AntiFam (11) models against the sequence database. In release 27.0, the models from AntiFam version 2.0 identified 2829 sequences for removal.

Family full alignments and trees

When building a Pfam release, we aim to ensure that the same set of post-processing operations are performed on all families regardless of size, thereby providing consistency both to the database and to the website. One of the distinguishing features of Pfam compared with most other protein family databases is our provision of full alignments. Unsurprisingly, however, with the exponential growth of the underlying sequence database, we have observed a similar dramatic increase in the size of our full alignments. Although generation of these alignments

does not currently present a scalability problem, aiding human interpretation through visualization has become increasingly difficult. Most approaches for facilitating alignment visualization natively in the browser do not scale well. Applets, such as the Jalview alignment viewer (12), partly solve the problem, but require Java to be installed and coupled to the browser.

For example, the largest Pfam-A family (version 27.0) with >363 000 matches to the profile HMM is the ABC transporters family (ABC_tran, accession PF00005)—its full alignment is thus too large to be useful for most purposes. The seed alignment, by contrast, contains just 55 representative sequences, which may be an insufficient number to represent the sequence diversity within the family. To provide more useable samples of the sequence diversity within a family, we now calculate model-matches for four additional sequence sets, based on ‘Representative Proteomes’ (RPs) (13). For the ABC_tran family, the RP alignments range in size from approximately a quarter of the size of the full alignment to less than one tenth.

In an RP set, each member proteome is selected from a grouping of similar proteomes. The selected proteome is chosen to best represent the set of grouped proteomes in terms of both sequence and annotation information. The grouping of proteomes is based on a clustering of UniProt, UniRef50, and includes all complete proteome sequences. In each cluster, sequences have $\geq 50\%$ identity and have at least an 80% overlap with the longest sequence. The similarity of two proteomes is determined by considering just the clusters containing sequences from either of the two proteomes. The two proteomes are grouped when the fraction of clusters that contain sequences from both proteomes out of the subset of proteome-specific clusters exceeds a given threshold. This threshold is termed the co-membership threshold. The percentage threshold of co-membership (or common clusters) can be adjusted down to produce larger groupings, and hence less redundant sequence sets.

We use the RP sequence sets constructed using co-membership thresholds of 75, 55, 35 and 15%, giving a range of sequence redundancy for each family. Using representative proteomes has the advantage that it still allows

for organism-specific copy numbers to be assessed, a feature that can be lost when using global non-redundancy thresholds on an entire sequence database. However, the major advantage for Pfam is the dramatic reduction in the size of the family full alignments, as shown in Table 1, which illustrates the reductions with increasingly redundant RPs for the 10 biggest families in Pfam. The RP sets do not currently include viruses, and so for some families such as GP120, there may not be a match to the RP sets.

The reduction in the size of the full alignments varies from family to family, reflecting in part the bias in the sequence database. Overall, across the whole of the database, using RP at 75, 55, 35 and 15% co-membership thresholds results in average alignment sizes that are, respectively, 38.8, 29.7, 20.4 and 11.6% of the full alignment size. As the number of sequences in the sequence database increases, we anticipate that the alignments based on RPs will grow at a more linear rate and provide a more convenient way of sampling the full alignment sequence diversity.

As illustrated in Table 1, the full alignment size for the top 10 families ranges from 129 000 to 363 000 sequences. With alignments of this size, it is no longer practical to calculate the neighbour-joining trees provided in previous Pfam releases. Before release 27.0, these approximate neighbour-joining phylogenetic trees (with bootstrapping values based on 100 replicas) were used to order the alignments, such that phylogenetically related sequences would be grouped together. From release 27.0 onwards, the full alignments are ordered according to the HMMER bit score of the match, with the highest scoring sequence found at the top of the alignment. The same phylogenetic trees are still provided for the seed alignments, but are merely a guide as they are calculated with the FastTree approximation algorithm (14). The seed alignment sequences remain ordered according to the calculated tree.

In the Pfam website, we use two different colouring schemes when displaying our alignments in a web browser: the Clustal scheme (15), based on the chemical properties of the amino acids found in the column, and a heat-map scheme that reflects the posterior

Table 1. The reduction in size of RP versus full alignments

Family identifier (accession)	Seed	Full	RP75	RP55	RP35	RP15
ABC_tran (PF00005)	55	363 409	26% (93 265)	21% (77 150)	16% (57 358)	8% (28 903)
COX1 (PF00115)	94	254 351	1% (2006)	0.7% (1661)	0.4% (1218)	0.2% (538)
zf-H2C2_2 (PF13465)	163	227 898	61% (138 033)	27% (60 664)	15% (34 039)	9% (21 562)
WD40 (PF00400)	1804	193 252	65% (125 805)	52% (100 531)	36% (69 386)	23% (21 562)
MFS_1 (PF07690)	195	181 668	30% (55 719)	25% (55 719)	17% (55 719)	8% (55 719)
RVT_1 (PF00078)	152	172 360	5% (8257)	4% (6662)	3% (5373)	2% (3604)
BPD_transp_1 (PF00528)	81	156 339	23% (36 523)	19% (29 422)	14% (22 134)	7% (10 630)
Response_reg (PF00072)	57	151 337	29% (44 329)	25% (37 848)	20% (29 453)	10% (15 208)
GP120 (PF00516)	24	146 453	N/A	N/A	N/A	N/A
HATPase_c (PF02518)	659	129 386	28% (36 085)	24% (30 935)	19% (24 121)	10% (12 473)

The seed alignment is used to construct the profile HMM and contains a representative set of sequences of the family. The full alignment contains all hits in pfamsq scoring above the gathering threshold. In Pfam 27.0, we have introduced four additional alignments based on RPs, which contain decreasing amounts of sequence redundancy from RP75 to RP15. For each RP data set, the percentage reduction in the size of the full alignment is shown, with the number of sequences given in brackets.

	Seed (94)	Full (254351)	Representative proteomes				NCBI (206187)	Meta (5121)
			RP15 (538)	RP35 (1218)	RP55 (1661)	RP75 (2006)		
Jalview	✓	✓	✓	✓	✓	✓	✓	✓
HTML	✓	—	✓	✓	✓	✓	✗	✗
PP/heatmap	✗ ₁	—	✓	✓	✓	✓	✗	✗
Pfam viewer	✓	✓	✗	✗	✗	✗	✗	✗

Figure 1. Table from the ‘Alignments’ tab of the family page for COX1 (PF00115), showing the availability of different views and different alignments for COX1. The posterior probability-based alignment is only available for the full alignments as it is derived from the alignment of a sequence to the HMM, as indicated by the subscript 1 in the corresponding seed alignment cell.

probability of alignment confidence (16). However, the complexity of the large multiple sequence alignments, in terms of gaps and variation, can result in vast numbers of HTML elements being generated to mark up an entire alignment. The maximum number of elements that can be displayed depends on the user’s browser and hardware, but, in an effort to protect users from attempting to view alignments that are unlikely ever to be rendered, we only make HTML versions of alignments that contain 5000 sequences or fewer. In an effort to convey which options for viewing an alignment are available for a given family via the website, we present a table indicating the availability of the alignment view option (Figure 1).

SEARCH-INTERFACE DEVELOPMENTS

As the volume of data in Pfam increases, it is important to make that data even more discoverable. Before Pfam 27.0, keyword searches were performed via the backend MySQL database, using the ‘fulltext’ indexing method offered by the database engine. However, the performance of this search was deteriorating as the database grew with each release, particularly when queried with common words. To ensure future scalability, keyword searches are performed outside of the database, using Apache Lucy (<http://lucy.apache.org>), a tool specifically designed for full-text indexing. This has allowed us to tailor the searches to improve specificity (any query term of ≥ 2 characters will be used as a query), such that all query-matching strings, including substrings, are found for text associated with a Pfam-A family, structures and ontology; the sequence-annotations are also indexed, but, due to the quantity of text, this index is built only to match complete words. Results from the different text indexes are amalgamated and ordered, based on the index—prioritized in the following order: Pfam, sequence annotation, structure, Gene Ontology and InterPro—and the query term score. Keyword searches are now interactive, typically returning in <100 ms.

Faster interactive DNA searches

Pfam has provided an asynchronous DNA search tool since 2000 (17). The function of this tool is to try to identify the presence of Pfam-A families on an input DNA sequence, with results emailed to the user. Currently, it is not possible to compare directly a protein profile HMM against a DNA sequence using

HMMER3. The previously described search was constructed around the GeneWise software (18), and would compare the DNA sequence to the protein profile HMMs via a gene model. The GeneWise software was originally written for profile HMMs built using the HMMER2 software suite, and although it is possible to back-convert HMMER3 models to HMMER2 format, we found that there was a significant loss in sensitivity for these searches. HMMER3 models tend to have lower relative entropy per position due to the altered prior weighting, compared with HMMER2. This, coupled with the tuning of GeneWise specificity, could account for the loss of sensitivity. However, the increased speed of HMMER3 presented an alternative approach for the detection of Pfam matches on DNA sequences. As opposed to the more sophisticated gene structure-aware approach used previously, we now can perform a standard six-frame translation on the DNA, and search each of the resulting ‘protein’ sequences against the Pfam-A library. This brute-force approach with HMMER3 is sufficiently quick to allow the use of the same interface as we use for the interactive protein sequence searches, thus unifying the sequence search interface for both protein and DNA. In the DNA search results page (Figure 2), each open reading frame is represented graphically, with the positions of the stop codons in the reading frame highlighted by red square lollipops and the positions of any domains represented using the standard Pfam domain representations. The DNA search functionality has also been incorporated into pfam_scan.pl, our downloadable tool for performing sequence searches against Pfam.

Changes associated with alternative target sequence databases

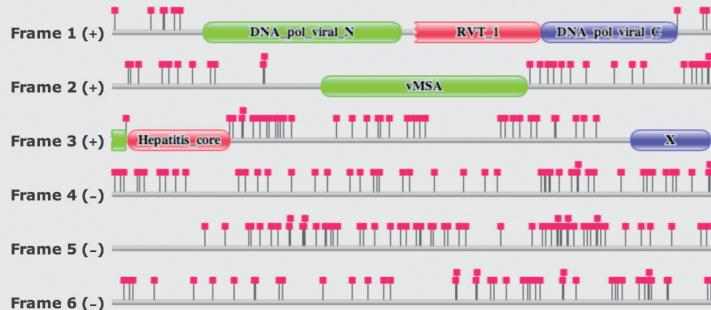
To streamline the production of the database, we no longer store the matches to the NCBI NR (non-redundant) protein sequence database (19) or our metagenomics sequence collection. We still provide Stockholm formatted alignments of all matches to each family found in these two sequence databases as well as retrieval of sequences via accession (e.g. 22125853 or EBH56784.1). However, the data for non-UniProt sequence pages come from an on-the-fly search of the sequence against the Pfam-A HMM library. Generating the data in this manner not only reduces the time required to populate the database, but also provides a more coherent view of the Pfam match data: overlapping matches arising from other clan families can be removed (previously all matches were reported for

DNA sequence search results

This page shows the results of searching your DNA sequence for Pfam-A matches. To do this we perform a six-frame translation to generate a set of protein sequences, which we then search using the normal Pfam-A HMMs and **GA** cut-offs.

[Show](#) the detailed description of this results page.

We have found **7** significant hits and **0** insignificant hits in **3** frames.



[Show](#) the DNA and protein sequences, and the URL for bookmarking these results.

[Return](#) to the search form to look for Pfam domains on a new sequence.

Significant Pfam-A Matches

Show or hide all alignments. [Toggle](#) between amino-acid and DNA sequence coordinates.

Frame (sense)	Family	Description	Entry type	Clan	Envelope		Alignment		HMM		HMM length	Bit score	E-value	Predicted active sites	Show/hide alignment
					Start	End	Start	End	From	To					
1 (+)	DNA_pol_viral_N	DNA polymerase (viral) N-terminal domain	Family	n/a	163	514	163	514	1	379	379	573.7	1.7e-172	n/a	Show
1 (+)	RVT_1	Reverse transcriptase (RNA-dependent DNA ...	Family	CL002Z	538	762	539	762	2	214	214	206.6	2.7e-61	n/a	Show
1 (+)	DNA_pol_viral_C	DNA polymerase (viral) C-terminal domain	Family	n/a	763	1005	763	1005	1	245	245	477.2	8.1e-144	n/a	Show
2 (-)	vMSA	Major surface antigen from hepadnavirus	Family	n/a	372	737	372	737	1	364	364	563.4	2.2e-169	n/a	Show
3 (+)	Hep_core_N	Hepatitis core protein, putative zinc fi ...	Domain	n/a	1	24	1	24	4	27	27	58.7	2.5e-16	n/a	Show
3 (+)	Hepatitis_core	Hepatitis core antigen	Domain	n/a	28	209	28	209	1	187	187	315.3	1.1e-94	n/a	Show
3 (-)	X	Trans-activation protein X	Family	n/a	923	1064	923	1064	1	142	142	268.1	1.4e-80	n/a	Show

Comments or questions on the site? Send a mail to pfam-help@sanger.ac.uk. Our cookie policy.

The Wellcome Trust

Figure 2. Results from searching Pfam with the Hepatitis B virus isolate G376-7, complete genome (GenBank accession AF384371.1), providing a striking example of overlapping genes. The six reading frames are displayed graphically in the top box of the results page. All three reading frames from the positive strand contain matches to Pfam-A, which are tabulated below. The positions of stop codons are indicated by the square lollipops. The results are shown with the ‘protein’ coordinates of the open reading frame, but it is also possible to toggle this to DNA sequence coordinates. This search tool accepts sequences up to 80 000 nucleotides in length, and searches the Pfam-A HMM library using the gathering threshold.

the NR and metagenomics sets) using the same rules that are used for UniProtKB sequences. As a result, the view is identical to the UniProt sequence page, where the data are retrieved from the database.

DEPRECATED FEATURES

In our 2004 article (20), we described the introduction of contextual domain-hits, which used language-modeling techniques to identify weak domain hits that fell just below the gathering threshold but had support from surrounding domains (or contextual information) (21). Unfortunately, the third-party software used to generate such matches in Pfam is no longer supported and the existing implementation fails to scale, both in terms of time and memory, when presented with the tens of millions of matches now reported by Pfam. Although there is merit in providing additional functional annotations via contextual domain-hits, the improved sensitivity offered by HMMER3, the introduction of clans (which

allows us to build multiple models for ubiquitous domains that cannot readily be matched by a single model) and/or simply improved models, means that many of these contextual domains are now reported by standard Pfam-A matches (Table 2). Since the last time it was calculated, in 2007, 37% of the previously identified contextual hits (10 559) are now covered by Pfam entries. The majority of contextual hits were for Pfam-A entries of type ‘Repeat’ and the highest proportion of unidentified hits belong to this entry type. This reflects the difficulty we have in generating profile HMMs that are able to detect all instances of a short degenerate, repeating sequence motif. Table 2 summarizes the breakdown of context hits that are now matched in Pfam 27.0.

In addition to removing features based on scalability issues, we also routinely analyze the web server access logs, to assess how the site is used. From such analyses, we have identified that the functional similarity search, which used a similarity tool (22) to identify sets of related Pfam-A families based on functional annotation

Table 2. Breakdown of contextual hits that are reported by Pfam entries in Pfam 27.0, according to the protein family type

Entry type	% Context regions reported in Pfam 27.0	% Context regions not reported in Pfam 27.0
Family	4	7
Domain	13	13
Motif	<1	2
Repeat	20	41
All	37	63

The percentage reported for each entry type is the fraction out of all of the 10 559 contextual domains, with the total for all domains shown at the bottom of the table.

(Gene Ontology terms), was not being used. We have removed this search facility from the site.

IMPROVING ACCESS TO PROTEOME DATA

Before release 27.0, Pfam proteome data came from Integr8, a project that has now closed and whose data have been distributed to other EBI resources. We now obtain our complete proteome data directly from UniProt, at the beginning of the release cycle when the sequence database is retrieved. This has resulted in better consistency between the sequence sets, with 40% (9 423 167 sequences) of the 23 193 494 sequences in pfamseq belonging to a complete proteome. Over the past few years, we have received an increasing demand for proteome-centric Pfam data. The data-interface to the proteome data is an area of future development but, to satisfy one of our most common user queries, we now provide a list of all Pfam-A matches per proteome on our FTP site (ftp://ftp.sanger.ac.uk/pub/databases/Pfam/current_release/proteomes). Each list can also be accessed from the corresponding proteome's 'domain composition' tab on the proteome-pages in the website.

REPRESENTING INTRINSIC SEQUENCE DISORDER

Pfam often quotes 'sequence coverage' and 'residue coverage' as statistics for tracking the extent of annotation provided by the database. We have previously noted that achieving 100% residue coverage is an unrealistic goal, as every residue in a sequence does not form part of a conserved globular domain (23), such as signal peptides and domain linker regions (short regions are essential for interdomain interactions, folding and stability) (24–27). To aid in the identification of non-globular domain regions, we have displayed the predictions of signal peptides (28), low complexity (29) and coiled-coils (<http://www.russelllab.org/cgi-bin/coils/coils-svr.pl>) for many years. As part of recent, focused curation efforts aimed at increasing the Pfam-A coverage of the human proteome (10), it became apparent that many regions not covered by Pfam-A are predicted to be intrinsically disordered. Disorder is not an indicator of a lack of function; on the contrary, it has been shown to be involved in cell signaling, protein interactions and

regulation (30–33). Some disordered regions are conserved and are found within existing domains, e.g. in PF03250 (Tropomodulin), but they generally appear to be less conserved and/or shorter than globular domains (10), making them more elusive to modeling in a conventional Pfam-A entry. Therefore, to provide a means of identifying more disordered regions in Pfam, we have incorporated IUPred predictions (34,35) (using the long disorder prediction option) for all pfamseq sequences. These data are stored in the MySQL database, and displayed graphically as grey boxes on the website graphical representation of a sequence, as in Figure 3. The IUPred disorder predictions supplement those already produced by SEG (29), which predict a single class of disorder. Although more common to eukaryotes, disordered regions are widespread in UniProtKB. In Pfam 27.0, there were 5.5 million IUPred disorder regions of 50 amino acids or more in length, corresponding to 5.6% of the 7.6 billion sequence residues in the database.

MAPPING PFAM-A ENTRIES TO PROTEIN STRUCTURES

A recurring issue, and one which is often raised in the literature (36) and by Pfam users, is the mapping of Pfam-A entries to PDB entries, a process that can provide 3D structural information for a protein family. This may seem like a trivial task, whereby one simply extracts all of the protein chains in all of the PDB entries and searches them against Pfam-A. However, although this approach works in principle, in practice it results in many omissions from the mapping. PDB entries frequently include only part of a sequence and the visible fragments are often simply too short to have matches to Pfam profile HMMs that are significant. For example, the crystal structure of the murine class I major histocompatibility antigen H-2D(B) has been determined in complex with a nine amino acid peptide derived from the LCMV gp33 protein (PDB identifier 1S7W) (37). Searching just the gp33 fragment against the Pfam-A models finds no hits. However, by using the residue mapping between PDB structures and UniProtKB entries provided by the SIFTS resource (38), we find that the fragment comes from a larger sequence, UniProtKB accession P07399, in a region that matches the Arena_glycoprot family (Pfam accession PF00798). This demonstrates the importance of using a comprehensive and accurate structure-to-sequence mapping, such as SIFTS, to unify structural and sequence information.

The caveat to the approach described earlier in the text is that structure, mapping and sequence data, from PDB, SIFTS and Pfam, respectively, must be time-synchronized. All resource providers are aware of the issues generated by multiple release cycles and our pipeline has been modified to ensure that, at the point of data acquisition, PDB, SIFTS and UniProt are as tightly synchronized as possible. However, as there is a steady flow of structures into the PDB every week and, since our data are often downloaded and frozen months before a release, it will almost always appear out of date. During the lifetime of

Protein: ABL1_HUMAN (P00519)

[Summary](#)
[Features](#)
[Sequence](#)
[Interactions](#)
[Structures](#)
[TreeFam](#)

[Jump to...](#)
 [Go](#)

1 architecture
1 sequence
0 interactions
1 species
36 structures

Summary

This is the summary of UniProt entry [ABL1_HUMAN \(P00519\)](#).

Description:	Tyrosine-protein kinase ABL1 EC=2.7.10.2
Source organism:	Homo sapiens (Human) (NCBI taxonomy ID 9606) View Pfam proteome data.
Length:	1130 amino acids

Please note: when we start each new Pfam data release, we take a copy of the UniProt sequence database. This snapshot of UniProt forms the basis of the overview that you see here. It is important to note that, although some UniProt entries may be removed after a Pfam release, these entries will not be removed from Pfam until the next Pfam data release.

Pfam domains

This image shows the arrangement of the Pfam domains that we found on this sequence. Clicking on a domain will take you to the page describing that Pfam entry. The table below gives the domain boundaries for each of the domains. [More...](#)

Source	Domain	Start	End
low_complexity	n/a	5	23
disorder	n/a	53	54
Pfam A	SH3_1	67	113
Pfam A	SH2	127	202
Pfam A	Pkinase_Tyr	242	493
low_complexity	n/a	249	260
disorder	n/a	489	490
disorder	n/a	503	511
disorder	n/a	513	719
low_complexity	n/a	605	616
low_complexity	n/a	629	639
low_complexity	n/a	701	710
disorder	n/a	724	993
Pfam B	Pfam-B_13565	864	1003
low_complexity	n/a	896	915
low_complexity	n/a	977	993
disorder	n/a	1009	1010
disorder	n/a	1013	1021
Pfam A	F_actin_bind	1022	1130
disorder	n/a	1023	1027

[Show](#) or [hide](#) domain scores.

Comments or questions on the site? Send a mail to pfa-help@sanger.ac.uk. Our [cookie policy](#).

The Wellcome Trust

Figure 3. Graphical representation of the Pfam sequence annotations for human tyrosine-protein kinase ABL1 sequence (UniProtKB accession P00519). This sequence matches four different Pfam-A entries, SH3_1 (PF00018), SH2 (PF00017), Pkinase_Tyr (PF007714) and F_actin-bind (PF08919). Between the Pkinase_Tyr and F_actin_bind families is a long region of disorder, indicated by the presence of the grey boxes on the sequence. A disorder prediction does not necessarily mean that the sequence is not conserved, highlighted by the presence of an overlapping Pfam-B region (striped box).

a Pfam release, the disparity will become increasingly wide. One solution would be to pull this data in dynamically during a Pfam release, but we are opposed to this approach because we believe that the data in a given Pfam release should be fixed, to provide a stable data source for the community to cite. Should obtaining the latest Pfam-PDB annotation-mapping be paramount, both PDBe (39) and RCSB (40) offer tab-delimited files with the latest mappings (ftp://ftp.ebi.ac.uk/pub/databases/msd/sifts/flatfiles/csv/pdb_chain_pfam.csv.gz or http://www.rcsb.org/pdb/rest/hmmer?file=hmmer_pdb_all.txt). A better

solution might be to make more frequent Pfam releases, thereby minimizing the data synchronization lags. Continued improvements in our release pipeline are designed to facilitate shorter release cycles in the future.

CONCLUSIONS

The core aim of Pfam is to produce protein families that reliably classify as much of sequence space as possible. The database continues to grow and evolve during 2013, with efforts concentrated on adding new families and

Downloaded from <http://nar.oxfordjournals.org/> at Iowa University on September 5, 2016

improving existing ones, while also trying to make the core family data as accessible as possible. The growing sequence database is competing with this effort. We continue to focus attention on meeting the needs of our users, which are often highlighted by recurring user requests. Part of this effort is to identify and remove features that have not been useful to users. It is always tempting to add progressively more features to the database, but this would make it impossible to keep Pfam maintainable in the long term. However, we still encourage the Pfam user community to ask for data sets that are either not provided or not easily accessible. We are committed to producing more frequent releases, a process which may result in further changes to the database and website.

FUNDING

Howard Hughes Medical Institute Janelia Farm Research Campus (to R.D.F., J.C. and S.R.E); the European Molecular Biology Laboratory, European Bioinformatics Institute (EMBL-EBI) (to A.B. and J.M.); Wellcome Trust [WT077044/Z/05/Z to P.C., R.Y.E., K.H., J.T. and M.P.]. Funding for open access charge: HHMI Janelia Farm Research Campus.

Conflict of interest statement. None declared.

REFERENCES

- Krogh,A., Brown,M., Mian,I.S., Sjölander,K. and Haussler,D. (1994) Hidden Markov models in computational biology. Applications to protein modeling. *J. Mol. Biol.*, **235**, 1501–1531.
- Eddy,S.R. (1998) Profile hidden Markov models. *Bioinformatics*, **14**, 755–763.
- UniProt Consortium. (2012) Reorganizing the protein space at the Universal Protein Resource (UniProt). *Nucleic Acids Res.*, **40**, D71–D75.
- Eddy,S.R. (2009) A new generation of homology search tools based on probabilistic inference. *Genome Inform.*, **23**, 205–211.
- Eddy,S.R. (2011) Accelerated profile HMM searches. *PLoS Comput. Biol.*, **7**, e1002195.
- Finn,R.D., Mistry,J., Schuster-Böckler,B., Griffiths-Jones,S., Hollich,V., Lassmann,T., Moxon,S., Marshall,M., Khanna,A., Durbin,R. *et al.* (2006) Pfam: clans, web tools and services. *Nucleic Acids Res.*, **34**, D247–D251.
- Punta,M., Coggill,P.C., Eberhardt,R.Y., Mistry,J., Tate,J., Boursnell,C., Pang,N., Forslund,K., Ceric,G., Clements,J. *et al.* (2012) The Pfam protein families database. *Nucleic Acids Res.*, **40**, D290–D301.
- Bernstein,F.C., Koetzle,T.F., Williams,G.J., Meyer,E.F., Brice,M.D., Rodgers,J.R., Kennard,O., Shimanouchi,T. and Tasumi,M. (1977) The Protein Data Bank: a computer-based archival file for macromolecular structures. *J. Mol. Biol.*, **112**, 535–542.
- Sillitoe,I., Cuff,A.L., Dessimoz,B.H., Dawson,N.L., Furnham,N., Lee,D., Lees,J.G., Lewis,T.E., Studer,R.A., Rentzsch,R. *et al.* (2013) New functional families (FunFams) in CATH to improve the mapping of conserved functional sites to 3D structures. *Nucleic Acids Res.*, **41**, D490–D498.
- Mistry,J., Coggill,P., Eberhardt,R.Y., Deiana,A., Giansanti,A., Finn,R.D., Bateman,A. and Punta,M. (2013) The challenge of increasing Pfam coverage of the human proteome. *Database*, **2013**, bat023.
- Eberhardt,R.Y., Haft,D.H., Punta,M., Martin,M., O'Donovan,C. and Bateman,A. (2012) AntiFam: a tool to help identify spurious ORFs in protein annotation. *Database*, **2012**, bas003.
- Waterhouse,A.M., Procter,J.B., Martin,D.M.A., Clamp,M. and Barton,G.J. (2009) Jalview Version 2—a multiple sequence alignment editor and analysis workbench. *Bioinformatics*, **25**, 1189–1191.
- Chen,C., Natale,D.A., Finn,R.D., Huang,H., Zhang,J., Wu,C.H. and Mazumder,R. (2011) Representative proteomes: a stable, scalable and unbiased proteome set for sequence analysis and functional annotation. *PLoS One*, **6**, e18910.
- Price,M.N., Dehal,P.S. and Arkin,A.P. (2010) FastTree 2—approximately maximum-likelihood trees for large alignments. *PLoS One*, **5**, e9490.
- Thompson,J.D., Gibson,T.J. and Higgins,D.G. (2002) Multiple sequence alignment using ClustalW and ClustalX. *Curr. Protoc. Bioinformatics*, **Chapter 2**, Unit 2.3.
- Finn,R.D., Mistry,J., Tate,J., Coggill,P., Heger,A., Pollington,J.E., Gavin,O.L., Gunasekaran,P., Ceric,G., Forslund,K. *et al.* (2010) The Pfam protein families database. *Nucleic Acids Res.*, **38**, D211–D222.
- Bateman,A., Birney,E., Durbin,R., Eddy,S.R., Howe,K.L. and Sonnhammer,E.L. (2000) The Pfam protein families database. *Nucleic Acids Res.*, **28**, 263–266.
- Birney,E., Clamp,M. and Durbin,R. (2004) GeneWise and genewise. *Genome Res.*, **14**, 988–995.
- NCBI Resource Coordinators. (2013) Database resources of the National Center for Biotechnology Information. *Nucleic Acids Res.*, **41**, D8–D20.
- Bateman,A., Coin,L., Durbin,R., Finn,R.D., Hollich,V., Griffiths-Jones,S., Khanna,A., Marshall,M., Moxon,S., Sonnhammer,E.L.L. *et al.* (2004) The Pfam protein families database. *Nucleic Acids Res.*, **32**, D138–D141.
- Coin,L., Bateman,A. and Durbin,R. (2004) Enhanced protein domain discovery using taxonomy. *BMC Bioinformatics*, **5**, 56.
- Schlicker,A., Huthmacher,C., Ramírez,F., Lengauer,T. and Albrecht,M. (2007) Functional evaluation of domain-domain interactions and human protein interaction networks. *Bioinformatics*, **23**, 859–865.
- Sammut,S.J., Finn,R.D. and Bateman,A. (2008) Pfam 10 years on: 10,000 families and still growing. *Brief. Bioinformatics*, **9**, 210–219.
- Gokhale,R.S. and Khosla,C. (2000) Role of linkers in communication between protein modules. *Curr. Opin. Chem. Biol.*, **4**, 22–27.
- George,R.A. and Heringa,J. (2002) An analysis of protein domain linkers: their classification and role in protein folding. *Protein Eng.*, **15**, 871–879.
- Wriggers,W., Chakravarty,S. and Jennings,P.A. (2005) Control of protein functional dynamics by peptide linkers. *Biopolymers*, **80**, 736–746.
- Reddy Chichili,V.P., Kumar,V. and Sivaraman,J. (2013) Linkers in the structural biology of protein-protein interactions. *Protein Sci.*, **22**, 153–167.
- Käll,L., Krogh,A. and Sonnhammer,E.L.L. (2004) A combined transmembrane topology and signal peptide prediction method. *J. Mol. Biol.*, **338**, 1027–1036.
- Wootton,J.C. (1994) Non-globular domains in protein sequences: automated segmentation using complexity measures. *Comput. Chem.*, **18**, 269–285.
- Midic,U., Oldfield,C.J., Dunker,A.K., Obradovic,Z. and Uversky,V.N. (2009) Unfoldomics of human genetic diseases: illustrative examples of ordered and intrinsically disordered members of the human disasome. *Protein Pept. Lett.*, **16**, 1533–1547.
- Babu,M.M., van der Lee,R., de Groot,N.S. and Gsponer,J. (2011) Intrinsically disordered proteins: regulation and disease. *Curr. Opin. Struct. Biol.*, **21**, 432–440.
- Tantos,A., Han,K.H. and Tompa,P. (2012) Intrinsic disorder in cell signaling and gene transcription. *Mol. Cell. Endocrinol.*, **348**, 457–465.
- Buljan,M., Chalancon,G., Dunker,A.K., Bateman,A., Balaji,S., Fuxreiter,M. and Babu,M.M. (2013) Alternative splicing of intrinsically disordered regions and rewiring of protein interactions. *Curr. Opin. Struct. Biol.*, **23**, 443–450.

34. Dosztányi,Z., Csizmok,V., Tompa,P. and Simon,I. (2005) The pairwise energy content estimated from amino acid composition discriminates between folded and intrinsically unstructured proteins. *J. Mol. Biol.*, **347**, 827–839.
35. Dosztányi,Z., Csizmok,V., Tompa,P. and Simon,I. (2005) IUPred: web server for the prediction of intrinsically unstructured regions of proteins based on estimated energy content. *Bioinformatics*, **21**, 3433–3434.
36. Xu,Q. and Dunbrack,R.L. (2012) Assignment of protein sequences to existing domain and family classification systems: Pfam and the PDB. *Bioinformatics*, **28**, 2763–2772.
37. Velloso,L.M., Michaëlson,J., Ljunggren,H.G., Schneider,G. and Achour,A. (2004) Determination of structural principles underlying three different modes of lymphocytic choriomeningitis virus escape from CTL recognition. *J. Immunol.*, **172**, 5504–5511.
38. Velankar,S., Dana,J.M., Jacobsen,J., van Ginkel,G., Gane,P.J., Luo,J., Oldfield,T.J., O'Donovan,C., Martin,M.J. and Kleywegt,G.J. (2013) SIFTS: structure integration with function, taxonomy and sequences resource. *Nucleic Acids Res.*, **41**, D483–D489.
39. Velankar,S., Alhroub,Y., Best,C., Caboche,S., Conroy,M.J., Dana,J.M., Fernandez Montecelo,M.A., van Ginkel,G., Golovin,A., Gore,S.P. et al. (2012) PDBe: Protein Data Bank in Europe. *Nucleic Acids Res.*, **40**, D445–D452.
40. Rose,P.W., Bi,C., Bluhm,W.F., Christie,C.H., Dimitropoulos,D., Dutta,S., Green,R.K., Goodsell,D.S., Prlić,A., Quesada,M. et al. (2013) The RCSB Protein Data Bank: new resources for research and education. *Nucleic Acids Res.*, **41**, D475–D482.

WEEK 5 READINGS

Protein structure homology modeling using SWISS-MODEL workspace

Lorenza Bordoli, Florian Kiefer, Konstantin Arnold, Pascal Benkert, James Battey & Torsten Schwede

Biozentrum, University of Basel and Swiss Institute of Bioinformatics, Klingelbergstrasse 50/70, CH 4056 Basel, Switzerland. Correspondence should be addressed to T.S. (torsten.schwede@unibas.ch).

Published online 11 December 2008; corrected online 18 June 2009 (details online); doi:10.1038/nprot.2008.197

Homology modeling aims to build three-dimensional protein structure models using experimentally determined structures of related family members as templates. SWISS-MODEL workspace is an integrated Web-based modeling expert system. For a given target protein, a library of experimental protein structures is searched to identify suitable templates. On the basis of a sequence alignment between the target protein and the template structure, a three-dimensional model for the target protein is generated. Model quality assessment tools are used to estimate the reliability of the resulting models. Homology modeling is currently the most accurate computational method to generate reliable structural models and is routinely used in many biological applications. Typically, the computational effort for a modeling project is less than 2 h. However, this does not include the time required for visualization and interpretation of the model, which may vary depending on personal experience working with protein structures.

INTRODUCTION

The three-dimensional structure of a protein provides important information for understanding its biochemical function and interaction properties in molecular detail. However, the number of known protein sequences is much larger than the number of experimentally solved protein structures. As of August 2008, more than 52,500 experimentally determined protein structures were deposited in the Protein Data Bank (PDB)¹. Yet, this number appears relatively small compared with the more than 6 million protein sequences held in the UniProt knowledge database². Fortunately, the number of different protein fold families occurring in nature appears to be limited³, and within a protein family, structural similarity between two homologous proteins can be inferred from sequence similarity⁴. Homology modeling (or comparative protein structure modeling) techniques have been developed to build three-dimensional models of a protein (target) from its amino-acid sequence on the basis of an alignment with a similar protein with known structure (template)^{5–7}. In cases where no suitable template structure can be identified, *de novo* (a.k.a. *ab initio*) structure prediction methods can be used to generate three-dimensional protein models without relying on a homologous template structure. However, despite recent progress in the field, *de novo* predictions are limited to relatively small proteins and fall short in terms of accuracy compared with comparative models^{8–12}. Therefore, homology modeling is the method of choice to build reliable three-dimensional *in silico* models of a protein in all cases where template structures can be identified.

Homology models are widely used in many applications, such as virtual screening, designing site-directed mutagenesis experiments or in rationalizing the effects of sequence variations^{13–17}. Stable, reliable and accurate systems for automated homology modeling are therefore required, which are easy to use for both nonspecialists and experts in structural bioinformatics.

Homology modeling

Homology modeling in general consists of four main steps: (i) identifying evolutionarily related proteins with experimentally solved structures that can be used as template(s) for modeling

the target protein of interest; (ii) mapping corresponding residues of target sequence and template structure(s) by means of sequence alignment methods and manual adjustment; (iii) building the three-dimensional model on the basis of the alignment; and (iv) evaluating the quality of the resulting model^{14,15}. This procedure can be iterated until a satisfactory model is obtained (Fig. 1).

Protein structure homology modeling relies on the evolutionary relationship between the target and template proteins. Potential structural templates are identified using a search for homologous proteins in a library of experimentally determined protein structures. From the resulting list of possible candidate structures, a template structure is chosen on the basis of its suitability according to various criteria such as the level of similarity between the query and template sequences, the experimental quality of the solved structures, the presence of ligands or cofactors and so on. Ideally, a large segment of the query sequence should be covered by a single high-quality template, although in many cases, the available template structures will correspond to only one or more distinct structural domains of the protein.

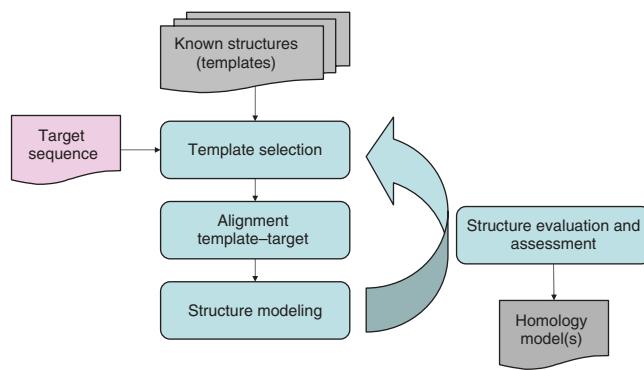


Figure 1 | The four main steps of comparative protein structure modeling: template selection, target-template alignment, model building and model quality evaluation.

Estimating the accuracy of a protein structure model is a crucial step in the whole process, as it is the quality of the model that determines its possible applications¹⁸. The quality of the obtained models will depend on the evolutionary distance between the target and the template proteins. It has been shown that there is a direct correlation between the sequence identity level of a pair of protein structures and the deviation of the C α atoms of their common core. The more similar two sequences are, the closer the corresponding structures can be expected to be and the larger the fraction of the model that can be directly inferred from the template⁴. As comparative models result from a structural extrapolation guided by a sequence alignment, the percentage of sequence identity between target and template is generally accepted as a reasonable first estimate of the quality of the structurally conserved core of the model. As a rule of thumb, the core C α atoms of protein models sharing 50% sequence identity with their templates will deviate by ~1.0 Å root mean square deviation from experimentally elucidated structures. Although the atomic coordinates of the three-dimensional model, for regions of the target protein aligned to the template, can be modeled on the basis of the information provided by template structure^{5–7}, regions that are not aligned with a template (insertions/deletions) require specialized approaches^{19–22}. Unaligned regions of the target that are modeled using *de novo* techniques, such as loops, will on average be less accurate than structurally conserved regions of the model on the basis of information derived directly from the template.

As the percentage identity falls below ~30% (in the so-called ‘twilight zone’), model quality estimation on the basis of sequence identity becomes unreliable, as the relationship between sequence and structure similarity gets increasingly dispersed^{18,23}. With decreasing sequence identity, alignment errors and the incorrect modeling of large insertions become the major source of inaccuracies. Correctly aligning the target sequence with the template is a crucial step and one of the primary sources of errors in the whole modeling procedure. The development of algorithms for sequence comparison and alignment is a major topic of study in bioinformatics, and the advancements in this field have been comprehensively reviewed in ref. 24. For estimating the overall quality of protein structure models and comparing predictions on the basis of alternative alignments^{25–27}, scoring functions such as statistical potentials of mean force have been developed. Methods that allow identifying local errors in models are currently an active field of research^{28–32}. The stereochemical plausibility of the generated models can be assessed using tools such as PROCHECK³³ and WHATCHECK³⁴, which help to identify amino-acid conformations deviating from expected values for structural features such as bond lengths and angles.

Accuracy and limitations of homology modeling

Comparative modeling relies on establishing an evolutionary relationship between the sequence of the protein of interest and other members of the protein family, whose structures have been solved experimentally by X-ray or NMR. For this reason, the major limitation of this technique is the availability of homologous templates, i.e., only regions of the protein corresponding to an identified template can be modeled accurately. As experimental protein structures are often available only for individual structural domains, it is often not possible to infer the correct relative domain orientation in a model.

Modeling oligomeric proteins, i.e., complexes composed of more than one polypeptide chain, may be straightforward in cases where the complex of interest is similar to a homologous complex of known structure. However, this situation is relatively rare, as most experimental structures in the PDB consist of individual proteins rather than complexes. Modeling complexes from individual components is a daunting task³⁵ and rarely successful without integrating additional information about the assembly³⁶.

Comparative protein modeling techniques rely on structural information from the template to derive the structure of the target. Large structural changes, e.g., caused by mutations, insertions, deletions and fusion proteins, are therefore, in general, not expected to be modeled accurately by comparative techniques. Nonetheless, homology models of a protein under investigation can provide a valuable tool for the interpretation of sequence variation and the design of mutagenesis experiment to elucidate the biological function of proteins^{16,17,37}.

The reliability of different protein modeling methods can be objectively evaluated by examining the quality of predictions made during blinded tests. For example, in CASP7, the ‘Community Wide Experiment on the Critical Assessment of Techniques for Protein Structure Prediction’ in 2006, predictions for 108 homology modeling targets were analyzed in detail to identify progress and limitations of current protein structure prediction methods¹¹. Particular emphasis was also given to the analysis of the results of automated prediction servers whose accuracy has significantly increased over the last years. Details about the participating servers and public accessibility are given in Table 1 of ref. 38. Similarly, the EVA³⁹ project provides a continuous assessment of the stability and accuracy of automated modeling servers on the basis of a large number of blind predictions. SWISS-MODEL was the first comparative modeling server to join the EVA project in May 2000. All results of this evaluation are available at <http://eva.compbio.ucsf.edu/~eva/>.

Availability

SWISS-MODEL workspace⁴⁰ can be freely accessed by the biological community on the Web at <http://swissmodel.expasy.org/workspace/>. SWISS-MODEL has been the first automated modeling server publicly available⁷. In the meantime, similar services have been developed by other groups, e.g., ModPipe⁴¹, 3D-JIGSAW⁴² or M4T⁴³. For a more complete listing of other publicly available comparative modeling servers, we refer the readers to the annual Nucleic Acids Research Web server issue⁴⁴.

SWISS-MODEL workspace

Each of the four steps in homology modeling requires specialized software as well as access to up-to-date protein sequence and structure databases. The SWISS-MODEL workspace⁴⁰ integrates the software required for homology modeling and databases in an easy-to-use, Web-based modeling environment. The workspace assists the user in building and evaluating protein homology models at different levels of complexity—depending on the difficulty of the individual modeling task. A highly automated modeling procedure with a minimum of user intervention is provided for modeling scenarios where highly similar structural templates are available^{7,14,45–47}. For more complex modeling tasks where target and template have lower sequence similarity, expert users are given control over the several steps of model building to construct a

protein model that is optimally adapted to their scientific problem⁴⁸. Modeling can be performed from within a Web browser without the need for downloading, compiling and installing large program packages or databases. The results of different modeling tasks are presented in a graphical summary. As quality evaluation is indispensable for a predictive method like homology modeling, every model is accompanied by several quality checks. In the following sections, we describe the main components of the SWISS-MODEL workspace.

Tools for target sequence feature annotation. Functional and structural domain annotation of the target sequence of interest is the first step toward the identification of a suitable template for building its three-dimensional model. Individual structural domains of multidomain proteins often correspond to units of distinct molecular function^{49–51}. Furthermore, the sensitivity of profile-based template detection methods can be enhanced when the search is performed at the domain level rather than searching the whole protein sequence. IprScan, a PERL-based InterProScan^{52,53} utility, has been integrated in the SWISS-MODEL workspace for the analysis of the domain architecture of the target protein and the annotation of its functional features. Prediction tools for secondary structure⁵⁴, disorder⁵⁵ and transmembrane (TM) regions⁵⁶ complement the tools for protein sequence analysis and aid the selection of suitable modeling templates for specific regions of the target proteins. In the twilight zone of sequence alignments, applying secondary structure prediction to the protein of interest may help deciding whether a putative template shares essential structural features. Intrinsically unstructured regions in proteins have been associated with numerous important biological cellular functions, from cell signalling to transcriptional regulation^{57,58}; several examples of such disordered regions undergoing the transition to an ordered state upon binding their ligand proteins have been reported⁵⁹. Prediction of disordered and transmembrane regions therefore complement the analysis of protein domain boundaries and functional annotation of the target protein.

Tools for template identification. The SWISS-MODEL workspace provides a set of increasingly sensitive sequence-based search methods for template detection that are applied depending on the evolutionary divergence between the target protein and the closest structurally characterized template protein. Close homologs of the target can be identified using a gapped BLAST⁶⁰ query against the SWISS-MODEL Template Library (SMTL)⁴⁰. When no closely related templates are found, or can be identified only for some segments of the target protein, more sensitive approaches for detecting evolutionary relationships are provided. (i) In the iterative profile Blast approach⁶⁰, which has been initially introduced as PDB-Blast by Godzik and coworkers, a profile for the target sequence is compiled from homologous sequences by iterative searches of the NR database⁶¹ and used subsequently to search SMTL for homologous structures. (ii) Alternatively, to detect more distantly related template structures, a Hidden Markov Model (HMM) for the target sequence is built on the basis of a multiple sequence alignment, similarly to the profile Blast approach discussed above. The HMM for the target sequence is subsequently used to search against the template library of HMMs generated for a nonredundant set of the sequences of the

SMTL template library culled at 70% sequence identity. HMM building, calibration and library searches are performed using the HHSearch (v. 1.5.01) software package⁶². For the selection of a suitable template, the following issues need to be considered:

1. The selection of the best template structure not only depends on sequence similarity but should also take into account other factors, such as experimental quality, bound substrate molecules or different conformational states of the template. For example, certain proteins undergo large conformational changes upon substrate binding as observed, e.g., between the apostructure and ATP- and ADP-bound forms of enzymes in the nucleotide kinase family⁶³. Depending on the planned model applications, such as structure-based ligand design, it is necessary to choose a structural template in the correct conformation.
2. For low-homology templates, the InterPro functional annotation of the target sequence can be used to verify that putative templates share essential functional features.
3. If the target-template alignment falls within the ‘twilight zone’ of sequence alignments (i.e., below 30% sequence identity), secondary structure prediction of the target protein may help to decide whether a putative template shares structural features with your protein and may therefore be used as template.
4. Predicted disorder regions may indicate the boundaries of protein domains and provide additional functional annotation of the protein.

Modeling. *Automated mode:* If the alignment between the target and the template sequences displays a sufficiently high similarity, a fully automated homology modeling approach can be applied. As a rule of thumb, automated sequence alignments are sufficiently reliable when target and template share more than 50% of sequence identity. Submissions in ‘Automated mode’ require only the amino-acid sequence or the UniProt² accession code of the target protein as input data. The hierarchical approach for template detection of the modeling pipeline will automatically select suitable templates on the basis of a Blast search or using an adapted sequence-to-HMM comparison HHSearch protocol⁶⁴. In cases where several similar template structures are available, the automated template selection will favor high-resolution template structures with good-quality assessment. Optionally, a specific template from the SMTL template library can be specified.

Alignment mode: For more distantly related target and template sequences, the number of errors in automated sequence alignments increases²³. This poses a major problem for automated homology modeling, as current methods are not capable of recovering from an incorrect input alignment. In many molecular biology projects, multiple sequence alignments are often the result of extensive theoretical and experimental exploration of a family of proteins. Such alignments can be used for comparative modeling using the ‘Alignment mode’ if at least one of the member sequences represents a protein for which the three-dimensional structure is known. The ‘Alignment mode’ allows the user to test several alternative alignments and evaluate the quality of the resulting models to achieve an optimal result.

Project mode: In the so-called ‘twilight zone’ of sequence alignments, when the sequence identity between target and template is below 30%, it is advisable to visually inspect and manually edit the

PROTOCOL

target–template alignment. This will lead to a significant improvement of the quality of the resulting model. The program DeepView (Swiss-PdbViewer)⁴⁸ can be used to display, analyze and manipulate modeling projects. DeepView project files contain one or more superposed template structures and the alignment between the target and template(s). Project files are also generated by the workspace template selection tools and are the default output format of the modeling pipeline. Project files with modified alignments can then be saved to disk and submitted as ‘Project mode’ to the workspace for model building by the SWISS-MODEL pipeline, thereby giving the user full control over essential modeling parameters: several template structures can be compared simultaneously to identify structurally conserved and variable regions and select the most suitable template. The placement of insertions and deletions in the target–template alignment can be visualized in their structural context and adjusted accordingly.

Protein structure assessment and model quality estimation. The percentage of sequence identity between target and template is generally accepted as a reasonable first estimate of the quality of a model. However, the accuracy of individual models may vary significantly from the expected average quality due to suboptimal target–template alignments, low template quality, structural flexibility or inaccuracies introduced by the modeling program. Individual assessment of each model is therefore essential. As a global indicator of the quality of a given model, the results of QMEAN⁶⁵, a composite scoring function for model quality estimation, and DFIRE³⁰, an all-atom distance-dependent statistical potential, are provided in the SWISS-MODEL workspace. However, a good global score does not guarantee that important functional sites of a protein have been modeled correctly. Therefore,

tools for local model quality estimates are included: graphical plots of ANOLEA mean force potential²⁸, GROMOS empirical force field energy⁶⁶ and the neural network-based approach ProQres³² are provided as indicators for local model quality.

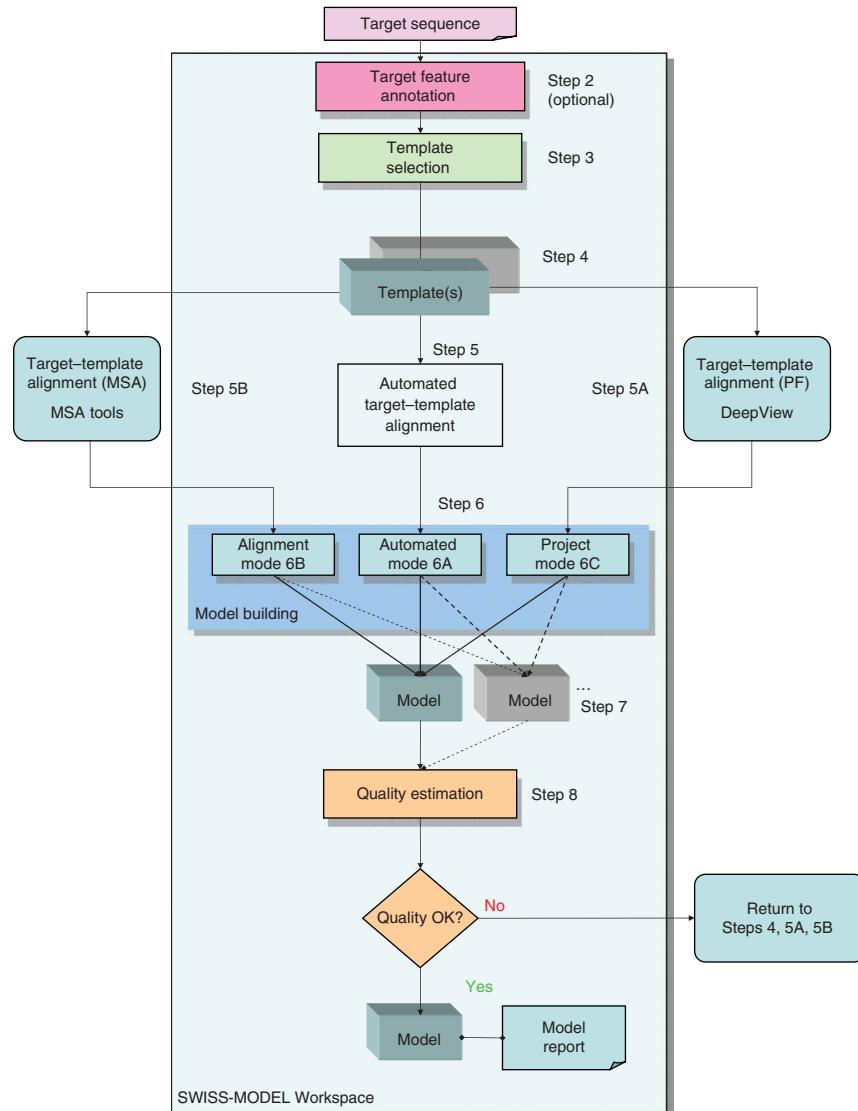


Figure 2 | Workflow of comparative protein structure modeling using SWISS-MODEL workspace. Starting from the amino-acid sequence of a ‘Target’ protein, three alternative routes for model building are provided—depending on the difficulty of the modeling task. Individual steps are described in detail in PROCEDURE.

Finally, Whatcheck³⁴ and Procheck³³ reports enable the user to assess the conformational quality of both models and template structures.

In this protocol, we describe in a step-by-step procedure (Fig. 2) how users can benefit from the integrated design of the SWISS-MODEL workspace to build and assess the accuracy of homology models.

MATERIALS

EQUIPMENT

- Amino-acid sequence of the protein to be modeled
- A computer with access to the Internet and a Web browser
- A multiple protein sequence alignment, including at least the sequences of the target protein and the template structure

(optional; see Step 6B in PROCEDURE for information on sequence alignment formats)

- DeepView for protein structure analysis and visualization (optional software). DeepView can be freely downloaded from the ExPASy website (<http://www.expasy.org/spdbv/>)

PROCEDURE**Access and personal user account**

1| Access and create a personal account for the SWISS-MODEL workspace at <http://swissmodel.expasy.org/workspace/>. The user data are stored in a password-protected personal user space, which is identified by the user's email address. It is also possible to access the workspace system anonymously without providing an email address. However, it is then necessary to bookmark the URLs of individual work units in the Web browser to be able to retrieve the results once the browser session has been closed. Once logged in your personal user account, individual modeling tasks are organized in work units under 'Workspace'; their current computational status is represented graphically (**Fig. 3**).

Sequence feature annotation

2| Examine your target sequence. The results of this analysis will assist you in deciding which of the possible template(s) (obtained in Step 3) to use to build homology model(s). Submit your protein sequence (as plain text, in FASTA format or its UniProt Accession Code) to one or more of the tools available in the 'Sequence Features Scan' session of the server, you find under 'Tools': use option A for InterPro domain scan, option B for PsiPred, option C for DISOPRED and option D for MEMSAT.

(A) InterPro domain scan

(i) InterPro domain scan⁵² identifies known protein domains and functional sites of the target sequence and possibly assigns the protein to a specific family. The following databases, currently part of the InterPro scan method, can be selected: HMMERfam—the target sequence is searched against the Pfam⁶⁷ database, a large collection of multiple sequence alignments and hidden Markov models covering many common protein domains and families; ProfileScan—the target sequence is searched against the profiles collection of PROSITE⁶⁸, a database of protein families and domains, and it consists of biologically significant sites, patterns and profiles that help in identifying to which known protein family a sequence belongs; ScanRegEx—*the target sequence is scanned for biologically significant patterns contained in the PROSITE database collection, e.g., enzyme catalytic sites, phosphorylation sites and so on.*

(ii) The occurrence of domains and functional sites are displayed on the target sequence. Domain boundaries and links to InterPro database instruct about distinctive features of a given functional domain or provide documentation relative to a specific protein family.

(B) PsiPred

(i) PsiPred⁵⁴ predicts secondary structure elements of the target sequence. The graphical representation shows the probability of a given residue of being part of an alpha helix (H), extended beta strand (E) or a coil region (C).

(C) DISOPRED

(i) DISOPRED2⁵⁵ predicts the occurrence of disordered regions in the target protein. The probability of being disordered (ranging from 0 to 1) is plotted for each position in the sequence. The 'output' and 'filter' curves represent the raw and filtered scores from the linear SVM classifier (DISOPREDsvm), respectively. Both outputs from DISOPREDsvm are included to allow the user to identify shorter, low-confidence predictions of disorder. Asterisks (*) and dots (.) denote predicted disorder and order, respectively. DISOPRED2 predictions are given at a default false-positive rate threshold of 2%, but this value can be changed by the user.

? TROUBLESHOOTING**(D) MEMSAT**

(i) MEMSAT⁵⁶ predicts the occurrence of putative TM segment in the protein. Central TM helix segments are indicated with 'X' in the output sequence. Information about the predicted TM topology is also provided.

The screenshot shows the 'Workspace' section of the SWISS-MODEL WORKSPACE. At the top, there are navigation links: [Workspace] [Modelling] [Tools], [Repository] [General Info] [Links] [Help], and [Settings] [logout]. Below this is a table titled 'Workspace' with a red question mark icon. The table has columns: Workunit, Type, Title, and Status. There are seven rows of data:

Workunit	Type	Title	Status
P000001	Sequence Feature Scan	AlkD - Q2PAD8	
P000002	Template Identification	AlkD - Q2PAD8	
P000003	Modelling - Automated Mode	AlkD - Q2PAD8	
P000004	Sequence Feature Scan	PKC delta - P83099	
P000005	Structure Assessment	PKC delta - P83099	
P000006	Modelling - Project Mode	PKC delta - P83099	
P000007	Modelling - Alignment Mode	LDL Domain	

Below the table, there is a legend for 'Symbols':

- submission not finished
- queued
- running
- failed/stopped
- finished
- 7 days left
- ... 1 days left
- will be deleted
- keep 7 days longer
- delete workunit

At the bottom of the workspace area, there is a URL: swissmodel.expasy.org/workspace and a link: [SWISS-MODEL Team].

Figure 3 | Example of a personal user workspace. In SWISS-MODEL workspace, individual modeling tasks are organized in work units; their current computational status is represented graphically.

PROTOCOL



Template identification and target template alignment

3| Submit your target sequence (as plain text, in FASTA format or its UniProt Accession Code) to one or more of the template identification tools of the ‘Template Identification’ session you find under ‘Tools’. The server provides access to a set of increasingly complex and computationally demanding methods: use option A for BLAST, option B for PSI-BLAST and option C for HMM-HMM-based searching.

(A) BLAST

(i) Closely related homologous templates are identified by running a gapped BLAST search⁶⁰. Adjust standard BLAST parameter like *E*-value cutoff or the choice of the substitution matrix to alter the sensitivity or specificity of your search.

? TROUBLESHOOTING

(B) PSI-BLAST

(i) More divergent template structures can be identified using iterative, profile-based BLAST⁶⁰.

(ii) Profile generation: selectivity and sensitivity of the search can be adjusted in the profile generation step by altering the number of iterations and the inclusion threshold for building the target profile. A more permissive *E*-value threshold and a greater number of iterations will increase the sensitivity of your search. Note that the inclusion of false positives, i.e., proteins that do not belong to the family of interest, during profile building can cause a drift in the search and lead to an increased false-positive rate among your hits.

(iii) *Profile search*: in the template library search step, the balance between selectivity and sensitivity can be adjusted by the choice of the substitution matrix.

? TROUBLESHOOTING

(C) HMM-HMM-based searching

(i) Distantly related templates can be identified using HMM-based profile matching using HHSearch⁶². A profile of the query sequence is generated and used for identifying matching HMM profiles in the template library. As this approach is computationally more intensive, compared with methods described in Steps 3A and B, the query is performed against a reduced version of the PDB database (culled at the 70% sequence identity level).

4| Select one or more structures from the result hit list as template to build comparative models. Results of template selection (Step 3) and domain identification (Step 2A) are displayed in a condensed graphical overview. This combined view allows you to analyze template coverage with respect to the domain boundaries and to identify templates spanning one or more domains of the target. Bars indicating matching regions will link you to the underlying target-template alignment and links to the SMTL library are available to facilitate the choice of a suitable template.

? TROUBLESHOOTING

5| Once you have selected one or more suitable templates, the following options are possible to improve the initial target-template alignment: Option A—DeepView Project or option B—alternative sequence alignment methods.

▲ CRITICAL STEP This is a particularly critical step, as homology modeling techniques cannot recover from an incorrect starting target-template alignment.

(A) DeepView Project

(i) The target-template sequence alignments generated by the different template database search techniques can be used as the basis for the subsequent model creation. The alignments can be downloaded as DeepView project file, which contains the target sequence aligned to the template structure.

(ii) The program DeepView allows you to display and analyze the alignment in the structural context of the template to manually adjust misaligned regions.

? TROUBLESHOOTING

(iii) Once you have finished editing the alignment, save the project file on the local disk and submit it to the ‘Project Mode’ of the Modeling session for model building (Step 6C).

(B) Alternative sequence alignment methods

(i) You might also want to apply alternative sequence alignment methods by using multiple sequence alignment programs to align the target and the template sequences obtained in Step 3. For a list of the most widely used sequence alignments tools, please refer to ref. 24, Table 1 therein.

(ii) The obtained sequence alignment between target and template (and additional homologous proteins) can be submitted to the ‘Alignment mode’ of the modeling session for model building (Step 6B).

Modeling

6| To obtain an homology model of your target sequence, you can choose among three different approaches—accessible through the ‘Modeling’ session of the server—whose applicability depends primarily on how distantly related your protein and the homologous template are: option A—automated mode; option B—alignment mode; or option C—project mode.

(A) Automated mode

- (i) In cases where the target–template similarity is sufficiently high to allow for unambiguous sequence alignment, homology modeling can be fully automated. Submit your target sequence as plain text, FASTA format or its UniProt accession code.
- (ii) Optionally, a specific template, e.g., identified by a Blast search in Step 3, can be specified by its PDB identifier and chain ID. Make sure that the specified template ID is present in the SWISS-MODEL template library.

? TROUBLESHOOTING**(B) Alignment mode**

- (i) With decreasing sequence similarity between target and template, the number of errors in automatically generated sequence alignments increases. Therefore, you might choose to submit an alignment generated by alternative sequence alignment tools (Step 5B). Provide a pairwise or multiple sequence alignment as input alignment in FASTA, MSF, ClustalW, PFAM or SELEX format.

? TROUBLESHOOTING

- (ii) After the alignment has been converted into a standard format, indicate which sequence corresponds to the target protein and which corresponds to a protein with known structure in the template library.

? TROUBLESHOOTING

- (iii) Submit your alignment for model calculation. Note that the SWISS-MODEL pipeline used for the modeling process might introduce minor heuristic modifications to improve the placement of insertions and deletions during model building.

(C) Project mode

- (i) In the twilight zone of sequence alignments, visual inspection and manual manipulation of the target–template alignment can significantly improve the quality of the resulting model. Using Project mode, you can submit Project files that you have obtained in Step 5A after adjusting the alignment in DeepView.

? TROUBLESHOOTING

- (ii) In project mode, you can also submit projects generated directly inside DeepView. With this option, it is possible to generate models using templates that are not part or not yet present in the SMTL library.

- (iii) *Oligomer modeling:* template-based modeling of oligomeric assemblies (Fig. 4) is possible using DeepView and subsequently submitting the file to the Project Mode (Box 1).

7 | After completion of the modeling procedure, the results are stored in the workspace and, if specified in your personal setting, you will be notified of the completion. Coordinates of the model, the underlying alignment, log files and quality evaluations can be accessed and downloaded from the personal workspace (Fig. 5). The model coordinates are available in PDB or DeepView project file format. The latter allows you to further inspect and manually modify the target–template alignment. Modified project files can then be saved to disk and submitted as ‘Project mode’ to the workspace for a further iteration of the model-building cycle (Step 6C). Energy profiles from the ANOLEA statistical potential²⁸ as well as the GROMOS force field⁶⁶ are

```
>TARGET
QQQEPPPEPRITLTVGQPVTFLVDTGAQH
SVLTQNPGPLSDRSAVQGATGGKRYRWT
DRKVHLATGKVTHSFLVPDCPYPLLGRDL
LTKLKAQI;
QQQEPPPEPRITLTVGQPVTFLVDTGAQH
SVLTQNPGPLSDRSAVQGATGGKRYRWT
DRKVHLATGKVTHSFLVPDCPYPLLGRDL
LTKLKAQI
```

Figure 4 | Example of the input format for an oligomeric target sequence (Box 1).

BOX 1 | OLIGOMER MODELING

- (i) Determine the correct quaternary state of the template. Asymmetric units of PDB files often do not correspond to the correct biological assembly of a protein. Assembled coordinate files of the most likely biological assembly of the template can be retrieved from PQS⁷⁷, PISA⁷⁸ or the PDB.

! CAUTION Homology between two proteins is not necessarily sufficient to signify that they share the same quaternary structure.

- (ii) Download and save the oligomer template coordinates as PDB file to your local disk.
- (iii) Open the file in DeepView and remove all nonamino-acid groups, such as ions, ligands, OXT and so on from the template (unless they are at the very end of the file). You can do this by selecting the groups in the control panel of DeepView and remove the selected residues ('Build' menu).
- (iv) Make sure each chain (protomer) has a unique chain identifier, e.g., 'A', 'B' and so on. Coloring the molecule by chain helps to check. You can rename chains with DeepView ('Edit → Rename' menu).
- (v) Create a FASTA file with the target sequences for each chain, i.e., 'A', then 'B' and so on, separated by semicolons. For hetero-oligomers, make sure the order is the same as in the template (Fig. 4).
- (vi) Adjust target–template alignment in DeepView. Load the FASTA file into DeepView ('SwissModel' Menu) and generate a preliminary target–template alignment ('Fit → Fit raw sequence' Menu). Open the alignment window and adjust the alignment. Be sure not to align residues of different chains or to align amino-acid residues to ligand (HETATM) groups or the C-terminal oxygen group (OXT) in the template. Make sure all insertions and deletions are correctly positioned in the structural context.
- (vii) Save the project to your local disk and submit the file to the Project mode of SWISS-MODEL workspace for model building (Step 6C).

PROTOCOL

calculated in the course of the modeling procedure, and the corresponding plots are also accessible from the results page. The percentage sequence identity between target and template on the basis of the alignment used to build the model is also reported on the results page.

? TROUBLESHOOTING

Quality estimation

8| To estimate the quality of your model(s), submit it to the programs provided in the ‘Structure Assessment’ section under ‘Tools’, using the following options: option A for sequence identity; option B for stereochemistry check; option C for global model quality estimation; and option D for local model quality estimation. Some of the tools described below can help identify incorrect regions in the predicted protein structure. One possibility to cope with the uncertainties in comparative modeling (especially when the sequence identity between target and template is low) is to build multiple models on the basis of alternative templates and/or alignments (Step 3–6) and to subsequently select the most favorable model.

▲ **Critical Step** Protein structure models generated by comparative modeling may contain errors and thus need to be treated with caution. Often, the quality varies between parts of the model.

? TROUBLESHOOTING

(A) Sequence identity

(i) The percentage sequence identity between target and template is a good predictor of the accuracy of a model. Model accuracy steadily increases with increasing sequence identity.

(B) Stereochemistry check

(i) The stereochemical plausibility of the model can be analyzed with the tools WHATCHECK³⁴ and PROCHECK³³. Deviations from ideal stereochemical values are reported by these programs.

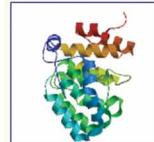
(C) Global model quality estimation

(i) The DFIRE statistical potential³⁰ as well as the QMEAN composite scoring function⁶⁵ both return a pseudo-energy for the entire model.

(D) Local model quality estimation

(i) The following tools are available for analyzing the local (per-residue) model reliability that can help in identifying potentially incorrect regions in the model: ProQres³²—an artificial neural network trained to predict the local model quality on the basis of the analysis of atom–atom contacts, residue–residue contacts, solvent accessibility surfaces and secondary structure propensities. The plot shows the local reliability of the model ranging from 0 (unreliable) to 1 (reliable) for each residue in the sequence; ANOLEA²⁸ is a statistical potential that can be used to analyze the packing quality of the model on the basis of nonlocal atomic interactions. The plot shows the ANOLEA pseudo-energy for each amino acid in the sequence. Negative values (colored in green) indicate that the amino acid is in a favorable environment, whereas positive values (colored in red) suggest that this part of the model has been incorrectly built; the GROMOS⁶⁶

Model Details: Segment 1



Model info:
modelled residue range: 12 to 223
based on template 2b6cA (2.10 Å)
Sequence Identity [%]: 31.604
Evalue: 0.00e-1

display model: as pdb - as DeepView project
download model: as pdb - as Deepview project - as text

Alignment [top]

TARGET	12	FIAHKNP EKAEPHARYM KNHFLFIGIQ TPERRQLLKD VIQIHTLPDP
2b6cA	3	t--lqfqknp etaakxsayx khqfqvafagip aperqalskq lkkeshtwpk
TARGET		h hhhhhhhh h hhhhhhhh h hhhhhhhh h
2b6cA		h hhhhhhhh h hhhhhhhh h hhhhhhhh h
TARGET	59	KDFRIIVREL WDLPEREFQAA ALDNMMQKYK KYINETHIPF LEELIVTKSW
2b6cA	51	eklcqeiaeay yqktereyyq vaidalalqnv qrfslleevva fkayvpqkaw
TARGET		hhhhhhhhhh h hhhh hhhhhhhh h hhhhhhhh h hhhhhhhh h
2b6cA		hhhhhhhhhh h hhhh hhhhhhhh h hhhhhhhh h hhhhhhhh h
TARGET	109	WDTVDSIVPFT FLGNFLQHP ELISAYIPKW IASDNIWLQR AAILFQLKYK
2b6cA	101	wdsdawrkf -fgsuvvalh telptifalp ygaenfvnr valnlqlxik
TARGET		hhhhhh hhh hhhh hhhh hhhh hhh hh hhh
2b6cA		hhhhhh hhh hhhh hhhh hhhh hhh hh hhh
TARGET	159	QKHDEELFW VIGQLHSSKE FFIQKAIGWV LREYAKTKPD VVWEYTVQNNE
2b6cA	150	ektnqdllkk aiiydrttee ffiqkaigws lrqysktntpq vveelxkelv
TARGET		hhhhhh hhhh hhhh hhhh hhhh hhhh hhhh hhhh
2b6cA		hhhhhh hhhh hhhh hhhh hhhh hhhh hhhh hhhh
TARGET	209	LAPLSRREAI KHIKE
2b6cA	200	lspaqregqs kylaka
TARGET		hhhhhh
2b6cA		hhhhhh

Figure 5 | Typical view of a SWISS-MODEL workspace result. In this example, a model for methylpurine-DNA glycosylase has been generated in automated mode. Upper panel: the green line represents the target sequence (237 residues). Blue lines indicate for which segments of the target models have been generated, in this case for residues 12–223. Middle panel: information on the template structure (2b6c, chain A) and quality (sequence identity, E-value) of the target–template sequence alignment shown in the lower panel. Model coordinates can be displayed within the Web browser window by clicking on the preview image or downloaded for manipulation with external software.

empirical force field is used to calculate the energy of each residue in the model. The graphical representation shows position in the sequence against the empirical force field energy. Negative values (colored in green) represent energetically favorable conformations, whereas positive values (colored in red) indicate unfavorable conformations.

? TROUBLESHOOTING

Troubleshooting advice can be found in **Table 1**.

TABLE 1 | Troubleshooting table.

Step	Problem	Solution
2B and C	Secondary structure prediction predicts a strand helix and the disorder prediction predicts the same region to be disordered	Examples of disordered regions undergoing the transition to an ordered state upon binding their ligand proteins have been reported ⁵⁹ . In case the predicted region aligns to a known template structure, check if the template has been solved in complex with a binding partner or is otherwise known to undergo structural rearrangement
3A	BLAST reports too many matches	Change the <i>E</i> -value cutoff for reporting hits to force BLAST to report only hits with a low <i>E</i> -value
	BLAST does not report any results	When no suitable templates are identified, or only parts of the target sequence are covered, two approaches for more sensitive detection of distant relationships among protein families are provided (3B and 3C)
3A and B	How will the choice of the substitution matrix influence the output of Blast/Profile Blast?	Use a substitution matrix adapted to the expected divergence of the searched sequences. For the BLOSUM family of matrices, the higher the matrix index is (i.e., BLOSUM 80), the more selective your search will be: it will exclude false positives but possibly miss true positives (closest to PAM120). Vice versa, the lower the index is (i.e., BLOSUM 45), the more sensitive the search will be: more true matches will be identified, but eventually, more false positives will be included (closest to PAM250)
3B	Profile Blast report too many matches	Change the <i>E</i> -value cutoff for reporting hits, or in the template library search step choose a library where the sequences of the templates are clustered at a lower sequence identity, e.g., ExPDB 70
4	The template identification methods cover only part of the sequence of my protein	The sensitivity of profile-based template detection methods (Step 3B and C) can be increased when the search is performed at the domain level rather than using the whole target sequence
	The template identification method predicts two templates with different structures	<i>Similar structures in different conformation:</i> protein structures can undergo large conformational changes, e.g., upon binding of ligand or post-translational modification. From the list of possible template structures, select the one most suitable for your application on the basis of the annotation provided, e.g., presence of ligands or cofactors and so on.
	No templates are found for the protein or domain of interest	<i>Template structures with different folds:</i> ambiguous results in fold assignment are expected if the evolutionary relationship between the target and possible template structures is too weak to be reliably detected by sequence-based methods, or no related template structure has been solved. Template structures with unclear evolutionary relationship to the target should not be used for homology modeling, unless supported by additional (experimental) evidence
	Can different templates, covering different regions of my protein, be combined to obtain a comparative model for the full length of the protein?	In this case, it is not possible to produce a reliable three-dimensional model for the protein by homology modeling. Alternatively, one can attempt to apply <i>de novo</i> prediction methods. However, the results of these types of prediction are currently far less accurate than comparative techniques and often not sufficient for specific biological applications ¹⁰
		Nonoverlapping templates cannot be combined, as the relative orientation of the different structures is unknown. It is, however, in principle possible, if the different templates are significantly overlapping (e.g., more than 20–40 amino acids). This feature is currently not supported by SWISS-MODEL workspace; users are referred to other modeling programs supporting this option, such as Modeller ⁶

(continued)

PROTOCOL

TABLE 1 | Troubleshooting table (continued).

Step	Problem	Solution
5A and 6C	Where can I find information on how to use the program DeepView?	Manuals for DeepView can be found on the program website: http://www.expasy.org/spdbv/ . It is highly recommended to start by following the tutorial provided by Gale Rhodes (University of Maine): http://www.usm.maine.edu/~rhodes/SPVTut/
6A and 6B	The template of interest is not present in the SMTL library	The SMTL library is updated biweekly, i.e., it might take a few days until newly released PDB structures are included. You can check if a specific PDB entry is available by querying the SMTL library in the ‘Tools’ section of the server. Alternatively, you can use DeepView to create a model project file on the basis of any template structure independently if this structure is part of PDB or SMTL
6B	Multiple sequence alignment is not correctly recognized by the server	Please make sure the alignment is in one of the supported formats. Use short unique names for sequences in the alignment; avoid nonalphanumeric characters. Good examples: ‘THN_DENCL’, ‘P01542’, ‘1crnA’ and so on
7	Unable to obtain a model from the server	In the majority of cases this is due to a poor alignment between the target and template sequences. Take time to carefully edit the alignment as suggested in Step 6A and B
	Despite careful checking of the target-template alignment, the server does not successfully deliver a model	This is usually the case if the target-template alignment contains large gaps (insertions and deletions) that were not successfully reconstructed. If this is the case, consider using other modeling programs, e.g., Modeller ⁶ . However, keep in mind that the results of <i>de novo</i> loop modeling techniques are less reliable than the template-based part of the model when using the model to answer the biological questions of interest
8	How to proceed when incorrect regions are identified and how to interpret them	There are several possible explanations for regions predicted as potentially incorrect (i.e., having low ProQres scores and/or high ANOLEA energies), e.g., alignment errors, incorrect modeling of insertions, unfavorable side-chain packing or false-positive assignment by the program itself. The identified regions of low reliability should be further analyzed by visually inspecting the alignment and the model. A model should always be interpreted with respect to its future application: a model with local errors outside the region of interest, such as the active site, can nevertheless be valuable for certain experiments. On the contrary, if, e.g., surface loops contain residues known to be involved in function, one needs to proceed with great caution when using the model for refining functional hypothesis

ANTICIPATED RESULTS

As an example, we apply the protocol described here to model the bacterial methylpurine-DNA glycosylase (AlkD, Uniprot AC: Q2PAD8) and the *Drosophila* putative protein kinase C delta type (Pkcdelta, UniProt AC: P83099). Please note that the results presented here illustrate a representative example at the time of writing. As sequence and structure databases are continuously updated, new template structures may become available at a later point and may lead to different, in general, better, modeling results.

AlkD is a DNA glycosylase⁶⁹ that functions as a DNA alkylation repair enzyme⁷⁰. AlkD belongs to a newly characterized DNA glycosylase superfamily⁷¹, for which no structures have yet been solved experimentally. Domain annotation (Step 2A) indicates that the protein belongs to a multihelical fold called the armadillo-like fold⁷². This is in agreement with the results of the secondary structure prediction analysis (Step 2B), predicting almost exclusively alpha-helices. A search for suitable structure templates (Step 3) yields a highly significant match spanning almost the entire length of the target protein, a putative DNA alkylation repair enzyme from *Enterococcus faecalis* (PDB: 2B6C) solved by the Midwest Center for Structural Genomics. This template was used by Dalhus *et al.*⁷³ to built a comparative model to elucidate the mechanism of AlkD.

The BLAST alignment between the target and the template displays only a single gap and a sufficiently high level of sequence similarity for it to be modeled using the automated mode of SWISS-MODEL workspace (Step 6A, **Fig. 5**). The location of the single gap in the target and sequence alignment can further be investigated in the structural context with the help of DeepView. The project file resulting from the automated step containing the template and the modeled protein can be opened in DeepView and the target-template alignment can be visualized with the help of the Alignment Window of DeepView. Secondary structure elements of the template can be highlighted in different colors using the color menu of the software. In the alignment resulting

from the automated mode, there is an inserted amino acid in the target sequence in a position that corresponds to an internal alpha helix residue of the template. We assume that the position of the gap could be improved by shifting it to the loop region connecting two adjacent alpha helices. The alignment can be edited directly within the alignment window of DeepView (Step 5A). The resulting modified project file is then saved locally and submitted to the Project Mode of the server for model building (Step 6C).

Dalhus *et al.*⁷³ predicted the location of a putative binding pocket in the model by residues conservation analysis of homologous proteins. The binding site of the obtained model was then used to design site-specific mutations to characterize the role of specific residues in the catalysis of DNA repair. Further insights into the mechanism of activity of the enzyme were gained by combining the obtained model with DNA coordinates extracted from the homologous protein AlkA.

In the second example, we build a model for a putative protein kinase C delta from *Drosophila* (Pkcdelta δ , UniProt AC: P83099). Domain annotation (Step 2A, Fig. 6) confirms that the target belongs to the protein kinase C (PKC) family⁷⁴. The PKC family members can be grouped into three classes: (i) the conventional PKCs (α , γ and $\beta 1$ and $\beta 2$), which requires diacylglycerol (DAG), phosphatidylserine (PS) and calcium for activation; (ii) the novel PKCs (δ , ϵ , η/λ , θ), which are activated by DAG and PS but are insensitive to calcium; and (iii) the atypical PKCs (ζ and ι/λ), which require only PS for full activity⁷⁵.

Several statistically significant matches for suitable templates are reported for the protein kinase domain (Step 4). Each template in the list is linked to the SMTL, and from there to other external resources, to allow for verification and a plausibility check. We have selected template 2JED (chain A) to build the model for our protein. Template 2JED corresponds to the crystal structure of the human kinase domain of the PKC θ , which belongs to the same class as our putative PKC δ . As in the previous example, the target-template alignment (derived from the iterative profile Blast search) is inspected with the help of DeepView, e.g., to verify that the residues corresponding to the typical signatures for the serine/threonine protein kinase active site (Prosite Accession number PS00108) and for the ATP-binding motif (Prosite Accession number PS00107) are correctly mapped in the target and template alignment. This can be done easily with the ‘search for PROSITE pattern’ function in the ‘Edit’ menu of DeepView. Subsequently, the target-template alignment is saved as project file and submitted to the Project Mode to obtain a model for the protein kinase domain of the protein (Fig. 7).

The two PE/DAG (Phorbol esters/diacylglycerol)-binding domains of the PKC δ protein can also be modeled on the basis of templates identified by iterative profile Blast and HMM-HMM search

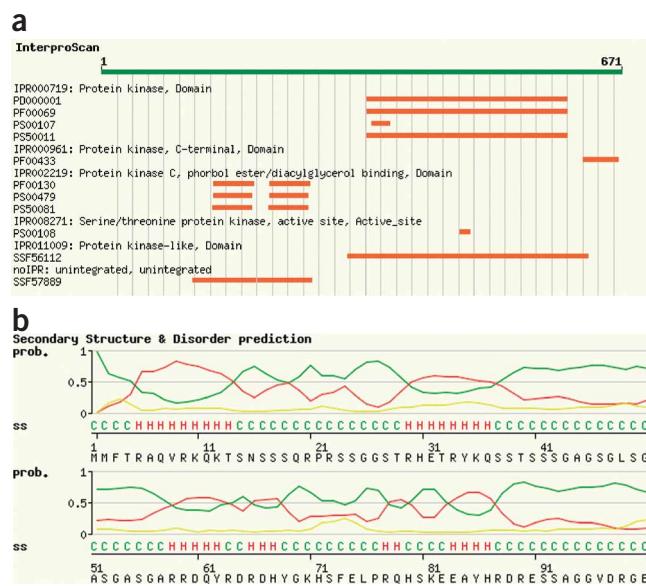


Figure 6 | Target sequence annotation for the putative protein kinase C delta from *Drosophila* (PKC δ , UniProt AC: P83099). **(a)** Three functional domains are identified using InterPro scan: two PE/DAG (Phorbol esters/Diacylglycerol)-binding domains and a PKC domain. **(b)** Secondary structure prediction for the N-terminal 100 residues of the target sequence.

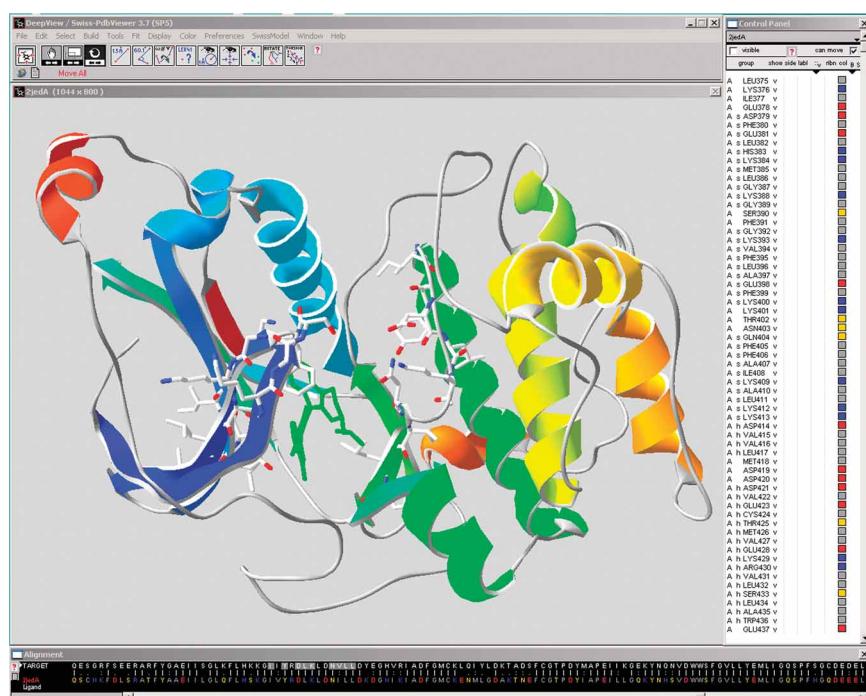


Figure 7 | Model of the kinase domain of the putative PKC δ from *Drosophila* shown as ribbon representation in DeepView colored from blue (N terminus) to red (C terminus). Characteristic residues of the Ser-Thr kinase active site and ATP-binding motif (identified by PROSITE) are shown as sticks. The position of the inhibitor molecule Nvp-Xaa228 in the template structure is highlighted in green.

methods. Few templates were detected by both methods and we decided to use the structure (PDB: 1PTQ⁷⁶) of the second activator-binding domain (PE/DAG) of an orthologous protein (the mouse PKC δ, UniProt AC: P28867) to build the models for the two PE/DAG domains. The alignment between the two PE/DAG domains of the PKC δ and the template is largely unambiguous, and the resulting model is of good quality according to the standard structure assessment tools (Step 8). Particular attention should be paid to the characteristic histidine and cysteine residues, which are assumed to be involved in the coordination of zinc ions. Correctly mapping these residues in the target–template sequence alignment ensures that they have a chemically plausible three-dimensional arrangement in the homology model.

Additional examples can be found in ref. 40 and in the tutorial provided on the SWISS-MODEL workspace website <http://swissmodel.expasy.org/workspace/tutorial/>.

ACKNOWLEDGMENTS We are grateful to Dr Michael Podvinec for his enthusiastic support and excellent coordination of the Scrum process for the SWISS-MODEL team. We are thankful for financial support of our group by the Swiss Institute of Bioinformatics (SIB).

Published online at <http://www.natureprotocols.com/>

Reprints and permissions information is available online at <http://npg.nature.com/reprintsandpermissions/>

1. Berman, H., Henrick, K., Nakamura, H. & Markley, J.L. The worldwide Protein Data Bank (wwPDB): ensuring a single, uniform archive of PDB data. *Nucleic Acids. Res.* **35**, D301–D303 (2007).
2. Wu, C.H. *et al.* The Universal Protein Resource (UniProt): an expanding universe of protein information. *Nucleic Acids. Res.* **34**, D187–D191 (2006).
3. Chothia, C. Proteins. One thousand families for the molecular biologist. *Nature* **357**, 543–544 (1992).
4. Chothia, C. & Lesk, A.M. The relation between the divergence of sequence and structure in proteins. *EMBO J.* **5**, 823–826 (1986).
5. Topham, C.M. *et al.* An assessment of COMPOSER: a rule-based approach to modelling protein structure. *Biochem. Soc. Symp.* **57**, 1–9 (1990).
6. Sali, A. & Blundell, T.L. Comparative protein modelling by satisfaction of spatial restraints. *J. Mol. Biol.* **234**, 779–815 (1993).
7. Peitsch, M.C. Protein modelling by e-mail. *BioTechnology* **13**, 658–660 (1995).
8. Tramontano, A. & Morea, V. Assessment of homology-based predictions in CASP5. *Proteins* **53** (Suppl. 6): 352–368 (2003).
9. Tress, M., Ezkurdia, I., Grana, O., Lopez, G. & Valencia, A. Assessment of predictions submitted for the CASP6 comparative modeling category. *Proteins* **61** (Suppl. 7): 27–45 (2005).
10. Jauch, R., Yeo, H.C., Kolatkar, P.R. & Clarke, N.D. Assessment of CASP7 structure predictions for template free targets. *Proteins* **69** (Suppl. 8): 57–67 (2007).
11. Kopp, J., Bordoli, L., Battey, J.N., Kiefer, F. & Schwede, T. Assessment of CASP7 predictions for template-based modeling targets. *Proteins* **69** (Suppl. 8): 38–56 (2007).
12. Kryshtafovych, A., Fidelis, K. & Moult, J. Progress from CASP6 to CASP7. *Proteins* **69** (Suppl. 8): 194–207 (2007).
13. Hillisch, A., Pineda, L.F. & Hilgenfeld, R. Utility of homology models in the drug discovery process. *Drug Discov. Today* **9**, 659–669 (2004).
14. Kopp, J. & Schwede, T. Automated protein structure homology modeling: a progress report. *Pharmacogenomics* **5**, 405–416 (2004).
15. Marti-Renom, M.A. *et al.* Comparative protein structure modeling of genes and genomes. *Annu. Rev. Biophys. Biomol. Struct.* **29**, 291–325 (2000).
16. Peitsch, M.C. About the use of protein models. *Bioinformatics* **18**, 934–938 (2002).
17. Tramontano, A. In *Computational Structural Biology* (eds. Schwede T. & Peitsch M.C.) (World Scientific Publishing, Singapore, 2008).
18. Baker, D. & Sali, A. Protein structure prediction and structural genomics. *Science* **294**, 93–96 (2001).
19. Soto, C.S., Fasnacht, M., Zhu, J., Forrest, L. & Honig, B. Loop modeling: sampling, filtering, and scoring. *Proteins* **70**, 834–843 (2008).
20. Rohl, C.A., Strauss, C.E., Chivian, D. & Baker, D. Modeling structurally variable regions in homologous proteins with rosetta. *Proteins* **55**, 656–677 (2004).
21. Fiser, A., Do, R.K. & Sali, A. Modeling of loops in protein structures. *Protein Sci.* **9**, 1753–1773 (2000).
22. Canutescu, A.A., Shelenkov, A.A. & Dunbrack, R.L. Jr. A graph-theory algorithm for rapid protein side-chain prediction. *Protein Sci.* **12**, 2001–2014 (2003).
23. Rost, B. Twilight zone of protein sequence alignments. *Protein Eng.* **12**, 85–94 (1999).
24. Dunbrack, R.L. Jr. Sequence comparison and protein structure prediction. *Curr. Opin. Struct. Biol.* **16**, 374–384 (2006).
25. Sommer, I., Toppo, S., Sander, O., Lengauer, T. & Tosatto, S.C. Improving the quality of protein structure models by selecting from alignment alternatives. *BMC Bioinformatics* **7**, 364 (2006).
26. Tress, M.L., Jones, D. & Valencia, A. Predicting reliable regions in protein alignments from sequence profiles. *J. Mol. Biol.* **330**, 705–718 (2003).
27. Vingron, M. Near-optimal sequence alignment. *Curr. Opin. Struct. Biol.* **6**, 346–352 (1996).
28. Melo, F. & Feytmans, E. Assessing protein structures with a non-local atomic interaction energy. *J. Mol. Biol.* **277**, 1141–1152 (1998).
29. Sippl, M.J. Calculation of conformational ensembles from potentials of mean force. An approach to the knowledge-based prediction of local structures in globular proteins. *J. Mol. Biol.* **213**, 859–883 (1990).
30. Zhou, H. & Zhou, Y. Distance-scaled, finite ideal-gas reference state improves structure-derived potentials of mean force for structure selection and stability prediction. *Protein Sci.* **11**, 2714–2726 (2002).
31. Fasnacht, M., Zhu, J. & Honig, B. Local quality assessment in homology models using statistical potentials and support vector machines. *Protein Sci.* **16**, 1557–1568 (2007).
32. Wallner, B. & Elofsson, A. Identification of correct regions in protein models using structural, alignment, and consensus information. *Protein Sci.* **15**, 900–913 (2006).
33. Laskowski, R.A., MacArthur, M.W., Moss, D.S. & Thornton, J.M. PROCHECK: a program to check the stereochemical quality of protein structures. *J. Appl. Cryst.* **26**, 283–291 (1993).
34. Hooft, R.W., Vriend, G., Sander, C. & Abola, E.E. Errors in protein structures. *Nature* **381**, 272 (1996).
35. Aloy, P., Pichaud, M. & Russell, R.B. Protein complexes: structure prediction challenges for the 21st century. *Curr. Opin. Struct. Biol.* **15**, 15–22 (2005).
36. Alber, F. *et al.* Determining the architectures of macromolecular assemblies. *Nature* **450**, 683–694 (2007).
37. Junne, T., Schwede, T., Goder, V. & Spiess, M. The plug domain of yeast Sec61p is important for efficient protein translocation, but is not essential for cell viability. *Mol. Biol. Cell* **17**, 4063–4068 (2006).
38. Battey, J.N. *et al.* Automated server predictions in CASP7. *Proteins* **69** (Suppl. 8): 68–82 (2007).
39. Koh, I.Y. *et al.* EVA: evaluation of protein structure prediction servers. *Nucleic Acids. Res.* **31**, 3311–3315 (2003).
40. Arnold, K., Bordoli, L., Kopp, J. & Schwede, T. The SWISS-MODEL workspace: a web-based environment for protein structure homology modelling. *Bioinformatics* **22**, 195–201 (2006).
41. Eswar, N. *et al.* Tools for comparative protein structure modeling and analysis. *Nucleic Acids. Res.* **31**, 3375–3380 (2003).
42. Bates, P.A., Kelley, L.A., MacCallum, R.M. & Sternberg, M.J. Enhancement of protein modeling by human intervention in applying the automatic programs 3D-JIGSAW and 3D-PSSM. *Proteins* (Suppl. 5): 39–46 (2001).
43. Fernandez-Fuentes, N., Madrid-Aliste, C.J., Rai, B.K., Fajardo, J.E. & Fiser, A. M4T: a comparative protein structure modeling server. *Nucleic Acids Res.* **35**, W363–W368 (2007).
44. Fox, J.A., McMillan, S. & Ouellette, B.F. Conducting research on the web: 2007 update for the bioinformatics links directory. *Nucleic Acids Res.* **35**, W3–W5 (2007).
45. Schwede, T., Diemand, A., Guex, N. & Peitsch, M.C. Protein structure computing in the genomic era. *Res. Microbiol.* **151**, 107–112 (2000).
46. Kopp, J. & Schwede, T. The SWISS-MODEL repository of annotated three-dimensional protein structure homology models. *Nucleic Acids Res.* **32**, D230–D234 (2004).
47. Schwede, T., Kopp, J., Guex, N. & Peitsch, M.C. SWISS-MODEL: an automated protein homology-modeling server. *Nucleic Acids Res.* **31**, 3381–3385 (2003).
48. Guex, N. & Peitsch, M.C. SWISS-MODEL and the Swiss-PdbViewer: an environment for comparative protein modeling. *Electrophoresis* **18**, 2714–2723 (1997).

49. Andreeva, A. *et al.* SCOP database in 2004: refinements integrate structure and sequence family data. *Nucleic Acids Res.* **32**, D226–D229 (2004).
50. Greene, L.H. *et al.* The CATH domain structure database: new protocols and classification levels give a more comprehensive resource for exploring evolution. *Nucleic Acids Res.* **35**, D291–D297 (2007).
51. Finn, R.D. *et al.* The Pfam protein families database. *Nucleic Acids Res.* **36**, D281–D288 (2008).
52. Zdobnov, E.M. & Apweiler, R. InterProScan—an integration platform for the signature-recognition methods in InterPro. *Bioinformatics* **17**, 847–848 (2001).
53. Mulder, N.J. *et al.* New developments in the InterPro database. *Nucleic Acids Res.* **35**, D224–228 (2007).
54. Jones, D.T. Protein secondary structure prediction based on position-specific scoring matrices. *J. Mol. Biol.* **292**, 195–202 (1999).
55. Jones, D.T. & Ward, J.J. Prediction of disordered regions in proteins from position specific score matrices. *Proteins* **53** (Suppl. 6): 573–578 (2003).
56. Jones, D.T., Taylor, W.R. & Thornton, J.M. A model recognition approach to the prediction of all-helical membrane protein structure and topology. *Biochemistry* **33**, 3038–3049 (1994).
57. Fink, A.L. Natively unfolded proteins. *Curr. Opin. Struct. Biol.* **15**, 35–41 (2005).
58. Radivojac, P. *et al.* Intrinsic disorder and functional proteomics. *Biophys. J.* **92**, 1439–1456 (2007).
59. Dyson, H.J. & Wright, P.E. Intrinsically unstructured proteins and their functions. *Nat. Rev. Mol. Cell Biol.* **6**, 197–208 (2005).
60. Altschul, S.F. *et al.* Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* **25**, 3389–3402 (1997).
61. Wheeler, D.L. *et al.* Database resources of the National Center for Biotechnology Information. *Nucleic Acids Res.* **33** Database Issue: D39–D45 (2005).
62. Soding, J. Protein homology detection by HMM-HMM comparison. *Bioinformatics* **21**, 951–960 (2005).
63. Muller, C.W., Schlauderer, G.J., Reinstein, J. & Schulz, G.E. Adenylate kinase motions during catalysis: an energetic counterweight balancing substrate binding. *Structure* **4**, 147–156 (1996).
64. Söding, J., Biegert, A. & Lupas, A.N. The HHpred interactive server for protein homology detection and structure prediction. *Nucleic Acids Res.* **33**, W244–248 (2005).
65. Benkert, P., Tosatto, S.C. & Schomburg, D. QMEAN: a comprehensive scoring function for model quality assessment. *Proteins* **71**, 261–277 (2008).
66. van Gunsteren, W.F. *et al.* *Biomolecular Simulations: the GROMOS96 Manual and User Guide* (VdF Hochschulverlag ETHZ, Zürich, 1996).
67. Bateman, A. *et al.* The Pfam protein families database. *Nucleic Acids Res.* **32**, D138–D141 (2004).
68. Hulo, N. *et al.* The PROSITE database. *Nucleic Acids Res.* **34**, D227–D230 (2006).
69. Stivers, J.T. & Jiang, Y.L. A mechanistic perspective on the chemistry of DNA repair glycosylases. *Chem. Rev.* **103**, 2729–2759 (2003).
70. Seeberg, E., Eide, L. & Bjoras, M. The base excision repair pathway. *Trends Biochem. Sci.* **20**, 391–397 (1995).
71. Alseth, I. *et al.* A new protein superfamily includes two novel 3-methyladenine DNA glycosylases from *Bacillus cereus*, AlkC and AlkD. *Mol. Microbiol.* **59**, 1602–1609 (2006).
72. Groves, M.R. & Barford, D. Topological characteristics of helical repeat proteins. *Curr. Opin. Struct. Biol.* **9**, 383–389 (1999).
73. Dalhus, B. *et al.* Structural insight into repair of alkylated DNA by a new superfamily of DNA glycosylases comprising HEAT-like repeats. *Nucleic Acids Res.* **35**, 2451–2459 (2007).
74. Nishizuka, Y. Membrane phospholipid degradation and protein kinase C for cell signalling. *Neurosci. Res.* **15**, 3–5 (1992).
75. Mellor, H. & Parker, P.J. The extended protein kinase C superfamily. *Biochem. J.* **332** (Part 2): 281–292 (1998).
76. Zhang, G., Kazanietz, M.G., Blumberg, P.M. & Hurley, J.H. Crystal structure of the cys2 activator-binding domain of protein kinase C delta in complex with phorbol ester. *Cell* **81**, 917–924 (1995).
77. Henrick, K. & Thornton, J.M. PQS: a protein quaternary structure file server. *Trends Biochem. Sci.* **23**, 358–361 (1998).
78. Krissinel, E. & Henrick, K. Inference of macromolecular assemblies from crystalline state. *J. Mol. Biol.* **372**, 774–797 (2007).

Corrigendum: Protein structure homology modeling using SWISS-MODEL workspace

Lorenza Bordoli, Florian Kiefer, Konstantin Arnold, Pascal Benkert, James Battey & Torsten Schwede

Nat. Protoc. **4**, 1–13 (2009); doi:10.1038/nprot.2008.197; published online 11 December 2008; corrected online 18 June 2009.

The version of this article initially published indicated that only Torsten Schwede was affiliated with the Swiss Institute of Bioinformatics in addition to the Biozentrum, University of Basel, Basel, Switzerland. However, all six authors are affiliated with both the Biozentrum and the Swiss Institute of Bioinformatics. The error has been corrected in the HTML and PDF versions of the article.

WEEK 6 READINGS

Mutation prediction by PolyPhen or functional assay, a detailed comparison of *CYP27B1* missense mutations

Minjing Zou · Essa Y. Baitei · Ali S. Alzahrani ·
Ranjit S. Parhar · Futwan A. Al-Mohanna ·
Brian F. Meyer · Yufei Shi

Received: 7 February 2011 / Accepted: 5 May 2011 / Published online: 21 May 2011
© Springer Science+Business Media, LLC 2011

Abstract Vitamin D-dependent rickets type 1 (VDDR-I) is caused by mutation in *CYP27B1*. The glycine residue at codon 102 is not conserved between human (G^{102}) and rodent (S^{102}). G102E mutation results in 80% reduction in its enzymatic activity but PolyPhen predicts benign change. It is not known whether G102S has any damaging effect on 1α -hydroxylase activity. We investigated the effect of *CYP27B1*^{G102S} on its enzymatic activity and compared mutation prediction accuracy for all known *CYP27B1* mutations among three free online protein prediction programs: PolyPhen, PolyPhen-2, and PSIPRED. G102S has no damaging effect on 1α -hydroxylase activity. G102D retained 30% enzymatic activity. All three programs correctly predicted damaging change for G102D. PolyPhen predicted benign change for G102S, whereas PolyPhen-2 and PSIPRED indicated possible damaging effect. Among 24 reported damaging mutations, PSIPRED, PolyPhen-2, and PolyPhen achieved 100%, 91.7% (22/24), and 75% (18/24) accuracy rate, respectively. The residues of incorrectly predicted mutations were not conserved. We conclude that G102D resulted in a significant reduction in 1α -hydroxylase activity, whereas G102S did not. PSIPRED

and PolyPhen-2 are superior to PolyPhen in predicting damaging mutations.

Keywords *CYP27B1* mutation · 1α -hydroxylase · Vitamin D · Rickets · P450c 1α

Introduction

Vitamin D plays an important role in calcium homeostasis and exists in two major forms: ergocalciferol (vitamin D2), and cholecalciferol (vitamin D3) [1]. Both forms need two-step hydroxylation at carbons 25 and 1 for activation. The first step occurs in the liver where vitamin D is hydroxylated to 25-hydroxyvitamin D [25(OH)D] by the hepatic 25-hydroxylase [1]. At least three enzymes have 25-hydroxylase activity: mitochondrial CYP27A1 [2], microsomal CYP3A4 [3], and CYP2R1 [4]. The second step occurs mainly in the kidney where 25(OH)D is hydroxylated by the mitochondrial vitamin D 1α -hydroxylase to biologically active hormone 1,25(OH) $_2$ D, which binds to its nuclear receptor and achieves its biological activities [1, 5, 6].

The human *CYP27B1* gene is located in chromosome 12q14 and encodes vitamin D 1α -hydroxylase [7–10]. It is around 5 Kb and composed of 9 exons and 8 introns [11]. Enzymatic deficiency of 1α -hydroxylase as a result of mutation in the *CYP27B1* gene causes vitamin D-dependent rickets type 1 (VDDR-I) [7, 12]. In our previous study, we reported a novel G102E mutation, which caused 80% reduction of 1α -hydroxylase activity [13]. However, PolyPhen program predicted a benign change, whereas PSIPRED program correctly predicted protein secondary structure change. In the present study, we created two additional mutations at the codon 102 (G102D and G102S) and investigated the correlation between mutation

M. Zou · E. Y. Baitei · B. F. Meyer · Y. Shi (✉)
Department of Genetics (MBC-03), King Faisal Specialist Hospital and Research Centre, P.O. Box 3354, Riyadh 11211, Saudi Arabia
e-mail: yufei@kfshrc.edu.sa

A. S. Alzahrani
Department of Medicine, King Faisal Specialist Hospital & Research Centre, Riyadh, Saudi Arabia

R. S. Parhar · F. A. Al-Mohanna
Department of Biological and Medical Research, King Faisal Specialist Hospital & Research Centre, Riyadh, Saudi Arabia

prediction and the effect of the mutation on vitamin D α -hydroxylase activity. We further compared the prediction accuracy rate for all known *CYP27B1* gene mutations among three protein structure prediction programs: PolyPhen, PolyPhen-2, and PSIPRED.

Materials and methods

Site-directed mutagenesis, cDNA cloning, and expression

The wild-type (wt) *CYP27B1* cDNA was used as a template for the creation of G102S and G102D mutants by site-directed mutagenesis [14]. The mutants were cloned into pcDNA3.1 expression vector (Invitrogen Co., CA) and stably expressed in CHO cells. The stable clones were pooled for gene expression as described previously [15]. T321R was reported to have no enzymatic activity and was used as a negative control [16].

Western blot analysis

60 µg of protein were loaded into a 12% SDS-polyacrylamide gel. Proteins were transferred to a PVDF membrane and probed with *CYP27B1* antibody (Santa Cruz Biotechnology, CA).

Analysis of 1 α -hydroxylase activity

CHO cells stably transfected with wt or the different mutants were seeded in 6-well plates overnight in growth medium and incubated in 2 ml serum-free medium with 0.1 and 1 µM 25(OH)D3 (Sigma, MO) for 1 and 4 h, respectively. The concentration of 1,25(OH) $_2$ D3 in the medium was measured by enzymeimmunoassay (EIA) according to the manufacturer's procedure (Immunodiagnostic Systems, AZ).

Mutation prediction

Three free online protein structure prediction programs: PolyPhen [17, 18], PolyPhen-2 [19], and PSIPRED [20, 21] were used to predict mutational consequence of *CYP27B1* (protein identifier # O15528). PolyPhen or PolyPhen-2 (an updated version) (<http://genetics.bwh.harvard.edu/pph/>) is a tool for prediction of possible impact of an amino acid substitution on the structure and function of a human protein using straightforward physical and comparative considerations [18, 19]. For PolyPhen and PolyPhen-2, three empirically derived outcomes were used: probably damaging (it is with high confidence supposed to affect protein function or structure), possibly damaging (it is

supposed to affect protein function or structure), and benign (most likely lacking any phenotypic effect). PSIPRED (<http://www.psipred.net>) is a simple and accurate secondary structure prediction method, incorporating two feed-forward neural networks which perform an analysis on output obtained from PSI-BLAST (Position Specific Iterated-BLAST) [20, 21]. For PSIPRED analysis, any predicted changes in protein secondary structure were considered to be damaging mutations.

Statistical analysis

Fisher's exact test was used to determine whether there is any statistical significance among different prediction programs.

Results

Partial protein alignment surrounding codon 102 of *CYP27B1* from different species

As shown in Fig. 1, the Glycine (G) residue at the position 102 is conserved across different species, even though mouse and rat have serine (S) residue at this position. The two rodents may represent an experiment of evolution. PolyPhen predicted benign change for G102S whereas PolyPhen-2 and PSIPRED indicated possible damaging effect. All the three programs predicted damaging change for G102D.

Expression of wild-type and *CYP27B1* mutants in CHO cells

Both wt and different mutants were transfected into CHO cells for stable expression. Figure 2a shows that wt and mutant *CYP27B1* proteins were expressed in CHO cells.

Effect of different *CYP27B1* mutants on 1 α -hydroxylase activity in CHO cells

1 α -hydroxylase activity was measured by its ability to convert 25(OH)D3 into 1, 25(OH) $_2$ D3 in CHO^{G102S}, CHO^{G102D}, CHO^{G102E}, CHO^{T321R}, and CHO^{WT} cells. CHO^{T321R} was used as a negative control. As shown in Fig. 2b, during 4 h incubation with 1 µM 25(OH)D3, 1, 25(OH) $_2$ D3 produced by CHO^{vector} cells was similar to that in culture medium (20 ± 3 fmol/ 10^5 cells vs. 18 ± 2), indicating that CHO cells have no 1 α -hydroxylase activity (the background value was thus subtracted for calculating enzymatic activity). CHO^{WT}, CHO^{G102E}, CHO^{G102D}, and CHO^{G102S} produced 405 ± 58 , 69 ± 11 , 115 ± 12 , 417 ± 72 fmol/ 10^5 cells of 1,25(OH) $_2$ D3 during 1 h

Fig. 1 Partial protein alignment of *CYP27B1* gene from human, monkey, cattle, dog, rat, mouse, and frog around position 102. The Glycine residue at the 102 is conserved across different species, even though mouse and rat have serine (S) residue at this position. Glycine (G) is a non-polar hydrophobic amino acid whereas serine (S) is a polar neutral amino acid

1	ref NP_000776.1	human (<i>homo sapiens</i>)	...AAPALVEELLRQE G PRPERCSFSPWTE...
2	ref XP_509175.2	chimpanzees [<i>Pan troglodytes</i>]	...AAPALVEELLRQE G PRPERCSFSPWTE...
3	ref XP_001116450.1	rhesus monkeys (<i>Macaca mulatta</i>)	...AAPALVEELLRQE G PRPERCSFSPWTE...
4	ref XP_588481.1	cattle (<i>Bos taurus</i>)	...AAPTLVEQLLRQE G PRPERCSFSPWTE...
5	ref NP_999160.1	pig [<i>Sus scrofa</i>]	...AAPTLVEQLLRQE G PLPERCSFSPWTE...
6	ref XP_538254.2	dog (<i>Canis lupus familiaris</i>)	...AAPALVEQLLRQE G PRPERCSFSPWAE...
7	ref XP_001490036.1	horse [<i>Equus caballus</i>]	...AAPTLIEQLLRQE G PRPERCSFSSWAE...
8	ref XP_001380742.1	opossum (<i>Monodelphis domestica</i>)	...AAPALIEQLLRQE G PHPERCSFSPWVE...
9	dbj BA22434.1	mouse (<i>Mus musculus</i>)	...ADPTLVEQLLRQE S HCPERCSFSSWAE...
10	ref NP_446215.1	rat (<i>Rattus norvegicus</i>)	...ADPALVEQLLRQE S HCPERCSFSSWSE...
11	ref XP_422077.2	red jungle fowl [<i>Gallus gallus</i>]	...ADRMVVAQVLRSE G RAPQRANMESWQE...
12	gb AAH77308.1	frog (<i>Xenopus laevis</i>)	...GDPEALQQLLRQE G KYPMRNKEDIWKA...
13	gb AAH94536.1	Xenopus tropicalis	...ASPELETLLRQE G KYPMRTDMFMWKE...

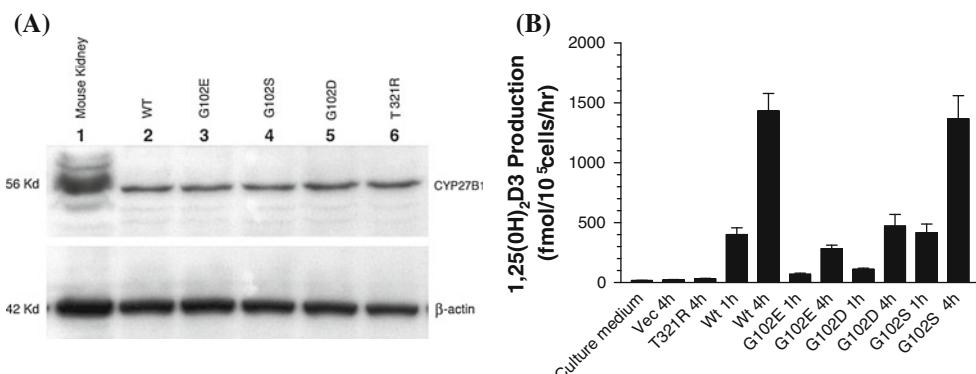


Fig. 2 CYP27B1 protein expression and 1 α -hydroxylase activity. **a** Western blot analysis of wild-type and mutant CYP27B1 expression in CHO cells. The CYP27B1 protein (56 Kd) was detected by a rabbit anti-human CYP27B1 antibody in CHO^{G102D}, CHO^{G102S}, CHO^{G102E}, CHO^{T321R}, and CHO^{WT}. The membrane was re-probed with a β -actin (42 Kd) antibody to monitor protein loading. **b** 1 α -hydroxylase activity in CHO cells expressing G102E, G102S, G102D, T321R, or

incubation, and 1432 ± 147 , 282 ± 30 , 472 ± 97 , and 1366 ± 195 fmol/ 10^5 cells of 1,25(OH)₂D₃ during 4 h incubation, respectively. T321R did not show any enzymatic activity (Fig. 2b). CHO^{G102D} lost about 70% of wt activity, whereas CHO^{G102S} maintained at least 95% enzymatic activity. These data indicate that G102S did not cause any significant change in the protein structure and function. Similar results were obtained with 0.1 μ M 25(OH)D₃ during 1 and 4 h incubation (data not shown).

Mutation prediction

Among 24 missense mutations known to cause reduction in 1 α -hydroxylase activity (Table 1), PolyPhen, PolyPhen-2, and PSIPRED correctly predicted damaging effect or protein secondary structure change in 18 (75%), 22 (91.7%), and 24 (100%) mutations, respectively. Eighteen mutations (18/24, 75%) were correctly predicted by all three programs. All misidentified mutations were located in amino acid residues not well conserved across different

wt CYP27B1 cDNA. The mutants and wt cDNAs were stably transfected into CHO cells. The cells were incubated with 1 μ M 25(OH)D₃ in 2 ml serum-free medium for 1 and 4 h, respectively. The concentration of 1,25 (OH)₂D₃ in the medium was measured by EIA. Data are expressed as means \pm SEM of three separate experiments

species (Table 1). We next analyzed 21 additional mutations in three different genes: *SRD5A2*, *PHEX*, and *CYP27A1*. Similar results were obtained as shown in Table 1. PolyPhen, PolyPhen-2, and PSIPRED correctly predicted damaging effect or protein secondary structure change in 15 (71.4%), 18 (85.7%), and 21(100%) mutations, respectively. There is no statistically significant difference between PolyPhen-2 and PolyPhen ($P = 0.059$, Chi-Square test), probably due to small sample size. These data suggest that PolyPhen-2 and PSIPRED are more accurate than PolyPhen in mutation prediction, especially in predicting mutations located in non-conserved residues. There are more structural changes in G102E and G102D as compared to G102S (Fig. 3). Interestingly, PSIPRED predicted structural change for G102S at two locations: 9 amino acid α -helix (residues 133–141, equivalent to α -helix C [22]) was shortened by one amino acid and C-terminal 19 amino acid α -helix (residues 456–474) was increased by one amino acid. The change is identical to that caused by G125E (Fig. 3).

Table 1 Mutation prediction by PolyPhen and PSIPRED for missense mutations

Mutation	Residue conservation	PolyPhen	PolyPhen-2	PSIPRED (protein secondary structure change)	Enzymatic activity (% of wt)	Reference
CYP27B1 ^{Q65H}	Not conserved	Benign	Probably damaging	Yes	0	[27]
CYP27B1 ^{G102E}	Conserved	Benign	Possibly damaging	Yes	20	[13]
CYP27B1 ^{G102D}	Conserved	Probably damaging	Possibly damaging	Yes	33	Current study
CYP27B1 ^{G102S}	Conserved	Benign	Possibly damaging	Yes	95	Current study
CYP27B1 ^{R107H}	Conserved	Probably damaging	Probably damaging	Yes	0	[12]
CYP27B1 ^{P112L}	Not conserved	Benign	Benign	Yes	0 ^a	[25]
CYP27B1 ^{G125E}	Conserved	Probably damaging	Probably damaging	Yes	0	[12]
CYP27B1 ^{P143L}	Not conserved	Possibly damaging	Possibly damaging	Yes	0	[16]
CYP27B1 ^{D164N}	Conserved	Probably damaging	Probably damaging	Yes	0	[16]
CYP27B1 ^{E189G}	Not conserved	Probably damaging	Probably damaging	Yes	22	[33]
CYP27B1 ^{E189K}	Not conserved	Benign	Possibly damaging	Yes	11	[33]
CYP27B1 ^{E189L}	Not conserved	Benign	Possibly damaging	Yes	1.6	[27]
CYP27B1 ^{T321R}	Conserved	Probably damaging	Probably damaging	Yes	0	[16]
CYP27B1 ^{S323Y}	Not conserved	Probably damaging	Probably damaging	Yes	0	[34]
CYP27B1 ^{R335P}	Not conserved	Probably damaging	Probably damaging	Yes	0	[12]
CYP27B1 ^{L343F}	Not conserved	Possibly damaging	Probably damaging	Yes	2.3	[33]
CYP27B1 ^{P382S}	Conserved	Probably damaging	Probably damaging	Yes	0	[12]
CYP27B1 ^{R389C}	Conserved	Probably damaging	Probably damaging	Yes	0	[16]
CYP27B1 ^{R389G}	Conserved	Probably damaging	Probably damaging	Yes	0	[33]
CYP27B1 ^{R389H}	Conserved	Probably damaging	Probably damaging	Yes	0	[27]
CYP27B1 ^{T409I}	Not conserved	Benign	Benign	Yes	0	[27]
CYP27B1 ^{R429P}	Not conserved	Possibly damaging	Possibly damaging	Yes	0	[27]
CYP27B1 ^{R453C}	Conserved	Probably damaging	Probably damaging	Yes	0	[27]
CYP27B1 ^{V478G}	Not conserved	Probably damaging	Probably damaging	Yes	0	[34]
CYP27B1 ^{P497R}	Conserved	Probably damaging	Probably damaging	Yes	0	[27]
SRD5A2 ^{L55Q}	Not conserved	Probably damaging	Probably damaging	Yes	b	[35]
SRD5A2 ^{G85D}	Not conserved	Possibly damaging	Possibly damaging	Yes	b	[36]
SRD5A2 ^{Q126R}	Conserved	Probably damaging	Probably damaging	Yes	b	[35]
SRD5A2 ^{G158R}	Not conserved	Possibly damaging	Benign	Yes	b	[35]
SRD5A2 ^{G183S}	Conserved	Possibly damaging	Probably damaging	Yes	b	[35]
SRD5A2 ^{G196S}	Conserved	Probably damaging	Probably damaging	Yes	b	[35]
SRD5A2 ^{A207D}	Not conserved	Probably damaging	Possibly damaging	Yes	b	[35]
SRD5A2 ^{S245Y}	Not conserved	Benign	Possibly damaging	Yes	b	[36]
SRD5A2 ^{R246W}	Conserved	Probably damaging	Probably damaging	Yes	b	[35]
PHEX ^{Y317F}	Not conserved	Benign	Benign	Yes	b	[37]
PHEX ^{P534L}	Conserved	Probably damaging	Probably damaging	Yes	b	[37]
PHEX ^{G579R}	Conserved	Probably damaging	Probably damaging	Yes	b	[37]
PHEX ^{Q621R}	Conserved	Probably damaging	Probably damaging	Yes	b	[37]
PHEX ^{A720T}	Not conserved	Benign	Possibly damaging	Yes	b	[37]
PHEX ^{F731Y}	Conserved	Benign	Probably damaging	Yes	b	[37]
PHEX ^{W749R}	Conserved	Probably damaging	Probably damaging	Yes	b	[37]
CYP27A1 ^{R127W}	Conserved	Probably damaging	Probably damaging	Yes	0	[38]
CYP27A1 ^{G145R}	Conserved	Probably damaging	Probably damaging	Yes	0	[38]
CYP27A1 ^{A216P}	Not conserved	Benign	Probably damaging	Yes	0	[38]

Table 1 continued

Mutation	Residue conservation	PolyPhen	PolyPhen-2	PSIPRED (protein secondary structure change)	Enzymatic activity (% of wt)	Reference
CYP27A1 ^{K259R}	Not conserved	Benign	Benign	Yes	0	[38]
CYP27A1 ^{T339M}	Conserved	Probably damaging	Probably damaging	Yes	0	[38]

^a Functional assay was not performed but patient had compound heterozygous mutations of P112L and R389H and severe rickets with seizures. The enzymatic activity is predicted to be severely compromised

^b Functional assay was not performed but are disease-causing mutations

Defects in *CYP27A1* cause cerebrotendinous xanthomatosis, a rare autosomal recessive lipid storage disease due to steroid 27-hydroxylase deficiency

Defects in *SRD5A2* (protein identifier #: P31213) cause pseudovaginal perineoscrotal hypospadias also known as male pseudohermaphroditism due to 5-alpha-reductase deficiency

Defects in *PHEX* (protein identifier #: P78562) cause X-linked hypophosphatemic rickets due to phosphate-regulating neutral endopeptidase deficiency

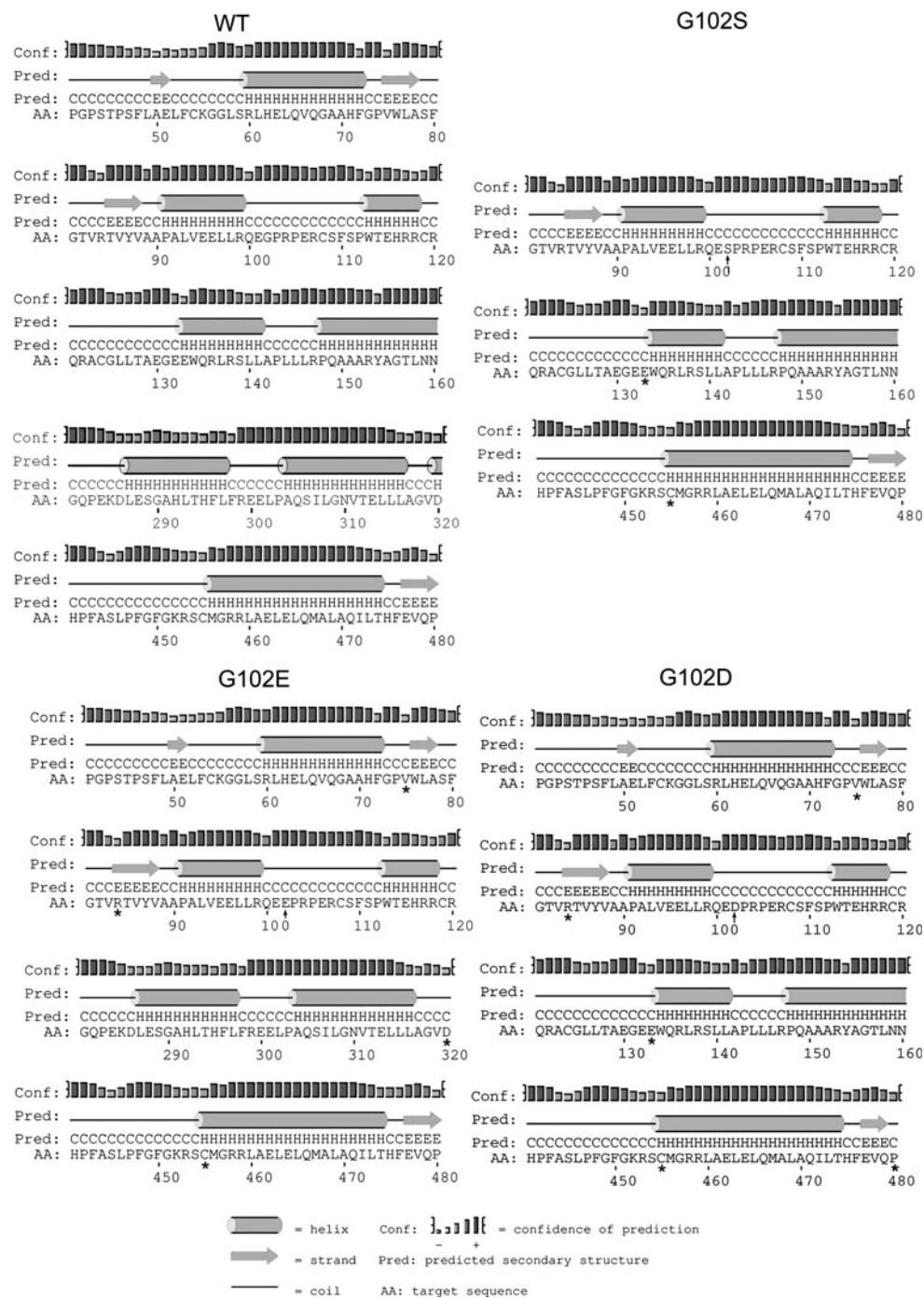
Discussion

Both human and mouse *CYP27B1* share significant protein sequence homology [7, 9, 23, 24]. Among 24 damaging missense mutations found in human *CYP27B1* [25], 19 occurred in amino acid residues common to both human and mouse, suggesting that these residues are important for enzymatic activity. However, there are five mutations located in residues that are not conserved: P112 in human versus S112 in mouse, G102 versus S102, T409 versus S408 (the mouse is one amino acid short as compared to human), and R429 versus N428. The changes from hydrophobic amino acids proline (P) or glycine (G) to neutral amino acid serine (S), neutral amino acid threonine (T) to serine, and basic hydrophilic amino acid arginine (R) to neutral amino acid asparagine (N) may not significantly affect protein structure and function. The current study has confirmed that the substitution of hydrophobic amino acid glycine (G) to neutral amino acid serine (S) at codon 102 would not cause any damaging effect, whereas the substitution of G to either hydrophilic amino acid aspartic acid (D) or glutamic acid (E) would result in detrimental effect on vitamin D 1 α -hydroxylase activity. S408 of mouse *CYP27B1* (corresponding to T409 of human *CYP27B1*) is involved in substrate binding by forming a hydrogen bond with 25-hydroxyl group of 25(OH)D3 [26]. It has been demonstrated that S408T has least effect on vitamin D 1 α hydroxylase activity among S408T, S408A, S408V, and S408I mutants; and S408I has the most detrimental effect [26]. The human T409I, which cause VDDR-I, has no 1 α -hydroxylase activity [27]. Human T321 is involved in oxygen activation [22]. T321S shows a 1 α -hydroxylase activity similar to wt. However, T321N, T321A, and T321R exhibit no activity [28] even though threonine (T) and asparagine (N) are both polar neutral

amino acids. Structurally, the threonine ($C_4H_9NO_3$, side group $-CH(CH_3)-OH$) is more resemble to S ($C_3N_7NO_3$, $-CH_2-OH$) than N ($C_4H_8N_2O_3$, $-CH_2-CONH_2$), which may explain no damaging effect for T321S substitution. Among 21 randomly selected mutations in *SRD5A2*, *PHEX*, and *CYP27A1*, G to S substitution is only present in *SRD5A2*^{G183S} and *G196S* (Table 1), which causes pseudovaginal perineoscrotal hypospadias (male pseudohermaphroditism) due to 5-alpha-reductase deficiency. G183 and G196 residues are highly conserved across different species. PSIPRED shows multiple protein secondary structure changes caused by G183S and G196S (data not shown). Therefore, the overall structural and functional context unique to each mutation should be considered to predict function.

PolyPhen mispredicted six mutations (Q65H, G102E, P112L, E189K, E189L, and T409I) and PolyPhen-2 missed two mutations (P112L, T409I). These amino acid residues are not conserved (sharing low amino acid sequence identity) among mitochondrial p450 enzymes or across different species but show remarkable conservation of their topology and their tertiary structures [29]. Q65 is in α -helix A' ($\alpha A'$) and T409 is in strand 3 of β -sheet 1 ($\beta 1-3$). Both of them are involved in substrate binding [26]. G102 and R107 are very close to strand 5 of β -sheet 1 ($\beta 1-5$). R107 has been described as a heme binding residue [30]. Homology modeling of human *CYP27B1* based on the crystal structure of rabbit *CYP2C5* suggests that R107H could disrupt protein folding [31]. G102E or G102D may disrupt heme incorporation or folding. P112 is located in the substrate recognition site 1 (SRS1) [26, 32] and is very close to α -helix B' ($\alpha B'$). E189 lies in the α -helix E ($\alpha E'$). The mutants P112L, E189K, and E189L would disrupt protein folding as well. Given that all these residues are not well conserved and their mutations result in either substrate

Fig. 3 Protein secondary structure prediction by PSIPRED. The predicted protein structural changes caused by G102S, G102E, and G102D are presented. The location of the changes is indicated by *asteisk*. The amino acids are indicated by the *single letter code*, and the location of the missense mutation is indicated by an *arrow*. C coil, H α -helix, E β -sheet



binding or folding defects, PolyPhen or PolyPhen-2 may have a higher rate of inaccuracy for prediction of non-conserved residues.

PSIPRED achieved the highest accuracy among three programs. The drawback for PSIPRED is that the protein structure change caused by a mutation is not highlighted by the program and one has to manually compare mutant structure prediction with that of wild-type. Subtle changes

may be missed. For example, E189K and E189L caused only one subtle change: 4 amino acid β -strand (residues 477–480) was shortened by one amino acid in E189K and 19 amino acid α -helix was increased by one amino acid in E189L (residues 456–474); T321R resulted in two subtle changes: N-terminal α -helix (residues 2–32) was shortened by one amino acid and C-terminal α -helix (residues 456–474) was increased by one amino acid. However, the

predicted subtle changes may not reflect the real change in protein structure or function and should be used only as a guide for prediction of damaging mutations.

Acknowledgments This study was funded by a grant from National Comprehensive Plan for Science & Technology.

Conflicts of interest All authors have nothing to disclose.

References

1. M.F. Holick, N. Engl. J. Med. **357**(3), 266 (2007)
2. W.L. Miller, A.A. Portale, Best Pract. Res. Clin. Endocrinol. Metab. **15**(1), 95 (2001)
3. R.P. Gupta, Y.A. He, K.S. Patrick, J.R. Halpert, N.H. Bell, J. Clin. Endocrinol. Metab. **90**(2), 1210 (2005)
4. J.B. Cheng, M.A. Levine, N.H. Bell, D.J. Mangelsdorf, D.W. Russell, Proc. Natl. Acad. Sci. USA **101**(20), 7711 (2004)
5. W.L. Miller, Endocrinology **146**(6), 2544 (2005). doi:[10.1210/en.2005-0096](https://doi.org/10.1210/en.2005-0096)
6. W.L. Miller, A.A. Portale, Endocr. Dev. **6**, 156 (2003)
7. G.K. Fu, D. Lin, M.Y. Zhang, D.D. Bikle, C.H. Shackleton, W.L. Miller, A.A. Portale, Mol. Endocrinol. **11**(13), 1961 (1997)
8. M. Labuda, K. Morgan, F.H. Glorieux, Am. J. Hum. Genet. **47**(1), 28 (1990)
9. R. St-Arnaud, S. Messerlian, J.M. Moir, J.L. Omdahl, F.H. Glorieux, J. Bone Miner. Res. **12**(10), 1552 (1997)
10. T. Monkawa, T. Yoshida, S. Wakino, T. Shinki, H. Anazawa, H.F. DeLuca, T. Suda, M. Hayashi, T. Saruta, Biochem. Biophys. Res. Commun. **239**(2), 527 (1997)
11. G.K. Fu, A.A. Portale, W.L. Miller, DNA Cell Biol. **16**(12), 1499 (1997)
12. S. Kitanaka, K. Takeyama, A. Murayama, T. Sato, K. Okumura, M. Nogami, Y. Hasegawa, H. Niimi, J. Yanagisawa, T. Tanaka, S. Kato, N. Engl. J. Med. **338**(10), 653 (1998)
13. A.S. Alzahrani, M. Zou, E.Y. Baitei, O.M. Alshaikh, R.A. Al-Rijjal, B.F. Meyer, Y. Shi, A novel G102E mutation of CYP27B1 in a large family with vitamin D-dependent rickets type 1. J. Clin. Endocrinol. Metab. **95**(9), 4176
14. A. Aiyar, Y. Xiang, J. Leis, Methods Mol. Biol. **57**, 177 (1996)
15. Y. Shi, M. Zou, K. Collison, E.Y. Baitei, Z. Al-Makhalafi, N.R. Farid, F.A. Al-Mohanna, J. Clin. Endocrinol. Metab. **91**(6), 2373 (2006)
16. S. Kitanaka, A. Murayama, T. Sakaki, K. Inouye, Y. Seino, S. Fukumoto, M. Shima, S. Yukizane, M. Takayanagi, H. Niimi, K. Takeyama, S. Kato, J. Clin. Endocrinol. Metab. **84**(11), 4111 (1999)
17. V. Ramensky, P. Bork, S. Sunyaev, Nucleic Acids Res. **30**(17), 3894 (2002)
18. S. Sunyaev, V. Ramensky, P. Bork, Trends Genet. **16**(5), 198 (2000)
19. I.A. Adzhubei, S. Schmidt, L. Peshkin, V.E. Ramensky, A. Gerasimova, P. Bork, A.S. Kondrashov, S.R. Sunyaev, Nat. Methods **7**(4), 248 (2010)
20. K. Bryson, L.J. McGuffin, R.L. Marsden, J.J. Ward, J.S. Sodhi, D.T. Jones, Protein structure prediction servers at University College London. Nucleic Acids Res. **33**(Web Server issue), W36 (2005)
21. D.T. Jones, J. Mol. Biol. **292**(2), 195 (1999)
22. W.L. Miller, A.A. Portale, Endocrinol. Metab. Clin. North Am. **28**(4), 825 (1999)
23. T. Shinki, H. Shimada, S. Wakino, H. Anazawa, M. Hayashi, T. Saruta, H.F. DeLuca, T. Suda, Proc. Natl. Acad. Sci. USA **94**(24), 12920 (1997)
24. K. Takeyama, S. Kitanaka, T. Sato, M. Kobori, J. Yanagisawa, S. Kato, Science **277**(5333), 1827 (1997)
25. C.J. Kim, L.E. Kaplan, F. Perwad, N. Huang, A. Sharma, Y. Choi, W.L. Miller, A.A. Portale, J. Clin. Endocrinol. Metab. **92**(8), 3177 (2007)
26. K. Yamamoto, E. Uchida, N. Urushino, T. Sakaki, N. Kagawa, N. Sawada, M. Kamakura, S. Kato, K. Inouye, S. Yamada, J. Biol. Chem. **280**(34), 30511 (2005)
27. J.T. Wang, C.J. Lin, S.M. Burridge, G.K. Fu, M. Labuda, A.A. Portale, W.L. Miller, Am. J. Hum. Genet. **63**(6), 1694 (1998)
28. N. Sawada, T. Sakaki, S. Kitanaka, S. Kato, K. Inouye, Eur. J. Biochem. **268**(24), 6607 (2001)
29. C.A. Hasemann, R.G. Kurumbail, S.S. Boddupalli, J.A. Peterson, J. Deisenhofer, Structure **3**(1), 41 (1995)
30. D.E. Prosser, G. Jones, Trends Biochem. Sci. **29**(12), 664 (2004)
31. K. Yamamoto, H. Masuno, N. Sawada, T. Sakaki, K. Inouye, M. Ishiguro, S. Yamada, J. Steroid Biochem. Mol. Biol. **89–90**(1–5), 167 (2004)
32. O. Gotoh, J. Biol. Chem. **267**(1), 83 (1992)
33. X. Wang, M.Y. Zhang, W.L. Miller, A.A. Portale, J. Clin. Endocrinol. Metab. **87**(6), 2424 (2002)
34. S.J. Smith, A.K. Rucka, J.L. Berry, M. Davies, S. Mylchreest, C.R. Paterson, D.A. Heath, M. Tassabehji, A.P. Read, A.P. Mee, E.B. Mawer, J. Bone Miner. Res. **14**(5), 730 (1999)
35. C. Hackel, L.E. Oliveira, L.F. Ferraz, M.M. Tonini, D.N. Silva, M.B. Toralles, E.G. Stuchi-Perez, G. Guerra-Junior, J. Mol. Med. **83**(7), 569 (2005)
36. F. Vilchis, J.P. Mendez, P. Canto, E. Lieberman, B. Chavez, Clin. Endocrinol. (Oxf) **52**(3), 383 (2000)
37. P.H. Dixon, P.T. Christie, C. Wooding, D. Trump, M. Grieff, I. Holm, J.M. Gertner, J. Schmidtke, B. Shah, N. Shaw, C. Smith, C. Tau, D. Schlessinger, M.P. Whyte, R.V. Thakker, J. Clin. Endocrinol. Metab. **83**(10), 3615 (1998)
38. G.N. Gallus, M.T. Dotti, A. Federico, Neurol. Sci. **27**(2), 143 (2006)

WEEK 7 READINGS

Basic Local Alignment Search Tool (BLAST)

By: Ingrid Lobo, Ph.D. (*Write Science Right*) © 2008 Nature Education

Citation: Lobo, I. (2008) Basic Local Alignment Search Tool (BLAST). *Nature Education* 1(1):215

Awash in a sea of data, how do scientists identify the function of a newly cloned gene? Online resources like the Basic Local Alignment Search Tool (BLAST) provide a helping hand.

Since the discovery of the genetic code, biological research has undergone a sea of change in the way it is performed. Until the early twentieth century, biology focused on the processes of living organisms and almost always involved experiments in laboratories and in the field. The growth of molecular biology during the twentieth century moved research into the test tube, where biological systems could be painstakingly dissected and reassembled. Then, beginning in the 1970s, scientists started accumulating DNA and protein sequence data at an exponential rate; in fact, researchers currently have approximately 97 billion bases sequenced and over 93 million records. Amazingly, this sequence data doubles every 18 months!

But how do investigators make sense of this massive amount of data? How can they identify the functions of newly cloned genes? And is it possible to estimate the evolutionary relationships between genes or proteins just by examining their nucleotide or amino acid sequences? To address these important issues, researchers must first tease out the relationships between different species that are descended from a common ancestor. Any sequence similarity can then be used to infer function and evolutionary relationships. In fact, one common method for examining and comparing genes is to search for similarities between newly sequenced DNA and databases of gene sequences that have already been described. By identifying related genes or gene families with known functions, scientists can infer the functions and evolutionary relationships of newly cloned genes or even whole genomes.

As gene and protein sequence databases grew at the end of the twentieth century, scientists turned to computers to help analyze this abundant and ever-growing amount of data. Today, one of the most common tools used to examine DNA and protein sequences is the Basic Local Alignment Search Tool, also known as BLAST (Altschul *et al.*, 1990). BLAST is a computer algorithm that is available for use online at the [National Center for Biotechnology Information \(NCBI\)](#) website, as well as many other sites. BLAST can rapidly align and compare a query DNA sequence with a database of sequences, which makes it a critical tool in ongoing genomic research. In fact, the initial paper describing the program, published in the *Journal of Molecular Biology* and entitled "[Basic Local Alignment Search Tool](#)," was the most highly cited publication of the 1990s (Taub, 2000). In recent years, the parallel development of large-scale sequencing projects and bioinformatic tools like BLAST has enabled scientists to study the genetic blueprint of life across many species, and it has also helped connect biology and computer science in the maturing field of bioinformatics.

Alignment Theory

Although the computer science principles behind BLAST have been around for some time, prior to BLAST, they had not been applied to biology. Before BLAST, alignment programs used dynamic programming algorithms, such as the Needleman-Wunsch and Smith-Waterman algorithms, that required long processing times and the use of supercomputers or parallel computer processors (Collins & Coulson, 1984; Gotoh &

Tagashira, 1986; Smith & Waterman, 1981).

Figure 1 depicts a Needleman-Wunsch alignment of the words "PELICAN" and "COELACANTH." The search space of the alignment is shown using a Cartesian grid and is proportional to the length of the sequences being compared plus one extra row and column (Figure 1A).

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

 Figure 1A: Initialization of the alignment matrix.

A Needleman-Wunsch alignment of the words \"PELICAN\" and \"COELACANTH.\"
Adapted with permission from Korf, I., Yandell, M. & Bedell, J. BLAST O'Reilly Media, Inc. 2003. All rights reserved.

Next, the alignment matrix is initialized with a zero in the upper left corner. For each letter of the word being aligned, a point is deducted so that each letter has a progressively more negative score. Why does the algorithm subtract a point? In an alignment, the diagonal is read from the upper left to the lower right, and when the analysis moves vertically or horizontally, it indicates a gap in the sequence. Thus, each time the program moves straight up or down, a gap penalty is applied that takes away points from the alignment score. Finally, a little arrow, or pointer, is added to indicate which direction to follow the alignment (Figure 1B). In the third stage, the algorithm starts to actually build and score the alignment in a step called fill, or induction.

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

 Figure 1B: Filling the axes of the alignment matrix.

When filling the axes of the alignment matrix, start in the upper left corner and set it to 0. Next, assign a score for each letter in the row or column. Note that there is a penalty for gaps, and that the arrow should point toward the origin of the alignment.

© 2003 O'Reilly Media Inc. All rights reserved.

In this example, the analysis begins by aligning the C to the P and calculating a score. In Figure 1C, one point is added if two letters match, and one point is subtracted if they do not. This calculation is carried out three times, once for each square to the left (dark blue), above (green), and to the upper left (brown). Using a value from either the upper square or the left square, the final score is -2 (-1 + -1). Using the 0 score in the upper left diagonal square, the final score is -1 (0 + -1). Because -1 is the highest score, this score is jotted down in the alignment matrix, and because the upper left square was the one leading to the best score, an arrow is inserted in the box pointing toward this square (light blue, Figure 1C).

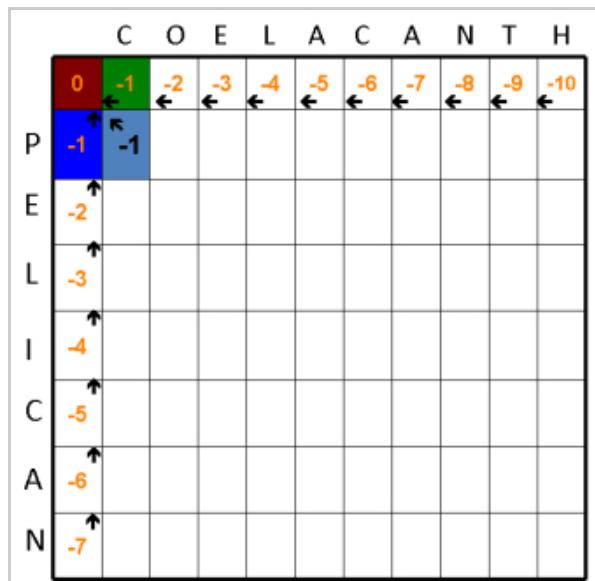


Figure 1C: Induction or filling of the alignment matrix, part I.

One point is added if two letters match, and one point is subtracted if they do not. Using a value from either the upper (green) or left (dark blue) square, the final score is -2; however, using the value from the upper left (brown) square, the final score is -1.

Because this is the highest score, it is recorded in the alignment matrix along with an arrow pointing to the upper left square.

© 2003 O'Reilly Media Inc. All rights reserved.

[Figure Detail](#)

This same process continues, calculating two scores for every square in the matrix (Figures 1D and 1E). At the end, there is a completely scored matrix with a series of arrows used to find the optimal alignment (Figure 1F).

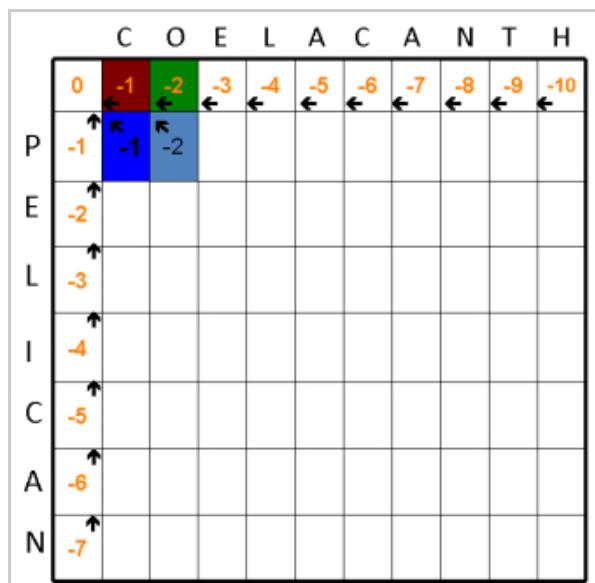


Figure 1D: Induction or filling in of the alignment matrix, part II.

The same process is carried out for the next square in the alignment. Here, using the value in upper left (brown) square yields a sum of -2, using the value in the upper (green) square yields a sum of -3, and using the value in the left (dark blue) square yields a sum of -2. Because -2 is the highest score and was initially calculated using the upper left square, -2 is recorded in the matrix along with an arrow pointing toward the brown square.

© 2003 O'Reilly Media Inc. All rights reserved.

[Figure Detail](#)

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

 Figure 1E: Induction or filling in of the alignment matrix, part III.

The rest of the matrix is completed using the same method.

© 2003 O'Reilly Media Inc. All rights reserved.

[Figure Detail](#)

The final steps of generating an alignment are called traceback, and they involve finding the optimal, highest-scoring alignment. The traceback starts in the lower right of the matrix (Figure 1F) and follows the pointers to the adjacent boxes. By definition, this will be the best scoring path through the alignment (Figure 1G).

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

 Figure 1F: Traceback of the optimal alignment.

The final steps of generating an alignment are called traceback, which involved finding the optimal, highest-scoring alignment.
 Adapted with permission from Korf, I., Yandell, M. & Bedell, J. BLAST O'Reilly Media, Inc. 2003. All rights reserved.

[Figure Detail](#)

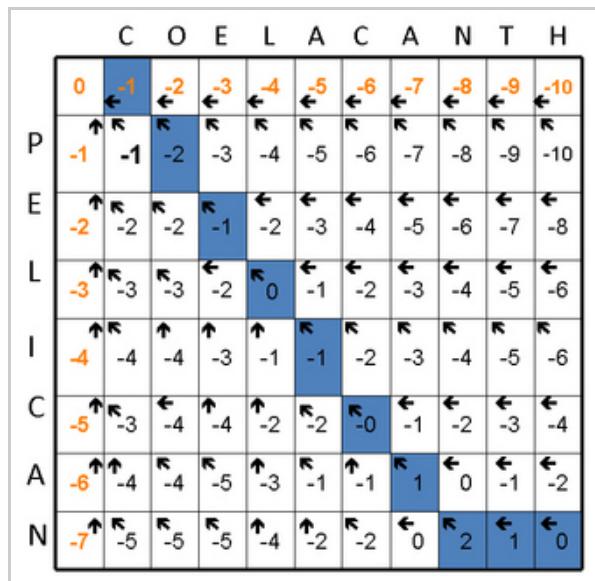


Figure 1G: Traceback of the optimal complete alignment.

During traceback, following the best scoring path reveals the best alignment.

Adapted with permission from Korf, I., Yandell, M. & Bedell, J. BLAST O'Reilly Media, Inc. 2003. All rights reserved.

[Figure Detail](#)

Although this sort of dynamic programming did a complete job of comparing every single residue of one sequence to every single residue of a second sequence and kept track of how well the sequences aligned at every step, these algorithms required a considerable amount of computer memory and processing time. Computing speed was an especially important concern, because these exhaustive programs had to search databases that continued to grow at exponential rates. Moreover, most regions of the search space did not score very well and therefore probably could have been skipped during the calculation process. Finally, these programs required powerful computing hardware that was expensive, rare, and ultimately impractical for most scientists and labs.

Researcher Stephen Altschul and colleagues wanted to bypass these challenges and develop a way for databases to be searched quickly on routinely used computers. In order to increase the speed of alignment, the BLAST algorithm was designed to approximate the results of an alignment algorithm created by Smith and Waterman (1981), but to do so without comparing each residue against every other residue (Altschul et

al., 1990). BLAST is therefore heuristic in nature, meaning it has "smart shortcuts" that allow it to run more quickly (Madden, 2005). However, in this trade-off for increased speed, the accuracy of the algorithm is slightly decreased.

The BLAST Heuristic

BLAST increases the speed of alignment by decreasing the search space or number of comparisons it makes. Specifically, instead of comparing every residue against each other, BLAST uses short "word" (w) segments to create alignment "seeds." BLAST is designed to create a word list from the query sequence with words of a specific length, as defined by the user (Figure 2). Requiring only three residues to match in order to seed an alignment means that fewer sequence regions need to be compared. Larger word sizes usually mean that there are even fewer regions to evaluate (e.g., Figure 3A compared to Figure 3B). Once an alignment is seeded, BLAST extends the alignment according to a threshold (T) that is set by the user. When performing a BLAST query, the computer extends words with a neighborhood score greater than T (Figure 3C). A cutoff score (S) is used to select alignments over the cutoff, which means the sequences share significant homologies. If a hit is detected, then the algorithm checks whether w is contained within a longer aligned segment pair that has a cutoff score greater than or equal to S (Altschul *et al.*, 1990). When an alignment score starts to decrease past a lower threshold score (X), the alignment is terminated (Figure 3C). These and many other variables can be adjusted to either increase the speed of the algorithm or emphasize its sensitivity.

Testing the BLAST Algorithm

Altschul and colleagues tested the BLAST algorithm on a database of randomly generated sequences, and they examined the output resulting from different w and T parameters. If T is set to be a lower threshold, then the algorithm detects more word pairs and requires a longer processing time (Altschul *et al.*, 1990). Thus, choosing the value for T was a major decision because the researchers wanted to reach a compromise between the algorithm's sensitivity and its processing time (e.g., Figure 3A compared to Figure 3B).

Next, Altschul and colleagues tested BLAST on a database of real sequences, and they found it was successful in quickly identifying alignments with high scores. In searching the globin gene family, for example, they found that BLAST identified 88 of the 89 globin alignments that scored above 80. Other gene families, including the immunoglobulins, protein kinases, and cytochrome c genes, were then examined to measure the number of alignments detected when using different T and S values. BLAST was also able to detect similar regions within pairs of long sequences. These tests therefore showed that BLAST was fast, sensitive, and accurate as a tool for analyzing sequence alignments (Altschul *et al.*, 1990).

Bringing Mathematical Rigor to Alignment

One of the most notable innovations of BLAST is that the program calculates the statistical significance for

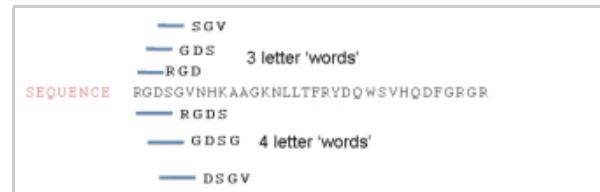


Figure 2: Generating word seeds.

Instead of comparing every residue against each other, BLAST uses short "word" (w) segments to create alignment "seeds." BLAST is designed to create a word list from the query sequence with words of a specific length, as defined by the user.

Adapted with permission from Korf, I., Yandell, M. & Bedell, J. BLAST O'Reilly Media, Inc. 2003. All rights reserved.

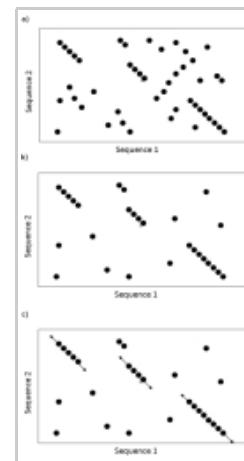


Figure 3
Figure Detail

each sequence alignment result. This is known as the expect value (E-value) or probability value (P-value), and it is calculated for each alignment. The E-value describes how many hits you can expect to see by chance when searching a database of a certain size, whereas the P-value describes the probability that the alignment you are observing is due to chance. In general, the lower the E- or P-value is, the more likely it is that an alignment is significant. Below the common 10^{-5} score, P and E are roughly equivalent (Madden, 2005).

The addition of statistical rigor to sequence alignment has been controversial. Some researchers rely too much on significance values to include or exclude sequences despite poorly chosen parameters, whereas others over-interpret "insignificant" results because the results "look" right. While all scientific results are subject to interpretation, BLAST scores and statistics bring much-needed objectivity to sequence comparisons, and the debate about them has helped improve methods for determining significance.

The BLAST Family

Since 1990, many variants of BLAST have been developed, each with specialized features. Early on, the original BLAST was split into two adaptations: NCBI BLAST and Washington University BLAST (WU BLAST). Both BLASTs have program variations. For instance, BLASTN can be used to compare a nucleotide sequence with a nucleotide database; BLASTP can be used to compare a protein sequence with a database of protein sequences; and BLASTX can take a nucleotide sequence, translate it, and query it versus a protein database in one step (Gish & States, 1993). TBLASTN compares a protein query sequence to all six possible reading frames of a database and is often used to identify proteins in new, undescribed genomes. Finally, TBLASTX compares all six reading frames of a query sequence to all six reading frames of a database—an intensive algorithmic feat that can bring even modern computers to a grinding halt if not used properly.

In addition, NCBI has some of its own specialized variants of BLAST. For example, MEGABLAST is a program that can rapidly complete searches for sequences with only minor variations and can more efficiently manage queries with longer sequences (Altschul *et al.*, 1994). PSI- and PHI- are other powerful BLAST tools that allow more complex and evolutionary divergent proteins to be aligned (Altschul *et al.*, 1997). These and other programs, as well as genomic BLAST databases, are all available on the [NCBI BLAST website](#).

Conclusion

Thus, since its creation, BLAST has become an essential tool for biologists. Its speed and sensitivity allow scientists to compare nucleotide and protein sequences to both single sequences and large databases. Most importantly, BLAST has helped democratize bioinformatics analysis and make it accessible to any researcher over the Internet; indeed, it is rare to read a modern molecular biology paper that does not refer to a BLAST alignment. In short, BLAST and its descendant applications have permitted scientists to predict the functions of genes and proteins in whole genomes, answering questions *in silico* that could never be answered at a lab bench or in the field.

References and Recommended Reading

Altschul, S. F., *et al.* Basic Local Alignment Search Tool. *Journal of Molecular Biology* **215**, 403–410 (1990)
doi:10.1016/S0022-2836(05)80360-2 ([link to article](#))

Altschul, S. F., *et al.* Issues in searching molecular sequence databases. *Nature Genetics* **6**, 119–129 (1994)
doi:10.1038/ng0294-119 ([link to article](#))

Altschul, S. F., *et al.* Gapped Blast and PSI-Blast: A new generation of protein database search programs.

Nucleic Acids Research 25, 3389–3402 (1997)

Collins, J. F., & Coulson, A. F. Applications of parallel processing algorithms for DNA sequence analysis. *Nucleic Acids Research* 12, 181–192 (1984)

Gish, W., & States, D. J. Identification of protein coding regions by database similarity search. *Nature Genetics* 3, 266–272 (1993) doi:10.1038/ng0393-266 ([link to article](#))

Gotoh, O., & Tagashira, Y. Sequence search on a supercomputer. *Nucleic Acids Research* 14, 57–64 (1986)

Korf, I., Yandell, M., & Bedell, J. *BLAST: An Essential Guide to the Basic Local Alignment Search Tool* (O'Reilly, Sebastopol, CA, 2003)

Madden, T. The BLAST sequence analysis tool. In *NCBI Handbook*, ed. J. McEntyre and J. Ostell (National Library of Medicine, Bethesda, MD, 2005)

Smith, T. F., & Waterman, M.S. Identification of common molecular subsequences. *Journal of Molecular Biology* 147, 195–197 (1981) doi:10.1016/0022-2836(81)90087-5

Taub, G. Sense from sequences: Stephen F. Altschul on bettering BLAST. *Science Watch* 11, 3–4 (2000)



Where did the BLOSUM62 alignment score matrix come from?

Sean R Eddy

Many sequence alignment programs use the BLOSUM62 score matrix to score pairs of aligned residues. Where did BLOSUM62 come from?

Back in the good old days, so many things were easier to understand. I once disassembled the engine of my 1972 MG just to see how it worked, but now I won't touch the squirrel's nest of technology that's inside my modern Honda Civic. Likewise, in the early days of sequence comparison, alignment scores were straightforward stuff that anybody could tweak. The first sequence comparisons just assigned -1 per mismatch and -1 per insertion/deletion, and if you didn't like that, you could make up whatever scores you thought gave you better-looking alignments. Those days are gone. Look inside a modern amino acid score matrix, and you'll see a squirrel's nest of 400 numbers. These highly tuned matrices, which go by industrialized acronyms like BLOSUM62 and PAM250, no longer seem to have any user serviceable parts inside. Blame probability theory.

Alignment scores are log-odds scores

What we want to know is whether two sequences are homologous (evolutionarily related) or not, so we want an alignment score that reflects that. Theory says that if you want to compare two hypotheses, a good score is a log-odds score: the logarithm of the ratio of the likelihoods of your two hypotheses. If we assume that each aligned residue pair is statistically independent of the others (biologically dubious, but mathematically convenient), the align-

ment score is the sum of individual log-odds scores for each aligned residue pair. Those individual scores make up a 20×20 score matrix. The equation for calculating a score $s(a,b)$ for aligning two residues a and b is:

$$s(a,b) = \frac{1}{\lambda} \log \frac{p_{ab}}{f_a f_b}$$

The numerator (p_{ab}) is the likelihood of the hypothesis we want to test: that these two residues are correlated because they're

The definition of 'conservative substitution' in a score matrix is purely statistical. It has nothing directly to do with amino acid structure or biochemistry.

chance ($p_{ab} > f_a f_b$), then the odds ratio is greater than one and the score is positive. Operationally, we say that positive scores mean conservative substitutions, and negative scores indicate nonconservative substitutions. This definition of 'conservative substitution' in a score matrix is purely statistical. It has nothing directly to do with amino acid structure or biochemistry.

This explains some details in BLOSUM62 that may seem counterintuitive at first glance. For instance, tryptophan (W/W) pairs score +11, while leucine (L/L) pairs only score +4; why shouldn't all identities get the same score? The rarer the amino acid is, the more surprising it would be to see two of them align together by chance. In the homologous alignment data that BLOSUM62 was trained on, leucine/leucine (L/L) pairs were in fact more common than tryptophan/trypophan (W/W) pairs ($p_{LL} = 0.0371$, $p_{WW} = 0.0065$), but tryptophan is a much rarer amino acid ($f_w = 0.099$, $f_L = 0.013$). Run those numbers (with BLOSUM62's original $\lambda = 0.347$) and you get +3.8 for L/L and +10.5 for W/W, which were rounded to +4 and +11.

Another example is that BLOSUM62 awards a +1 to an apparently nonconservative alignment of a positively charged glutamic acid, but a seemingly more innocuous alignment of an alanine to a leucine gets penalized -1. A/L pairs are indeed slightly more frequent in homologous alignments than K/E pairs ($p_{AL} = 0.0044$, $p_{KE} = 0.0041$ in the BLOSUM62 training data), but A and L are more common amino acids ($p_A = 0.074$, $p_L = 0.099$, $p_K = 0.058$, $p_E = 0.054$). With $\lambda = 0.347$, this gives a score of -1.47 for A/L (rounded to -1) and 0.76 for K/E (rounded to +1).

Sean R. Eddy is at Howard Hughes Medical Institute & Department of Genetics, Washington University School of Medicine, 4444 Forest Park Blvd., Box 8510, Saint Louis, Missouri 63108, USA.
e-mail: eddy@genetics.wustl.edu

Where did those numbers come from?

So much for the scores. But we've just pushed the question to a different level. Where did we get the target frequencies p_{ab} ?

The target frequencies are the probability we expect to see a,b aligned in homologous alignments. Thus, the basic idea is to take lot of known, trusted pairwise alignments similar to what we expect our next alignment to look like, and count the frequency at which each residue pair occurs.

The more information we have about the two sequences we're aligning, the better we'll be able to estimate what their target frequencies should be. For example, if we know that we're aligning the sequences of two integral membrane proteins, our target frequencies would be biased toward hydrophobicity. There are endless ways of slicing sequence alignment databases and estimating new score matrices specialized for certain organisms or certain types of sequences. A cottage industry of bioinformatics toils in this happy realm. For a general purpose matrix like BLOSUM62, though, we can't really use sequence- or species-specific sources of information. One source of information remains crucial: evolutionary distance. The target frequencies depend very strongly on the evolutionary distance between the two sequences. If the two sequences diverged recently, the target frequencies should be peaked on identical residues. The more divergent the relationship we're looking for, the flatter the target frequencies need to be. All modern amino acid score matrices are therefore estimated from frequencies observed in trusted alignment data, using some procedure to make a series of related matrices that are appropriate for different expected divergences.

The procedure that Steve and Jorja Henikoff used to estimate the BLOSUM matrices was straightforward¹. The Henikoffs took a big database of trusted alignments (their BLOCKS database), and (in effect) only counted pairwise sequence alignments related by less than some threshold percentage identity. A threshold of 62% identity or less resulted in the target frequencies for the BLOSUM62 matrix. An 80% threshold gave the more highly conserved target frequencies of the BLOSUM80 matrix, and a 45% threshold gave the more divergent BLOSUM45 matrix. Empirically, the BLOSUM matrices have performed very well. BLOSUM62 has become a *de facto* standard for many protein alignment programs.

DIY score matrices

We can even make up the values if we state some assumptions, which is especially practical for smaller, simpler 4×4 DNA score matrices. Say we want to make a DNA scoring matrix optimized for finding 88% identity alignments. Let's assume that all mismatches are equiprobable, and the composition of both alignments and background sequences is uniform at 25% for each nucleotide. Then, our values are 0.22 for the four identities and 0.01 for each of the 12 types of mismatch, and our background frequencies $f_a, f_b = 0.25$ for all a, b .

What's the difference between making up our target frequencies and calculating scores, versus just making up scores?

Plug those into the log-odds equation, and we get (if $\lambda = 1$) +1.26 for a match and -1.83 for a mismatch. Scale up a bit with $\lambda = 0.25$ and round off, and *voilà*, we have a new scoring system of +4/-7.

What's the difference between making up our target frequencies and calculating scores, versus just making up scores? When we make up our p_{ab} values, we're directly describing what we expect homologous alignments to look like (here, simply 88% identity), and the resulting score matrix is optimal for detecting alignments that match our target frequencies. If instead, we make up an arbitrary score matrix, we're blindly looking for a scheme that works well.

Even arbitrary scores imply target alignment frequencies

Remarkably, even if we do make up arbitrary scores, they still imply target frequencies. It's useful to know what these implicit target frequencies are, so we know what sort of alignments the score matrix will optimally detect. The proof that arbitrary scores still imply optimal target frequencies is subtle (an important statistical result from Sam Karlin and Steve Altschul^{2,3}), but the arithmetic is straightforward.

Rearrangement of the log-odds equation gives us $p_{ab} = f_a f_b e^{\lambda s_{ab}}$; the problem is the unknown λ . The sum of all the p_{ab} values must be 1, by definition, because they're probabilities. So, set

$$\sum_{a,b} f_a f_b e^{\lambda s_{ab}} = 1$$

and solve for a nonzero λ . Such a λ exists so long as the score matrix has two key properties: it must have at least one positive score, and the expected score for random sequence alignments must be negative. Most score matrices have these properties because the same properties are necessary to make local sequence alignment algorithms like BLAST and Smith/Waterman work³. (Both conditions are met by definition for matrices derived as log-odds scores, except for the useless case of $p_{ab} = f_a f_b$ for all a, b .)

For instance, both FASTA and WU-BLASTN use an arbitrary +5/-4 scoring system for matches/mismatches in DNA alignments, whereas NCBI BLASTN uses a +1/-2 scoring system. Is there a big difference? Probably hard to tell just from looking at those scores. If you run the calculation, you find that these two scoring systems are almost polar opposites. NCBI BLASTN's +2/-1 system is optimal for detecting homologous DNA alignments that are 95% identical—almost perfect matches. FASTA and WU-BLASTN's +5/-4 system is optimal for detecting homologous DNA alignments that are only 65% identical—at the edge of the ‘twilight zone’ for gapped alignment methods’ ability to recognize homologous DNA alignments.

Note: Supplementary information is available on the Nature Biotechnology website.

1. Henikoff, J.G. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA* **89**, 10915–10919 (1992).
2. Karlin, S. & Altschul, S.F. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc. Natl. Acad. Sci. USA* **87**, 2264–2268 (1990).
3. Altschul, S.F. Amino acid substitution matrices from an information theoretic perspective. *J. Mol. Biol.* **219**, 555–565 (1991).

Further study

You can download an ANSI C program for calculating the implicit target frequencies p_{ab} of a score matrix (see **Supplementary Notes**). The BLOSUM62 score matrix and its background frequencies are included as an example. The code also contains two basic methods of solving for roots of equations like the one for λ : the bisection method, and the Newton/Raphson method.

**nature
biotechnology**

Wondering how some other mathematical technique really works?
Send suggestions for future primers to
askthegeek@natureny.com.



BLAST Homepage and Selected Search Pages

Introducing the BLAST homepage and form elements/functions of selected search pages

<http://blast.ncbi.nlm.nih.gov>

National Center for Biotechnology Information • National Library of Medicine • National Institutes of Health • Department of Health and Human Services

Background

BLAST [1] is a suite of programs provided by NCBI for aligning query sequences against those present in a selected target database. The NCBI BLAST homepage (<http://blast.ncbi.nlm.nih.gov>) provides an access point for these tools to perform sequence alignment on the web.

The BLAST Homepage

The BLAST homepage consists of several sections, each provides a specific set of functions:

1. The common header (A), present in most BLAST-related pages, provides easy access to other content or functions not directly accessible from the homepage.
2. The right hand column (B) provides links to a list of recently completed results, BLAST-related news and search tips.
3. Pages with web forms for submitting searches are listed as links in the body of the BLAST homepage. These links are organized into three categories, “BLAST Assembled RefSeq Genomes” (C), “Basic BLAST” (D), and “Specialized BLAST” (E).
4. The search box (F) in the “BLAST Assembled Genomes” section takes the name of an organism as input and suggests a list of candidates. Selecting from the suggested list will locate the best genomic sequence dataset for BLAST alignment purposes.

The screenshot shows the NCBI BLAST homepage with various sections highlighted by yellow arrows labeled A through F:

- A:** The common header at the top left, containing the NCBI logo, the word "BLAST", and tabs for "Home", "Recent Results", "Saved Strategies", and "Help".
- B:** The right-hand sidebar titled "Your Recent Results" which lists recent search results.
- C:** The "BLAST Assembled Genomes" section, which contains a search bar for entering an organism name and a list of suggested organisms.
- D:** The "Basic BLAST" section, which lists four main search types: nucleotide blast, protein blast, blastx, and tblastn.
- E:** The "Specialized BLAST" section, which lists various specialized search options like Primer-BLAST, MOLE-BLAST, and IgBLAST.
- F:** The search box within the "BLAST Assembled Genomes" section, used for suggesting organisms.

The common BLAST header

The common BLAST header provides a convenient way to navigate among different pages to access different contents or functions.

- The NCBI logo (G) links to the NCBI homepage (<http://www.ncbi.nlm.nih.gov>) to allow access to non-BLAST related functions and content.
- The “Home” tab (H) links to the BLAST homepage.
- BLAST search results are temporarily saved for 36 hours. The “Recent Results” tab (I) links to a page that keeps track of recently submitted search requests that have not expired. The Request ID uniquely assigned to a submitted search provides a one-click access to that result.
- The “Saved Strategies” tab (J) lists a set of search setups saved earlier. It allows the examination of search settings used, quick re-launch of these searches, and download of specific strategies for sharing or re-use in standalone BLAST.
- The “Help” tab (K) points to page with a list of links to help documents, tutorials, references and useful download directories on the BLAST ftp site.
- My NCBI [2] is a free account from NCBI, which allows users to customize their site preference and manage their works performed on the NCBI site. Login-related links for My NCBI (L) are at the right. BLAST searches performed while logged in to a My NCBI account enables access to BLAST search results for their full 36-hour life span through the “Recent Results” tab. Strategies saved will be saved permanently.

The “Recent Results” page

BLAST search results are available for 36 hours. The “Recent Results” tab displays a list of recently submitted search requests that have not expired. The list is session-specific and will be lost if session cookie is cleared upon browser exit. For this reason, it is recommended that BLAST searches be done with an active login to a My NCBI account such as the one shown below. The My NCBI login from the header is shown as an insert at the upper right (**A**). Each result is given as a row in the table. The identifier in the Request ID (RID) column (**B**) provides an one-click access to the search result. The program, Title and Database column (**C**) combine to provide a summary for a specific search. Database restriction applied are indicated by “more ...” and the popup upon mouseover (**D**). The “save”, “download” and the red x (**E**) allow saving the search strategy, downloading the search strategy, and removing the search from the list.

NCBI/BLAST/Recent Results
Links to your unexpired BLAST jobs appear below. [more...](#)

A My NCBI
Welcome ttao_phd. [\[Sign Out\]](#)

B Your Recent Results
(Click headers to sort columns)

Submitted at	Request ID	Status	Program	Title	Qlength	Database	Expires at				
08-11 12:23	0FEKPWZ1014	Running	d.megablast	segment of chromosome 1	50001	wgs	more...	08-13 00:23	save	download	×
08-11 11:49	0FCKVM77015	Done	blastn	(3) - emb F12345 (327 letters)	327	wgs	more...	08-12 23:49	save	download	×
08-11 00:23	0E4E3Y9C014	Done	megablast	(2) - emb F12345 (327 letters)	327	Genome (all assembled scaffolds)	Home sapiens <small>Included:
Vicugna pacos (taxid:30538)
</small>	08-12 12:23	save	download	×
08-11 00:22	0E4BKB2X014	Done	blastp	dbj BAG37749.1 (122 letters)	122	nr		08-12 12:22	save	download	×
08-10 23:12	0E089W23015	Done	blastn	emb F12345 (327 letters)	327	refseq_rna		08-12 11:12	save	download	×

F Request ID: Go
C Submitted at Request ID Status Program Title Qlength Database Expires at
D **E**

NCBI/BLAST/Format Request
Request ID [View report](#) Show results in a new window

G

H **I** **J** **K** **L** **M** **N**

Format
Show Alignment as HTML Old View Reset form to defaults
Alignment View Pairwise
Display Graphical Overview NCBI-gi CDS feature
Masking Character: Lower Case Color: Grey
Limit results Descriptions: 100 Graphical overview: 100 Line length: 60
Pairwise
Pairwise
Pairwise with dots for identities
Query-anchored with dots for identities
Query-anchored with letters for identities
Flat query-anchored with dots for identities
Flat query-anchored with letters for identities
Organism Type common name, binomial, taxid, or group name. Only 20 top taxa will be shown.
Enter organism name or id—completions will be suggested Exclude +
Entrez query:
Expect Min: Expect Max:
Percent Identity Min: Percent Identity Max:
Format for PSI-BLAST with inclusion threshold:

The input box (**F**) above the table is for retrieving other results using their assigned RIDs, such as those shared among colleagues, used as teaching or demonstration examples, or those with issues encountered and reported to NCBI's blast-help group.

Clicking the “Go” button without an RID loads the format form (**G**) where the content (**H**) and format (**I**) of results can be adjusted.

The “Saved Strategies” page

The “Saved Strategies” tab (shown below) displays a list of search strategies. The first four columns (**J**) provide a good summary of the search settings for each saved entry. The “view” link (**K**) loads the settings in a search page, while the “download” link (**L**) saves the settings in an ASN.1 formatted file for use with standalone BLAST or reloading on the web services using the “Choose File” and “View” button (**M**). Clicking the red X (**N**) removes the entry from the list.

NCBI/BLAST/Saved Strategies
Browse and run your saved sets of BLAST search parameters. (Query sequences larger than 10K are not saved.) [more...](#)

M Upload Search Strategy
Upload file : No file chosen

J Your Saved Strategies
(Click headers to sort columns)

Program	Created	Title	Database	view	download	×
blastp	2013-08-11 00:33:50	dbj BAG37749.1 (122 letters)	nr	view	download	×
megablast	2013-08-11 00:26:41	(2) - emb F12345 (327 letters)	GPIPE/9606/current/all_contig	view	download	×
blastn	2012-07-11 13:28:04	AP Biology Fossil Specimen Gene 3	nr	view	download	×
blastp	2010-04-09 16:01:46	(2) - NP_055023:dentin sialophosphoprotein preproprotein...	nr	view	download	×

K **L** **N**

Functions of BLAST search pages

There are five BLAST search pages, each performs a specific type of sequence alignment. These pages are the foundation for the NCBI BLAST service and will be described in more detail. Table 1 below summarizes key aspects of pages. These pages access a set of common databases, a summary of the contents for these databases are given in Table 2.

Table 1. Key features of the BLAST search pages in the “Basic BLAST” category

Search page	Query & database combination	Alignment type	Programs & functions (default program in bold)
nucleotide blast	nucleotide vs nucleotide	nucleotide vs nucleotide	<u>megablast</u> : for sequence identification, intra-species comparison <u>discontiguous megablast</u> : for cross-species comparison, searching with coding sequences <u>blastn</u> : for searching with shorter queries, cross-species comparison
protein blast	Protein vs protein	protein vs protein	<u>blastp</u> : general sequence identification and similarity searches <u>DELTA-BLAST [2]</u> : protein similarity search with higher sensitivity than blastp <u>PSI-BLAST</u> : iterative search for position-specific score matrix (PSSM) construction or identification of distant relatives for a protein family <u>PHI-BLAST</u> : protein alignment with input pattern as anchor/constraint
blastx	nucleotide (translated) vs protein	protein vs protein	<u>blastx</u> : for identifying potential protein products encoded by a nucleotide query
tblastn	protein vs nucleotide (translated)	protein vs protein	<u>tblastn</u> : for identifying database sequences encoding proteins similar to the query
tblastx	nucleotide (translated) vs nucleotide (translated)	protein vs protein	<u>tblastx</u> : for identifying nucleotide sequences similar to the query based on their coding potential

Table 2. Contents of the common BLAST sequence databases

Database	Type	Content
nr (nt) default	Nucleotide	All GenBank + EMBL + DDBJ + PDB sequences, excluding sequences from PAT, EST, STS, GSS, WGS, TSA and phase 0, 1 or 2 HTGS sequences, mostly non-redundant.
refseq_rna	Nucleotide	Curated (NM_, NR_) plus predicted (XM_, XR_) sequences from NCBI Reference Sequence Project.
refseq_genomic	Nucleotide	Genomic sequences from NCBI Reference Sequence Project.
chromosome	Nucleotide	Complete genomes and complete chromosomes from the NCBI Reference Sequence project.
Human G+T	Nucleotide	The genomic sequences plus curated and predicted RNAs from the current build of the human genome.
Mouse G+T	Nucleotide	The genomic sequences plus curated and predicted RNAs from the current build of the mouse genome.
est	Nucleotide	Database of GenBank + EMBL + DDBJ sequences from EST division
HTGS	Nucleotide	Unfinished High Throughput Genomic Sequences; Sequences: phases 0, 1 and 2
wgs	Nucleotide	Assemblies of Whole Genome Shotgun sequences.
pat	Nucleotide	Nucleotides from the Patent division of GenBank.
pdb	Nucleotide	Nucleotide sequences from the 3-dimensional structure records from Protein Data Bank.
alu_repeats	Nucleotide	Selected Alu repeats from REPBASE, suitable for identifying Alu repeats from query sequences. See "Alu alert" by Claverie and Makalowski, Nature 371: 752 (1994).
TSA	Nucleotide	Transcriptome Shotgun Assemblies, assembled from RNA-seq SRA data
16S microbial	Nucleotide	16S Microbial rRNA sequences from Targeted Loci Project

nr default	Protein	Non-redundant GenBank CDS translations + RefSeq + PDB + SwissProt + PIR + PRF, excluding those in PAT, TSA, and env_nr.
refseq_protein	Protein	Protein sequences from NCBI Reference Sequence project.
swissprot	Protein	Last major release of the UniProtKB/SWISS-PROT protein sequence database (no incremental updates).
pat	Protein	Proteins from the Patent division of GenBank.
pdb	Protein	Protein sequences from the 3-dimensional structure records from the Protein Data Bank.
env_nr	Protein	Protein sequences translated from the CDS annotation of metagenomic nucleotide sequences.
tsa_nr	Protein	Protein sequences translated from CDSs annotated on transcriptome shotgun assemblies.

Elements of the Standard Nucleotide BLAST search page

The “nucleotide-blast” link loads the “Standard Nucleotide BLAST” search page. The top of the page (below the common BLAST header) contains the breadcrumb indicating the page position in the site hierarchy (**A**), the page title, a set of tabs for quick navigation among the five core BLAST search pages (**B**), plus links to set the page back to default and to bookmark a search page with customized settings (**C**). The default display of the page contains three sections with the functions described below.

Enter Query Sequence

The main input box (**D**) takes nucleotide query sequences in various formats [1]. For a single query, “Query subrange” boxes (**E**) define a segment of the query to be used in the search. Query sequences saved in a plain text file can be uploaded using the “Choose File” button (**F**). The “Align two or more sequences” checkbox (**G**) changes the “Choose Search Set” sections below to “Enter Subject Sequence” to allow comparison between the query and those in the subject input box (**H**).

NCBI/ BLAST/ blastn suite Standard Nucleotide BLAST

blastn blastp blastx tblastn tblastx

A Standard Nucleotide BLAST

B

BLASTN programs search nucleotide databases using a nucleotide query. [more...](#)

C

Enter Query Sequence

Enter accession number(s), gi(s), or FASTA sequence(s)

From _____ To _____

Or, upload file Choose File **F** chosen

Title **G**

Enter a descriptive title for your BLAST search

Align two or more sequences **H**

From _____ To _____

Enter Subject Sequence

Enter accession number, gi, or FASTA sequence

Clear

Subject subrange

From _____ To _____

Or, upload file Choose File No file chosen

Choose Search Set

BLAST database can be selected from the standard list using the pull-down menu (**I**). A search can be restricted to a subset of entries in the selected database using the “Organism” field by typing the name of the species, strains, or taxonomic group in the text-box and selecting from the suggested list (**J**). Checking the exclusion box to the right excludes sequences from the selected organism from the search.

Multiple organisms can be selected by adding extra input box using the “+” button. Specific types of relatively low value sequences can be excluded using the checkboxes below (**K**). For certain databases, entering custom queries in the “Entrez Query” textbox (**L**) will restrict a search to entries satisfying the specified criteria. For example, entering “biomol_mrna[prop] AND 500:1000[slen]” will restrict a search to mRNA entries 500 to 1000 bases long.

Choose Search Set

Database Human genomic + transcript Mouse genomic + transcript Others (nr etc.):
Nucleotide collection (nr/nt)

Organism Optional Enter organism name or id—completions will be suggested Exclude

Exclude Optional Models (XM/XP) **J** Excluded/environmental sample sequences

Limit to Optional Sequences from type material

Entrez Query Optional mouse

Nucleotide collection (nr/nt)
Genomic plus Transcript
Human genomic plus transcript (Human G+T)
Mouse genomic plus transcript (Mouse G+T)
Other Databases
Nucleotide collection (nr/nt)
Reference RNA sequences (refseq_rna)
Reference genomic sequences (refseq_genomic)
NCBI Genomes (chromosome)
Expressed sequence tags (est)
Genomic survey sequences (gss)
High throughput genomic sequences (HTGS)
Patent sequences(pat)
Protein Data Bank (pdb)
Human ALU repeat elements (alu_repeats)
Sequence tagged sites (dbsts)
Whole-genome shotgun contigs (wgs)
Transcriptome Shotgun Assembly (TSA)
16S ribosomal RNA sequences (Bacteria and Archaea)

Program Selection

Three programs (**M** and Table 1) with different speed and sensitivity are available for nucleotide vs nucleotide sequence alignment. The default megablast is better for certain tasks, such as identifying the input query

Program Selection

Optimize for

Highly similar sequences (megablast) **M**

More dissimilar sequences (discontiguous megablast)

Somewhat similar sequences (blastn)

Choose a BLAST algorithm

N

BLAST

Search database Nucleotide collection (nr/nt) using Megablast (Optimize for highly similar sequences)

Show results in a new window

+Algorithm parameters **O**

and searching with large genomic query; discontiguous megablast works better in finding related sequences from other organisms; while blastn works better for short input queries and identifying short matches, it also works better for cross-species searches than megablast. Clicking the “BLAST” button (**N**) submits the search to BLAST server for processing. Results will be automatically displayed when completed.

“Algorithm parameters” link (**O**) opens a normally collapsed section that allows access to other parameter settings, including adjustment of number of alignments saved, search stringencies, scoring systems (score matrix) and gap penalties, as well as query filtering. Detailed descriptions are given next.

Elements of the Standard Nucleotide BLAST search page (cont.)

General Parameters

Parameters in this section specify the search sensitivity. The “Max target sequences” (A) sets the maximum database matches BLAST saves for a query. The checked “Short queries” checkbox (B) allows BLAST to automatically optimize settings for short input queries 50 nucleotides or less. The “Expect threshold” (C) filters out matches that are less significant with Expect value above the setting. The “Word size” (D) set the size of the initial seed match phase, smaller settings are more sensitive. The “Max matches in a query range” (E) limits the matches saved to a given region of the query (such as from repeats) so matches to other region of the query are not crowded out. The default setting of “0” means no limit.

Scoring Parameters

Parameters in this section specify the search sensitivity. The “Match/Mismatch Scores” (F) specifies the reward assigned to exact match and penalty assigned to a mismatch. The “Gap Costs” (G) field specifies how gaps introduced in the alignment should be penalized. For megablast, the default is linear, no penalty for opening a gap, while extending a gap assumes a linear penalty proportional to the length of the gap. For both parameters, non-default settings can be selected using the pull-down menu.

Filters and Masking

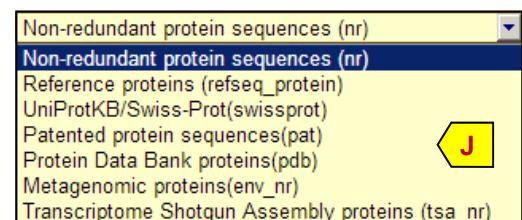
Parameters in the section specify whether low complexity sequences and organism-specific repeats should be filtered (H) and whether to filter only at the initial match stage (Mask for lookup table only) or during alignment extension as well(I). Lower case letters in the query (provided as a mixed upper and lower case letters in FASTA, representing custom features) can also be masked.

Elements of the Standard Protein BLAST search page

The “protein-blast” link in the “Basic BLAST” links to the “Standard Protein BLAST” search page. The top of this page has the same tab and links found in the “Standard Nucleotide BLAST” search page (pg. 4) that provide the same functions. The default display contains three sections with the functions described below.

Enter Query Sequence

Refer to the description for “Standard Nucleotide BLAST” (pg.4) for details. In addition, checking the “Align two or more sequences” will change the “Program Selection” section to leave blastp as the only choice.



Choose Search Set

Most of the components are similar to the “Standard Nucleotide BLAST” page (pg.3). The main difference is that the database pull-down menu contains a smaller list of protein databases (J).

Program Selection

Four different programs (K and Table 1) are available to satisfy various search needs. The default blastp is a general purpose protein alignment program for identifying a sequence or finding others similar to it. PSI-BLAST is for finding more distant relatives and for PSSM construction. PHI-BLAST does protein alignment with a pattern in the query as a constraint. DELTA-BLAST is a more sensitive search using conserved domain matches the query to build a PSSM for the match evaluation.

More complex searches may require adjustment of other search settings listed under the “Algorithm parameters” link (L). Parameters specific to protein blast are described next.

Elements of the Standard Protein BLAST search page (cont.)

The “Algorithm parameters” portion of the “Standard Protein BLAST” search page is organized in a similar manner to that for the “Standard Nucleotide BLAST.”

General Parameters

This section is the same as that in the “Standard Nucleotide BLAST” (pg. 4).

Scoring Parameters

Eight score matrices from two families are available (**A**). The default BLOSUM62 matrix is the best general purpose matrix. For short queries, PAM30 is often selected. Each matrix has its own set of supported gap penalties under the Gap Costs pull-down menu (**B**).

Scores for protein alignment can be adjusted to account for the bias in sequence composition using various approaches as indicated by the “Compositional adjustments” setting. Other options including “no adjustment” can be selected using the pull-down menu (**C**).

Filters and Masking

Parameters in the section specify whether low complexity and whether to filter only at the seed lookup stage (**D**). Lower case letters in the query (provided as a mixed upper and lower case letters in FASTA, representing custom features) can also be masked (**E**). These settings are not needed when compositional adjustments are used.

Items unique to translated search pages

The page layout for translated BLAST search pages is the same as “Standard Protein BLAST.” However, they do contain a few program-specific parameters.

Translated BLAST: blastx

In the “Enter Query Sequence” section, a “Genetic code” field (**F**) is present under the “Choose File” button specify the codon table used in the translation of the input nucleotide query. Choose a code appropriate for the source of the query sequence. The remaining sections are the same as the “Standard Protein BLAST” page.

Translated BLAST: tblastn

The page layout is the same as the “Standard Protein BLAST” search page. The key difference is that the Database field lists available nucleotide databases instead.

Translated BLAST: tblastx

The layout differences are the presence of the “Genetic code” field (**F**), which is also present in the blastx page, and the databases listed under the Database pull-down menu. In addition, the main input box in the “Enter Query Sequence” takes a nucleotide query.

Other search pages

BLAST search pages under the “BLAST Assembled RefSeq Genomes” category differ from those under the “Basic BLAST” category only in the databases they access. The link names clearly indicate the source organism(s) for the database sequences against which the query will be searched.

The “Specialized BLAST” category contains different types of search pages. Those using the core BLAST programs and the same general layout described about are summarized in Table 3.

Table 3. Function of Specialized BLAST pages following the standard layout

Page name	Searching against specialized databases
Search trace archives	unannotated nucleotide capillary sequencing trace reads from various organisms
Search sequences that have gene expression profiles (GEO)	nucleotide sequences with expression information
Align two (or more) sequences using BLAST (bl2seq)	direct comparison of two groups of sequences. This is the “Basic BLAST” page with “Align two or more sequences” checked
Search protein or nucleotide targets in PubChem BioAssay,	nucleotide or protein sequences with associated chemical activity assay data from PubChem
Search SRA by experiment	raw nucleotide reads from the next generation sequencing technology
Search RefSeqGene	nucleotide sequences representing selected loci from the RefSeqGene project

Other search pages (cont.)

Other search pages in the “Specialized BLAST” category have non-standard page layouts. These services use BLAST or other alignment programs in combination with other tools to accomplish specialized tasks. Table 4 provides a functional summary of these pages.

Table 4. Function of Specialized BLAST pages not following the standard layout

Page link name	Functions
Make specific primers with Primer-BLAST [4, 5]	Designing primers using the primer3 algorithm and checking their template specificity using BLAST
Search immunoglobulins and T cell receptor sequences (IgBLAST) [6]	Searching immunoglobin or T cell receptor sequences against germline databases for annotation of the input immunoglobulin sequences
Screen sequence for vector contamination (vecscreenc)	Screening input nucleotide sequences against a library of known vector and other artificial sequences to identify contaminations
Find conserved domains in your sequence (cds) [7]	Searching an protein sequence against a database of curated domains for functional analysis. This search is performed for all protein-blast requests.
Find sequences with similar conserved domain architecture (cdart)	Identifying conserved domains present in the input protein sequence followed by finding other sequences containing these identified domains
Constraint Based Protein Multiple Alignment Tool [8]	A protein multiple sequence alignment tool from NCBI. COBALT search can also be launched from any protein BLAST search results.
Needleman-Wunsch Global Sequence Alignment Tool	NCBI implementation of the Needleman-Wunsch global pair-wise alignment tool for nucleotide or protein queries
Cluster multiple sequences together with their database neighbors (MOLE-BLAST)	This tool takes multiple related input sequences, finds their nearest neighbors from the database using BLAST, and then cluster these sequences according to their sequence similarities.

Other ways to access NCBI web BLAST services

In addition to access through a web browser, BLAST web services described above, with the exception for those listed in Table 4, can also be accessed using alternative venues. These venues include the “-remote” option in different standalone BLAST+ programs, the RESTful BLAST service (known as Qblast or BLAST URLAPI), and BLAST SOAP services. The features of these venues are summarized in Table 5 below.

Table 5. Features of available methods to access NCBI web BLAST services

Venue	Features
Web browser	Intuitive: graphical user interfaces and result presentation Convenience: ease of searching with single or small batch of query sequences Speed: fast turnaround from the distributed computing system Versatility: available option enables searching against custom sequences Job Limitation: Not meant for high throughput searches with 1 hour CPU time limit Data partition: Access to different database requires different search pages
Standalone BLAST+ (-remote option) [9]	Comprehensive: more options available than on the Web for customizing and fine tuning the search Batch processing: search with large query sequences by submitting them in smaller batches automatically Less manual intervention: option for saving output in various formats Workflow incorporation: input and output can be integrated in custom workflow Extra requirements: installing standalone BLAST+ package and configuring it properly
RESTful BLAST (Qblast, BLAST URLAPI) [10]	Comprehensive: more available options to customize and fine tune the search than the Web Batch processing: search with large query sequences possible through batching Workflow incorporation: input and output can be integrated in custom workflow Extra requirements: efficient usage requires scripting/programming for requesting URL construction and result checking
blastn_vdb & tblastn_vdb Program similar to their counterparts from standalone blast+ that are included in the SRAToolkit [11]	Similar to their BLAST+ counterparts, but access SRR, WGS, and TSA files stored in vcb format. Comprehensive: more options available than on the Web for customizing and fine tuning the search Built-in client function: automatically downloads the data file to blast if not prefetched Less manual intervention: option for saving output in various formats Workflow incorporation: input and output can be integrated in custom workflow Extra requirements: installing SRAToolkit and configuring it properly

Technical assistance

NCBI provides technical assistance to the BLAST user community through its blast-help group. Problem and bug reports, suggestions and feature requests, as well as other questions related to BLAST usage should be addressed to the group (blast-help@ncbi.nlm.nih.gov). Submitting detailed information along with the problem report will help expedite the investigation.

Information needed when reporting problems encountered during web BLAST searches are:

- A description on the goal of the search
- The RID of the search
- The detailed error message
- The search page and settings used along with a summary of the input query, particularly if the RID was not issued

CPU related errors are caused when searches exceed the processing time limit. Repeat the search using “Edit and re-submit” link to get back to the search page with the following adjustments will help resolve the issue:

- Reduce the number or size of the input query sequence(s), use subsequence for large single query if possible
- Add database limit using Organism or Entrez query box to search a focused subset
- Increase the search stringency by using
 - ◊ A lower Expect value
 - ◊ A larger Word size
 - ◊ Filters and repeat masking
 - ◊ A lower number for Maximum target sequences

For errors occurred from using “-remote” option of the standalone BLAST+ package, as well as standalone BLAST+ package for local searches, the following pieces of information should be provided:

- A description on the goal of the search
- The platform and version of the installed BLAST+ package
- The complete error message
- The complete command line used
- A summary and a small sample of the input query file
- BLAST server returned RIDs if available

For RESTful BLAST and SOAP BLAST, the following pieces of information should be provided:

- A description on the goal of the search
- The platform and relevant code used to call the service
- The complete error message
- A summary and small sample of the input query file

References

1. NCBI BLAST: a better web interface. NCBI BLAST: a better web interface. Johnson M et. al. Nucleic Acids Res. 2008 Jul 1;36(Web Server issue):W5-9.
2. My NCBI Help. <http://www.ncbi.nlm.nih.gov/books/NBK3843/>
3. Domain enhanced lookup time accelerated BLAST. Boratyn GM, et. al. Biol Direct. 2012 Apr 17;7:12. doi: 10.1186/1745-6150-7-12.
4. Primer-BLAST: a tool to design target-specific primers for polymerase chain reaction. Ye J, et. al. BMC Bioinformatics. 2012 Jun 18;13:134. doi: 10.1186/1471-2105-13-134.
5. Factsheet: Primer-BLAST. ftp://ftp.ncbi.nih.gov/pub/factsheets/HowTo_PrimerBLAST.pdf
6. IgBLAST: an immunoglobulin variable domain sequence analysis tool. Ye J, et. al. Nucleic Acids Res. 2013 Jul 1;41 (Web Server issue):W34-40.
7. CDD: specific functional annotation with the Conserved Domain Database. Marchler-Bauer A, et. al. Nucleic Acids Res. 2009 Jan;37(Database issue):D205-10.
8. COBALT: constraint-based alignment tool for multiple protein sequences. Papadopoulos JS, Agarwala R. Bioinformatics. 2007 May 1;23(9):1073-9.
9. BLAST® Help manual. <http://www.ncbi.nlm.nih.gov/books/NBK1762/>.
10. Qblast's URL API User Guide. <http://www.ncbi.nlm.nih.gov/blast/Doc/urlapi.html>
11. Using SRA Toolkit to BLAST SRA Data Locally: ftp.ncbi.nlm.nih.gov/pub/factsheets/HowTo_Local_SRA_BLAST.pdf
12. Entrez Programming Utilities Help. <http://www.ncbi.nlm.nih.gov/books/NBK25501>.
13. Blastdbinfo: API access to a database of BLAST databases. NCBIINSIGHT blog entry <http://ncbiinsights.ncbi.nlm.nih.gov/2013/03/19/blastdbinfo-api-access-to-a-database-of-blast-databases/>



What is a hidden Markov model?

Sean R Eddy

Statistical models called hidden Markov models are a recurring theme in computational biology. What are hidden Markov models, and why are they so useful for so many different problems?

Often, biological sequence analysis is just a matter of putting the right label on each residue. In gene identification, we want to label nucleotides as exons, introns, or intergenic sequence. In sequence alignment, we want to associate residues in a query sequence with homologous residues in a target database sequence. We can always write an *ad hoc* program for any given problem, but the same frustrating issues will always recur. One is that we want to incorporate heterogeneous sources of information. A genefinder, for instance, ought to combine splice-site consensus, codon bias, exon/intron length preferences and open reading frame analysis into one scoring system. How should these parameters be set? How should different kinds of information be weighted? A second issue is to interpret results probabilistically. Finding a best scoring answer is one thing, but what does the score mean, and how confident are we that the best scoring answer is correct? A third issue is extensibility. The moment we perfect our *ad hoc* genefinder, we wish we had also modeled translational initiation consensus, alternative splicing and a polyadenylation signal. Too often, piling more reality onto a fragile *ad hoc* program makes it collapse under its own weight.

Hidden Markov models (HMMs) are a formal foundation for making probabilistic models of linear sequence ‘labeling’ problems^{1,2}. They provide a conceptual toolkit for building complex models just by draw-

Sean R. Eddy is at Howard Hughes Medical Institute & Department of Genetics, Washington University School of Medicine, 4444 Forest Park Blvd., Box 8510, Saint Louis, Missouri 63108, USA.
e-mail: eddy@genetics.wustl.edu

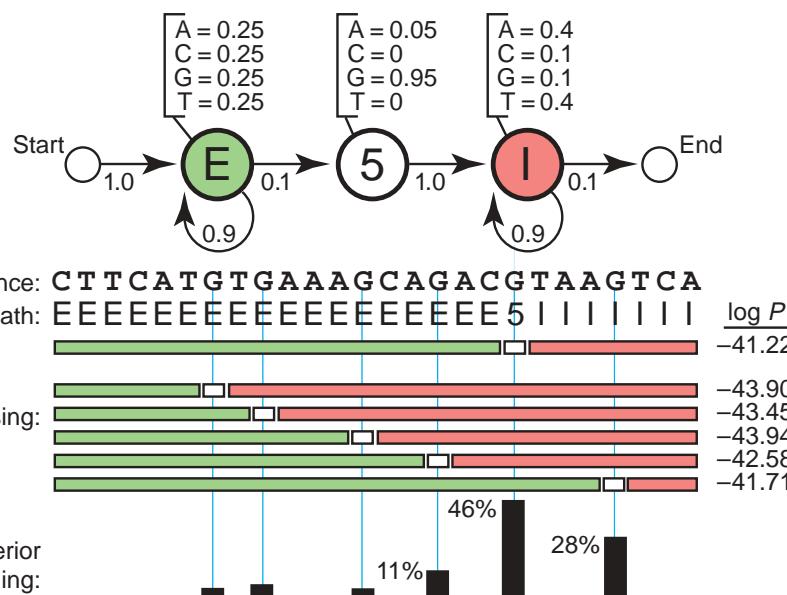


Figure 1 A toy HMM for 5' splice site recognition. See text for explanation.

ing an intuitive picture. They are at the heart of a diverse range of programs, including genefinding, profile searches, multiple sequence alignment and regulatory site identification. HMMs are the Legos of computational sequence analysis.

A toy HMM: 5' splice site recognition

As a simple example, imagine the following caricature of a 5' splice-site recognition problem. Assume we are given a DNA sequence that begins in an exon, contains one 5' splice site and ends in an intron. The problem is to identify where the switch from exon to intron occurred—where the 5' splice site (5'SS) is.

For us to guess intelligently, the sequences of exons, splice sites and introns must have

different statistical properties. Let's imagine some simple differences: say that exons have a uniform base composition on average (25% each base), introns are A/T rich (say, 40% each for A/T, 10% each for C/G), and the 5'SS consensus nucleotide is almost always a G (say, 95% G and 5% A).

Starting from this information, we can draw an HMM (Fig. 1). The HMM invokes three *states*, one for each of the three labels we might assign to a nucleotide: E (exon), 5' (5'SS) and I (intron). Each state has its own *emission probabilities* (shown above the states), which model the base composition of exons, introns and the consensus G at the 5'SS. Each state also has *transition probabilities* (arrows), the probabilities of moving from this state to a new state. The transition

probabilities describe the linear order in which we expect the states to occur: one or more Es, one 5, one or more Is.

So, what's hidden?

It's useful to imagine an HMM generating a sequence. When we visit a state, we emit a residue from the state's emission probability distribution. Then, we choose which state to visit next according to the state's transition probability distribution. The model thus generates two strings of information. One is the underlying *state path* (the labels), as we transition from state to state. The other is the *observed sequence* (the DNA), each residue being emitted from one state in the state path.

The state path is a Markov chain, meaning that what state we go to next depends only on what state we're in. Since we're only given the observed sequence, this underlying state path is hidden—these are the residue labels that we'd like to infer. The state path is a *hidden Markov chain*.

The probability $P(S, \pi | \text{HMM}, \theta)$ that an HMM with parameters θ generates a state path π and an observed sequence S is the product of all the emission probabilities and transition probabilities that were used. For example, consider the 26-nucleotide sequence and state path in the middle of Figure 1, where there are 27 transitions and 26 emissions to tote up. Multiply all 53 probabilities together (and take the log, since these are small numbers) and you'll calculate $\log P(S, \pi | \text{HMM}, \theta) = -41.22$.

An HMM is a *full probabilistic model*—the model parameters and the overall sequence 'scores' are all probabilities. Therefore, we can use Bayesian probability theory to manipulate these numbers in standard, powerful ways, including optimizing parameters and interpreting the significance of scores.

Finding the best state path

In an analysis problem, we're given a sequence, and we want to infer the hidden state path. There are potentially many state paths that could generate the same sequence. We want to find the one with the highest probability.

For example, if we were given the HMM and the 26-nucleotide sequence in Figure 1, there are 14 possible paths that have non-zero probability, since the 5'SS must fall on one of 14 internal As or Gs. Figure 1 enumerates the six highest-scoring paths (those

with G at the 5'SS). The best one has a log probability of -41.22, which infers that the most likely 5'SS position is at the fifth G.

For most problems, there are so many possible state sequences that we could not afford to enumerate them. The efficient Viterbi algorithm is guaranteed to find the most probable state path given a sequence and an HMM. The Viterbi algorithm is a dynamic programming algorithm quite similar to those used for standard sequence alignment.

Beyond best scoring alignments

Figure 1 shows that one alternative state path differs only slightly in score from putting the 5'SS at the fifth G (log probabilities of -41.71 versus -41.22). How confident are we that the fifth G is the right choice?

This is an example of an advantage of probabilistic modeling: we can calculate our confidence directly. The probability that residue i was emitted by state k is the sum of the probabilities of all the state paths that use state k to generate residue i (that is, $\pi_i = k$ in the state path π), normalized by the sum over all possible state paths. In our toy model, this is just one state path in the numerator and a sum over 14 state paths in the denominator. We get a probability of 46% that the best-scoring fifth G is correct and 28% that the sixth G position is correct (Fig. 1, bottom). This is called *posterior decoding*. For larger problems, posterior decoding uses two dynamic programming algorithms called Forward and Backward, which are essentially like Viterbi, but they sum over possible paths instead of choosing the best.

Making more realistic models

Making an HMM means specifying four things: (i) the symbol alphabet, K different symbols (e.g., ACGT, $K = 4$); (ii) the number of states in the model, M ; (iii) emission probabilities $e_i(x)$ for each state i , that sum to one over K symbols x , $\sum_x e_i(x) = 1$; and (iv) transition probabilities $t_i(j)$ for each state i going to any other state j (including itself) that sum to one over the M states j , $\sum_j t_i(j) = 1$. Any model that has these properties is an HMM.

This means that one can make a new HMM just by drawing a picture corresponding to the problem at hand, like Figure 1. This graphical simplicity lets one focus clearly on the biological definition of a problem.

For example, in our toy splice-site model, maybe we're not happy with our discrimination power; maybe we want to add a more realistic six-nucleotide consensus GTRAGT at the 5' splice site. We can put a row of six HMM states in place of '5' state, to model a six-base ungapped consensus motif, parameterizing the emission probabilities on known 5' splice sites. And maybe we want to model a complete intron, including a 3' splice site; we just add a row of states for the 3'SS consensus, and add a 3' exon state to let the observed sequence end in an exon instead of an intron. Then maybe we want to build a complete gene model...whatever we add, it's just a matter of drawing what we want.

The catch

HMMs don't deal well with correlations between residues, because they assume that each residue depends only on one underlying state. An example where HMMs are usually inappropriate is RNA secondary structure analysis. Conserved RNA base pairs induce long-range pairwise correlations; one position might be any residue, but the base-paired partner must be complementary. An HMM state path has no way of 'remembering' what a distant state generated.

Sometimes, one can bend the rules of HMMs without breaking the algorithms. For instance, in genefinding, one wants to emit a correlated triplet codon instead of three independent residues; HMM algorithms can readily be extended to triplet-emitting states. However, the basic HMM toolkit can only be stretched so far. Beyond HMMs, there are more powerful (though less efficient) classes of probabilistic models for sequence analysis.

1. Rabiner, L.R. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* **77**, 257–286 (1989).
2. Durbin, R., Eddy, S.R., Krogh, A. & Mitchison, G.J. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids* (Cambridge University Press, Cambridge UK, 1998).

nature biotechnology

Wondering how some other mathematical technique really works?
Send suggestions for future primers to
askthegeek@natureny.com.

WEEK 8 READINGS

Jalview Reference

Version 2.0 www.jalview.org

File Tools Help Window

Load Alignments and Projects

Input Alignment

Alignment File Menu

Save in a different format or export as an image, web page or lineart diagram. Import a newick tree with common sequence IDs.

Save As

Edit Menu

Undo and Redo.
 Copy, cut and paste sequences.
 Remove groups, gaps, residues and sequence homologs.

Sequence ID display

Displays sequence name, start and end positions.
 Right-click for sequence ID menu.
 Left-click: select sequences (shaded grey).
 Up and down arrow keys: reorder selection.
 Left-click+drag: change display width.
 Double left-click: view uniprot record.

Alignment Annotation label area

Right-click on row label to get annotation row menu.
 Left-click+drag to change height of area.

Amino acid property conservation

Measurement of the conservation of physicochemical properties in a column.

Alignment Quality

BLOSUM62 score based on observed substitutions.

Residue Consensus

Gives the commonest residues and their percentage for each column of the alignment.

View Menu

Options for alignment and annotation rendering that affect display and print/export.
 Controls display of overview window and uniprot annotation.

Alignment Color Menu

Define and change residue colour scheme for alignment and defined regions.
 Apply residue conservation shading to residue colour.

Font...

- Wrap
- Show Full Sequence ID
- Boxes
- Text
- Colour Text
- Show Gaps
- Show Annotations
- Sequence Features
- Overview Window...

Calculate Menu

Functions for reordering of sequences in the alignment. See over for calculating trees, spatial clusterings, alignments and predictions locally or via the web.

Sort

The Alignment Window

Its title gives the name, original format and subsequent analysis steps for the alignment.

Alignment position bar

Click here to toggle position markers, and drag to select ranges of columns in the alignment.

Alignment display

Click and drag to define a region selection. Control+C copies selection to clipboard. Control+A to select all alignment. Escape to clear the current selection. Shift+left-click and drag: insert and delete gaps at pointer position.

Region Selection

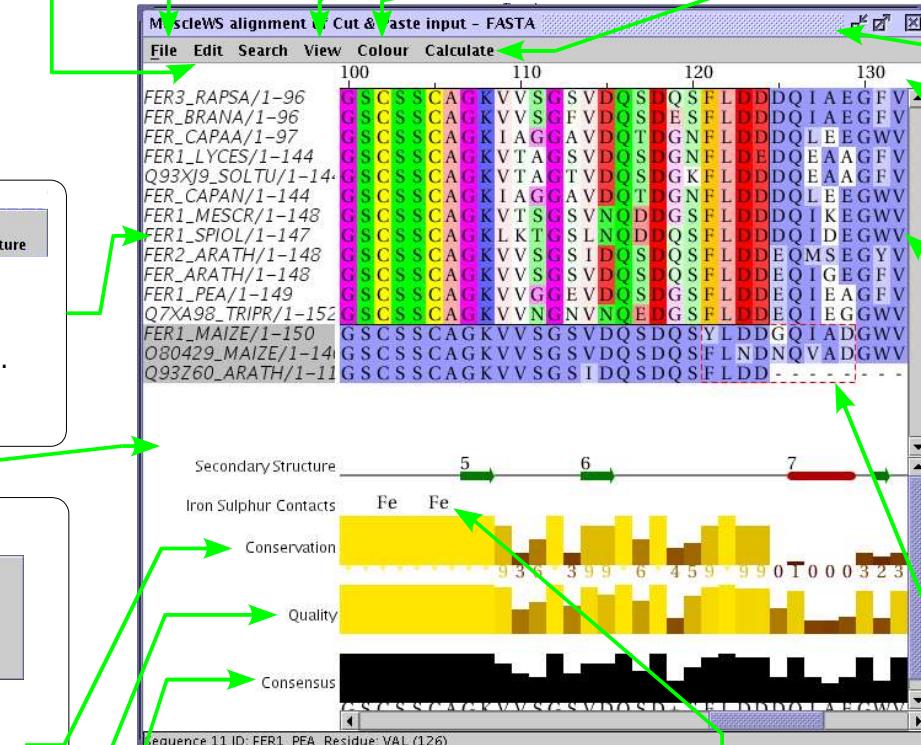
Right-click to get region menu to define a colour, view style, and name for selected area.

Define
 BLOSUM62
 Percentage Identity
 Zappo
 Taylor
 Hydrophobicity
 Helix propensity
 Strand propensity
 Turn propensity
 Buried Index
 Nucleotide
 User Defined
 Above % Identity
 Conservation

Residue information bar

Gives sequence number, residue name and position in sequence for nearest residue to mouse pointer.

Helix
 Sheet
 Label
 Colour
 Remove Annotation



User Defined Annotation

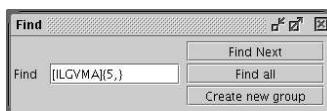
Rows are first added through the annotation row menu.
 Left-click: select position for annotation.
 Ctrl+left-click: add to selection.
 Right-click: menu to add secondary structure and/or label.

Control or shift+click+drag: insert/delete gaps in selected sequences at pointer position. Use tree or web service calculation menu to analyse region.

File Tools Help Window

Find

Enter regular expressions to search sequences and their IDs.



Regular Expression Element	Effect
.	Matches any single character
[]	Matches any one of the characters in the brackets
^	Matches at the start of an ID or sequence
\$	Matches at the end of an ID or sequence
*	Matches if the preceding element matches zero or more times
?	Matches if the preceding element matched once or not at all
+	Matches if the preceding element matched at least once
{count}	Matches if the preceding element matches a specified number of times
{min,}	Matches if the preceding element matched at least the specified number of times
{min,max}	Matches if the preceding element matches min or at most max number of times

Where to get help

About Documentation

Web services Menu

Submit sequences and regions for alignment and realignment, or make a secondary structure prediction.

Clustal Alignment... Muscle Alignment... JNet... Clustal Realign...

Sort Menu

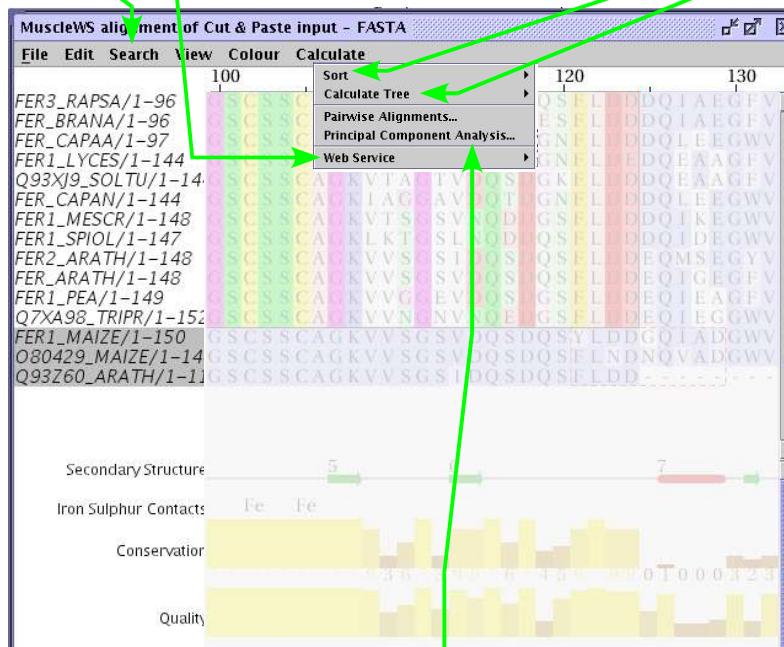
Options for reordering sequences.

by ID by Group by Pairwise Identity By Tree Order

Tree Menu

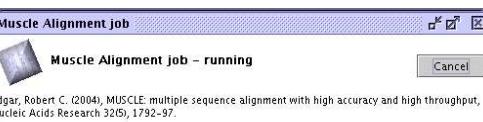
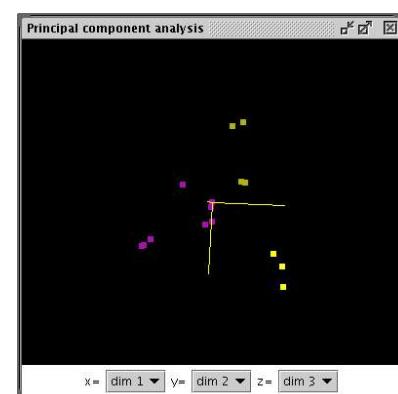
Create trees for the whole alignment or just a selected region.

Average Distance Using % Identity... Neighbour Joining Using % Identity... Average Distance Using BLOSUM62... Neighbour Joining using BLOSUM62...



Principal Component Analysis Viewer

Visualizes sequence clusters as a cloud of points in 3D. Move the mouse over a point to identify a sequence. Click and drag to change the view. Use the x,y and z menus to change the PCA dimensions.



Web Service Dialog

Gives the name, method reference, current status and log information for a webservice calculation. The Cancel button stops the current job permanently.

Tree Viewer

For browsing calculated or imported trees. Click on a node to swap the branch order. Click anywhere else to partition the tree and define sequence groups.

Tree File Menu

Print or save your tree as an an image, postscript or newick tree file.

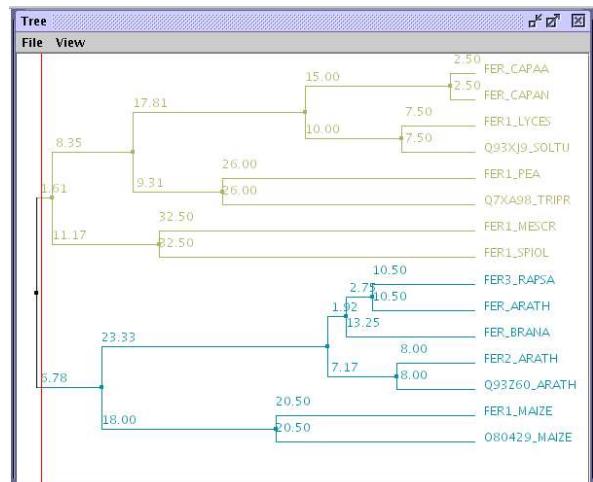
Save as
Output to Textbox...
Print

Newick Format
EPS
PNG

Tree View Menu

Change the appearance and scaling of the tree view.

Fit To Window
 Font Size - 12
 Show Distances
 Show Bootstrap Values
 Mark Unlinked Leaves



Comparative Analysis of Multiple Protein-Sequence Alignment Methods

Marcella A. McClure, Taha K. Vasi, and Walter M. Fitch

Department of Ecology and Evolutionary Biology, University of California, Irvine

We have analyzed a total of 12 different global and local multiple protein-sequence alignment methods. The purpose of this study is to evaluate each method's ability to correctly identify the ordered series of motifs found among all members of a given protein family. Four phylogenetically distributed sets of sequences from the hemoglobin, kinase, aspartic acid protease, and ribonuclease H protein families were used to test the methods. The performance of all 12 methods was affected by (1) the number of sequences in the test sets, (2) the degree of similarity among the sequences, and (3) the number of indels required to produce a multiple alignment. Global methods generally performed better than local methods in the detection of motif patterns.

Introduction

Comparison of primary sequence information is rapidly becoming the major source of data in the elucidation of the molecular mechanisms of replication and evolution of all organisms. There are basically three levels in the analysis of primary sequence information: (1) the search for homologues, (2) the multiple alignment of homologues, and (3) the phylogenetic reconstruction of the evolutionary history of homologues.

Many multiple sequence alignment programs and various scoring schemes have been developed to analyze potential relationships among sequences. Although a review (Myers 1991) and a comparison (Chan et al. 1992) of some methods from a computational perspective are available, there are no studies to date that evaluate these methods from a biologically informed perspective. The purpose of this study is to evaluate the ability of existing software to correctly identify the ordered series of motifs that are conserved throughout a given protein family.

There are two biological approaches to the multiple alignment of protein sequences: one attempts to align homologous (ancestrally related) features, while the other attempts to align functionally or spatially equivalent features of a protein family. While there is considerable overlap in the alignments produced by methods with these two goals, the intents are distinctly different.

Key words: sequence comparison, multiple alignment, protein family motifs.

Present address and address for correspondence and reprints:
Marcella A. McClure, Department of Biological Sciences, University of Nevada, Las Vegas, Nevada 89154-4004.

Mol. Biol. Evol. 11(4):571–592, 1994.

© 1994 by The University of Chicago. All rights reserved.
0737-4038/94/1104-0002\$02.00

Multiple alignment methods are often used without knowledge of the assumptions implicit in their operation. We will assess the major academically produced methods available, regardless of their intent, and indicate the assumptions implicit in each of the methods (table 1). Our basic premise is that, regardless of the final goal, a method that cannot find the functional motifs that are highly conserved throughout a given protein family has diminished value for detecting new biologically informative patterns.

The multiple protein-sequence alignment problem may be divided into the following two conceptual steps: (1) the initial inference of an ordered series of motifs defining the limits of a protein family and (2) detection of the ordered series of motifs in other proteins, thereby expanding the family. Many software packages, both academic and commercial, rely on the existence of previously defined protein families to provide the motifs of the family. How are such protein-family patterns initially determined? Among highly conserved sequences (>50% identity) it is very difficult to deduce which residues of a protein are necessary for function or structure, on the basis of multiple alignment of protein sequences alone. Laboratory experiments can provide clues as to which residues are critical for function and structure, but few generalizations can be made from such studies. Among distantly related proteins (<30% identical residues), however, conserved residues often indicate the essentially invariable regions of the protein that are necessary for function or structure. When multiple alignments of such data are derived, however, it soon becomes apparent that the currently available methods are not very satisfactory. Even with the utilization of the most sophisti-

Table 1
Multiple Alignment Methods

http://mbe.oxfordjournals.org
by guest on September 26, 2016

Method (Developer)	Algorithm	Matrix ^a	Indels	Limits ^b	Assumptions ^c	Features ^d	Data Type ^e
Global:							
AMULT (G. Barton)	NW	Any	C		Y, S	R, SE	P
ASSEMBLE (M. Vingron)	Dot matrix NW	Log odds	I+E		Y, S		P
CLUSTAL V (D. Higgins)	WL	Any	I+E			I	P, N
DFALIGN (D.-F. Feng)	NW	Log odds	C	UP	Y, E, O		P
GENALIGN ^f (H. Martinez)	CW, NW	UM	I+E			SE	P, N
MSA (S. Altschul)	CL	PAM250	I+E	ROS		B, FA	P
MULTAL (W. Taylor)	NW	UM, PAM250	C			AP, FA	P
MWT (J. Kececioglu)	maximum weight trace	Any	C	ROS			P
TULLA (S. Subbiah)	NW	Any	RGW	10 sequences		R, SE	P
Local:							
MACAW (G. Schuler)	SW	PAM250		DOS	Y	SE, FA, MD	P
PIMA (P. Smith)	SW	AACH	I+E		Y	MD	P
PRALIGN (M. Waterman)	CW	PAM250	I+E ^g		Y	MD, MC	P, N ^h

^a The matrices are log odds and PAM250 (Dayhoff et al. 1978); UM = unitary matrix (Feng et al. 1985); and AACH = amino acid cluster hierarchy (Smith and Smith 1990).

^b UP = unpublished parameters; ROS = easily runs out of computer space, thereby limited to six sequences; and DOS = runs only on a DOS system with Windows.

^c Y or N = yes or no to the question Has homology been established?; S or E = multiple alignment is of structural or evolutionary intent; and O = input sequences must be in nearest-neighbor order, and a program is provided for this purpose.

^d R = user-specified no. of iterations for refinement; SE = statistical evaluation is provided; I = interactive mode so that user may choose intermediate alignments; FA = specified region can be forced to align; B = correction for bias of overrepresentation of sequences; AP = alteration of parameters between iterations; MD = user-specified motif density; and MC = user-specified degree of motif conservation.

^e P = protein; and N = nucleic acid.

^f Licensed to IntelliGenetics.

^g This indel penalty applies to CWs only.

^h A separate program is available for nucleic acids.

cated software developed to date, refinement of such relationships still relies on the visual pattern-recognition skills of the human operator. The initial inference of the motifs defining a protein family by primary sequence analysis, therefore, requires the combination of multiple alignment methods and human pattern-recognition skills with corroborating experimental evidence (e.g., site-directed mutagenesis and crystallography).

We have tested both global and local multiple alignment methods for their ability to identify the ordered series of motifs that are conserved throughout the hemoglobin, kinase, ribonuclease H (RH), and aspartic acid protease protein families. The study presented here, while not exhaustive, indicates that all the methods analyzed suffer, to varying degrees, from three types of problems: (1) the inability to produce a single multiple alignment from correctly aligned subsets of the input sequences, (2) sensitivity to the number of sequences in the test, and (3) sensitivity to which specific sequences are in the test. The ramifications of these shortcomings for the identification of functional motifs, as well as phylogenetic reconstruction, are discussed.

Methods Used for Comparative Analysis of Alignment Programs

All analyses were conducted on a SPARCstation GS running SUN OS 4.1.1. The test sequences were extracted from the nonredundant database composed of PIR version 34.0, SWISS-PROT version 23.0, and GenPept (translated GenBank version 73.0) developed by the National Center for Biotechnology Information, National Library of Medicine (W. Gish, personal communication).

Scoring for Motifs

In general, we define a motif as a conserved contiguous run of 3–9 residues often involved in the function or structural integrity of a protein, as inferred by multiple alignment analysis or laboratory experiments. In some cases only remnants of a motif can be found, and we call this a semiconserved motif (e.g., see fig. 3, motif II). Occasionally a single residue, which is completely conserved among all members of a protein family, is found between larger motifs. In such cases we consider the single residue as one of the motifs comprising the ordered series of motifs (e.g., see fig. 4, motif II). An ordered series of motifs is defined as a set of conserved or semiconserved motifs that are found in the same arrangement relative to one another in all the sequences of a protein family. The spacing between the motifs can be highly variable, reflecting the regions of a protein that are less restricted by functional or structural constraints. These regions may evolve more rapidly and be more

subject to insertion, deletion, and duplication. There are two features of motifs that must be considered in their evaluation. The first, the motif density, is the percentage of the sequences in which a given motif is present. The second, the motif conservation, is the degree to which a motif is conserved in various members of the family (i.e., are the residues identical, or has conservative replacement occurred? have insertions and deletions [indels] occurred? or can more than one set of residues define a motif?). The motif conservation can be expressed in a variety of ways. In the PRALIGN program, e.g., the user specifies the number of mismatches and indels allowed within the motifs as two separate parameters.

Initially we planned to develop an independent scoring scheme to measure the global "goodness" of the alignments produced by the global methods. It soon became apparent, however, that some of the methods could not even identify the motifs known to be involved in the function of a given protein family. We decided, therefore, to score for each method's ability to detect each motif in four different data sets. A score for each motif is the percentage of the number of sequences in each data set for which the motif is correctly identified (see figs. 1–4; correct motifs are indicated by blackened bars and roman numerals). Some methods could find one or more correct motifs in more than one subset of the sequences without being able to align these motifs to one another to produce a single multiple alignment of the all the input sequences. In these cases the total percent correct match is a combined score of the aligned subsets (tables 2–5), allowing full credit for motif identification in each subset as if the motifs were each aligned correctly throughout the set. This scheme allows us to compare local and global methods to one another as well as among themselves.

Test Data Sets

We have chosen four protein families as data sets to test the ability of the multiple alignment methods to reconstruct known biologically informative patterns. To date, standard sets of protein sequences have not been established for assessing multiple alignment methods. The hemoglobin family has often been used to illustrate the reconstructive ability of a new multiple alignment method. In light of the extensive hemoglobin-sequence conservation, it is not surprising that many methods succeed in aligning various members of this family reasonably well.

A more rigorous test of these methods would be to measure their ability to identify the highly conserved motifs involved in the function of various protein families. Many of these motifs were first inferred from primary protein-sequence multiple alignment analysis and were confirmed by biochemical and crystallographic

Table 2
Scores for Programs Tested Using Globins

Program and No. of Sequences Tested	Motif I (7 residues)	Motif II (5 residues)	Motif III (5 residues)	Motif IV (5 residues)	Motif V (3 residues)	Parameters/Comments ^a
Global Methods						
AMULT:						
12	100	100	100	100	100	Single-order alignment; defaults except: indel = 8 (4–10) and iteration = 1 (1–4)
10	100	100	100	100	100	
6	100	100	100	100	100	
ASSEMBLE:						
12	100	92	100	100	100	Defaults except: FIL-SUM algorithm
10			Did not perform alignment, since filter produces empty plots ^b			
6	100	100	100	100	100	FIL-LOG, I = 8 (8–12)
CLUSTAL V:						
12	100	92	100	100	100	Defaults; parameters tweaked are: pairwise: indel (1–8) and k-tuple (1–2); multiple alignment: I (6–12) and E (2–10)
10	100	92	100	100	100	
6	100	92	100	100	100	
DFALIGN:						
12	100	100	100	100	100	Defaults
10	100	100	100	100	100	
6	100	100	100	100	100	
GENALIGN:						
12	92 (67, 25) ^c	100	100	83 (67, 17) ^c	92 (67, 25)	Defaults except: match weight = 2; NW
10	90 (60, 30)	90	90 (50, 40)	80 (60, 20)	90 (60, 30)	Defaults except: match weight = 1; NW
6	83	100 ^c	83 (50, 33) ^c	67 (2 × 33)	67 (2 × 33)	

MULTAL:						Matrix weight ^d = 0–5; cycles ^e = 12; indel = 20; window size = 15–50; cutoff score = 900–300; span ^f = 8–128 ^g
12	100	90	100	100	100	
10	100	90	100	100	100	
6	100	90	100	100	100	
TULLA:						RGW = 2–4–6; median 2 or 4 (2–12) RGW = 8 (4–12)
10	90	80	80	80	80	
6	83	83	83	67	83	
Local Methods						
MACAW:						Cutoff score = 30 (20–30); MD = 50% (25%–50%); result list size = 100, for all subsets; several overlapping blocks ^h
12	75	92	75	67	67	
10	70	80	70	60	60	
6	100	67	100	67	67	
PIMA:						E = 0.33; ML clusters ⁱ
12	100	100	100	100	100	
10	100	100	100	100	100	
6	100	100	100	100	100	
PRALIGN:						SB clusters ⁱ
12	67	67 (33, 2 × 17)	75 (33, 25, 17)	67 (33, 2 × 17)	84 (67, 17)	Window size = 20 (10–40); word size = 3 (3–5); MC = 1 (0–2); indel = 0; MD = 30% (20%–50%)
10	50 (30, 20)	60 (3 × 20)	60 (3 × 20)	20	50	
6	67 (2 × 33)	33	33	0	50	

NOTE.—The score for each test is calculated as a percentage of the no. of sequences in each data set in which the motif was identified. Some methods find the correct matches in >1 subset of the data without being able to align these subsets to one another. In these cases, the total percent correct match is a combined score of the subsets (values in parentheses). Abbreviations are as in table 1.

^a Deviations from default parameters are indicated by a dash for a single data set and by a bracket for two data sets or for new parameters used in all tests. The explored range of parameter values is indicated in parentheses.

^b ASSEMBLE tends to produce only “correct” results or nothing.

^c Has gaps in motif(s).

^d Specifies the mix ratio between the identity matrix and the PAM250 (e.g., a weight of 2 indicates a 0.8 [identity matrix] + 0.2 [PAM250] mix).

^e Specifies the no. of attempts the program makes to merge subalignments.

^f Pairwise distance upper limit for the comparison of all sequences.

^g MULTAL allows the user to change parameters for each cycle. Thus, the range shown in some of the parameters indicates the change of that parameter for each cycle.

^h Creates several blocks for each cluster. One has to manually (with the help of the MACAW editor) merge them to get the percentages for each cluster.

ⁱ Creates alignments by using two types of clusters, maximal linkage (ML) clusters (Smith and Smith 1990) and sequence branching (SB) clusters (Smith and Smith 1992).

Table 3
Scores for Programs Tested Using Kinases

Program and No. of Sequences Tested	Motif I (6 residues)	Motif II (1 residue)	Motif III (1 residue)	Motif IV (9 residues)	Motif V (3 residues)	Motif VI (3 residues)	Motif VII (8 residues)	Motif VIII (1 residue)	Parameters/Comments
Global Methods									
by guest order									
AMULT:	12	100	83	92	100	100	100	100	Tree-based alignment
	10	100	90	90	100	100	100	90	Single order alignment; iteration
	6	100	67	67	100	100	100	100	= 4 (1-4)
ASSEMBLE:	12	83	58 (33, 25)	83	100	100	100	100 (67, 33)	Defaults except: FIL-SUM algorithm.
	10	90	30	0	100	100	100	70	FIL-LOG, I = 8 (8-12)
	6	67	0	0	100	100	100	50	
CLUSTAL V:	12	100	92	92 (50, 42)	100	100	100	100 (58, 42)	Defaults; parameters tweaked are: pairwise: indel (1-8) and k-tuple (1-2); multiple alignment: I (6-12) and E (2-10)
	10	100	80 (50, 30)	80	100	100	100	90 (50, 40)	
	6	100	83	67	100	100	100	100 (67, 33)	
DFALIGN:	12	100	100	100	100	100	100	100	Begin weighting sequence 3 with value 2
	10	100	100	100	100	100	100	100	Begin weighting sequence 2 with value 2
	6	100	100	100	100	100	100	67	Begin weighting sequence 2 with value 2

GENALIGN:								
12	100 ^a	75 (42, 33)	83	100	100	100	100 (2 × 50)	92 (67, 25)
10	80 (60, 20)	60 (40, 20)	80	100	100	100 (2 × 50)	100 (2 × 50)	90
6	67	50	83 (50, 33)	100 (2 × 50)	100 (2 × 50)	100 (2 × 50)	100 (2 × 50)	83
MULTAL:								
12	100	75 (58, 17)	83 (50, 33)	100	100	100 (58, 42)	100	Cycles = 14; window size = 15–
10	100	80	50	100	100	100	100	140; cutoff score = 900–200;
6	83	33	67	100	100	100	100	all others as in table 2 ^b
TULLA:								
10	90 ^a	60	80	100	100	90	90	RGW = 8–10–12, median 8
6	83 ^a	83 (50, 33)	67	100	100	100	100	Defaults

Local Methods								
MACAW:								
12	67	0	75	100	100	83	100	0
10	70	0	50	100	100	90	90	0
6	100	0	0	100	100	100	100	50
PIMA:								
12	100	92	92	100	100	100	100	SB clusters ^d ; E = 0.33 (0.2–1.75)
10	100	90	100	90	90	90	90	50 (30, 20) SB clusters ^d
6	100	100	67	100	100	100	100	SB clusters ^d ; E = 0.5 (0.2–1.75)
PRALIGN:								
12	100	84 (2 × 42)	50 (33, 17)	33	75 (42, 33)	75 (42, 33)	33	Window size = 20 (10–40); word
10	90	80 (30, 2 × 20)	20	40	70 (40, 30)	60 (2 × 30)	30	size = 3 (3–5); MC = 1 (0–2)
6	67 (2 × 33)	67 (2 × 33)	0	0	67 (2 × 33)	67 (2 × 33)	33	indel = 0; MD = 30% (20%–50%)

NOTE.—All designations and abbreviations are as in tables 1 and 2.

^a See footnote “c” of table 2.

^b See footnotes “d”–“g” of table 2.

^c See footnote “h” of table 2.

^d See footnote “i” of table 2.

Table 4
Scores for Programs Tested Using Proteases

Program and No. of Sequences Tested	Motif I (3 residues)	Motif II (5 residues)	Motif III (3 residues)	Parameters/Comments
Global Methods				
AMULT:				
12	92	58	83	Tree-based alignment; SD ordering ^a
10	90	80 (50, 30)	70 (40, 30)	Single-order alignment; indel = 8 (4–10); iteration = 1
6	67	0	50	(1–4) Tree-based alignment; SD ordering
ASSEMBLE:				
12				Did not perform alignment, since filter produces empty plots ^b
10				
6				
CLUSTAL V:				
12	100	75 (50, 25)	50 (2 × 25)	Defaults; parameters tweaked are: pairwise: indel (1–8) k-tuple (1–2); multiple alignment: I (6–12), E (2–10)
10	100	70 (40, 30)	70 (30, 2 × 20)	
6	100	0	67	
DFALIGN:				
12	100	100 (70, 30)	100	Begin weighting sequence 3 with value 2
10	100	100 (70, 30)	100	Begin weighting sequence 2 with value 2
6	100	50	83	
GENALIGN:				
12	92	67 (42, 25) ^c	58 (25, 2 × 17)	Defaults except: match weight = 4, deletion weight = 2; NW
10	90 (70, 20)	50 (30, 20) ^c	80 (60, 20) ^c	
6	67	33	0	Defaults except: match weight = 2; NW
MULTAL:				
12	83	58 (33, 25)	75 (50, 25)	Cycles = 14; cutoff score = 900–200; all others as in table 2 ^d
10	90 (50, 40)	70 (30, 2 × 20)	90 (50, 40)	
6	50	0	33	
TULLA:				
10	70	50 (30, 20)	70 (40, 30)	RGW = 2–4–6 median 4 (2–12)
6	83	33	0	RGW = 6–8–10 median 8 (2–12)
Local Methods				
MACAW:				
12	100	25	67	Cutoff score = 20 (10–20); MD = 25%, 30%, 33% (20%–50%); result list size = 100, for all subsets; several overlapping blocks ^e
10	100	30	70	
6	100	0	33	
PIMA:				
12	100	42 (25, 17)	42 (25, 17)	SB clusters ^f
10	100	60 (40, 20)	70	SB clusters ^f ; E = 0.33 (0.2–1.75)
6	100	0	33	SB clusters ^f
PRALIGN:				
12	67 (2 × 33)	34 (2 × 17)	67 (2 × 25, 17)	Window size = 20 (10–40); word size = 3 (3–5); MC = 1 (0–2); indel = 0; MD = 30% (20%–50%)
10	100 (40, 2 × 30)	30	70 (30, 2 × 20)	
6	100 (3 × 33)	0	30	

NOTE.—All designations and abbreviations are as in tables 1 and 2.

^a SD ordering uses the standard deviation between sequence pairs to form an order.

^b See footnote "b" of table 2.

^c See footnote "c" of table 2.

^d See footnotes "d"–"g" of table 2.

^e See footnote "h" of table 2.

^f See footnote "i" of table 2.

analysis. In addition to the hemoglobins, therefore, we have analyzed three such data sets: the kinase family, the aspartic acid protease family (both eukaryotic and viral), and the RH region of both the RNA-directed DNA polymerase (the reverse transcriptase) and the *Escherichia coli* RH enzyme.

From each family we have selected a representative set of sequences with a broad phylogenetic distribution. The percentage range of identical residues among all sequence pairs in the hemoglobin data set is 10%–70%. The percentage range of identical residues among all sequence pairs within each of the enzymatic data sets is 8%–30%, indicating that only those residues involved in function are conserved among these highly divergent sequences. The alignments of figures 1–3 were extracted from larger alignments (50–65 sequences) produced by the program DFALIGN and corrected manually. All sets of test sequences are available through EMBL (identification no. DS16117).

The hemoglobin data set includes α - and β -globins from mammals and birds; myoglobins from mammals; and hemoglobins from insects, plants, and bacteria. We designated five regions of the alignment to serve as the ordered series of motifs defining the globin family. There is no external measure of the authenticity of this choice, as there is in the case of enzymatic protein families (see below). The decision was made to provide a test for the globins that is consistent with the tests of the kinase, aspartic acid protease, and RH families. We score for five motifs that are conserved or semiconserved throughout the phylogenetic distribution of the globin family. Motif I is essentially helical region C, motifs II and III, in helical regions E and F respectively, are within the heme-binding region, and motifs IV and V are in helical regions G and H, respectively (fig. 1) (Bashford et al. 1987).

The eukaryotic kinase proteins constitute a large enzymatic family that regulates the most basic of cellular processes. These proteins have been categorized by primary sequence analysis on the basis of the conservation of the ordered series of eight motifs found in their catalytic domains (Hanks and Quinn 1991) (fig. 2). Crystallographic studies of the cyclic adenosine monophosphate-dependent protein kinase confirm that most of the conserved motifs of the kinase protein core do cluster into the regions of the protein involved in nucleotide binding and catalysis (Knighton et al. 1991). The kinase data set includes serine/threonine, tyrosine, and dual specificity kinases from mammals, birds, fungi, retroviruses, and herpes viruses.

The eukaryotic aspartic acid protease family consists of pepsins, chymosin, and renins. These proteases have two domains. Each domain has an ordered series

of three motifs that contribute to the active site of the enzyme. The most prominent motif is three consecutive, conserved residues—aspartic acid, threonine, and glycine (single-letter code, "DTG") (fig. 3). It has been suggested that the aspartic acid proteases evolved through duplication of a single-domain prototype (Tang et al. 1978). The retrovirus aspartic acid proteases are about half the size of the cellular proteases. Primary sequence analysis of retrovirus proteases indicated an ordered series of three motifs, suggesting that they function as dimers and that they diverged from the eukaryotic aspartic acid proteases prior to the latter group's duplication event (Pearl and Taylor 1987; Doolittle et al. 1989). Crystallographic studies subsequently confirmed the dimeric nature and catalytic site of the retrovirus aspartic acid proteases, as predicted from primary sequence analysis (Miller et al. 1989). The aspartic acid protease data set includes pepsin (only the amino-terminal domain of this double-domain protease) from mammals, birds, and fungi and from representative members of the retrovirus family, such as retroviruses, caulimoviruses, and retrotransposons (fig. 3) (McClure 1992).

The RH domain of the RNA-directed DNA polymerase (reverse transcriptase) of the retrovirus elements resides in the carboxyl one-third of the protein. Amino acid sequence comparisons of the retroviral proteins correctly predicted the position of the RH activity in the RNA-directed DNA polymerase by identification of motifs conserved with the *E. coli* RH sequence (Johnson et al. 1986). Subsequent mutational studies confirmed the predicted position (Tanese and Goff 1988). The highly conserved motifs of the retrovirus family and *E. coli* RH proteins have been shown to cluster in the catalytic site, as identified in the crystal structures of the *E. coli* RH protein (Katayanagi et al. 1990) and the HIV-I RH domain (Davies et al. 1991) (fig. 4). The RH data set includes sequences from *E. coli* and representative members of the retrovirus family, including retroviruses, caulimoviruses, hepadnaviruses, retrotransposons, retroviruses, and group II plasmids of filamentous ascomycete mitochondria (McClure 1993).

Subsets of 6, 10, and 12 sequences were used to assay the ability of each method to identify the ordered series of motifs defining each protein family. There are two reasons for varying the sequence number: (1) by varying the number of subsets of sequences tested, we could evaluate the effects of both the sensitivity to the number of sequences and to specific sequences in each test; and (2) some methods can only handle small data sets (table 1). Each six-sequence data set contains the widest distance distribution of sequence relationship. The 10- and 12-sequence data sets were created by addition of sister sequences to the 6-sequence data sets.

Table 5
Scores for Programs Tested Using RH

Program and No. of Sequences Tested	Motif I (3 residues)	Motif II (1 residue)	Motif III (3 residues)	Motif IV (5 residues)	Parameters/Comments
Global Methods					
AMULT:					
12	92	75 (58, 17)	67 (50, 17)	59 (25, 2 × 17)	Single-order alignment; defaults except: iteration = 4 (1–4)
10	100	70	60	90 (60, 30)	
6	100	83 (50, 33)	67	80 (50, 33)	
ASSEMBLE:					
12					
10					
6					
Did not perform alignment, since filter produces empty plots ^a					
CLUSTAL V:					
12	100	75	75 (58, 17)	75 (33, 25, 17)	Defaults; parameters tweaked are: pairwise: indel (1–8) and <i>k</i> -tuple (1–2); multiple alignment: I (6–12) and E (2–10)
10	100	70	70	80 (2 × 30, 20)	
6	100	67	50	50	
DFALIGN:					
12	100	100	83	100	Begin weighting sequence 3 with value 3
10	100	60	70	100	Begin weighting sequence 4 with value 3
6	100	100	67	100	Begin weighting sequence 2 with value 2
GENALIGN:					
12	100 (83, 17) ^b	58	67 (33, 2 × 17) ^b	75 (33, 25, 17) ^b	Defaults except: NW; match weight = 1
10	80 ^b	90	70 (40, 30) ^b	90 (30, 3 × 20) ^b	
6	100 ^b	67	67	67	

MULTAL:

12	92 (75, 17)
10	100 (70, 30)
6	100

92 (58, 2 × 17)
90
83

75 (50, 25)
80 (60, 20)
67

83
70
83

Cycles = 14; cutoff score = 900–200; All others
as in table 2^c

TULLA:

10	100 ^b
6	100

50
50

40
67

80 (2 × 40)
50

Defaults except: RGW = 8–10–12 median 8

Local Methods**MACAW:**

12	58
10	80
6	83

42
70
67

58
70
67

17
40
67

Cutoff score = 20 (10–20); MD = 25%, 30%,
33% (20%–50%); result list size = 100, for all
subsets; several overlapping blocks^d

PIMA:

12	83
10	100 (80, 20)
6	100

75
80
100

67 (33, 2 × 17)
80 (40, 2 × 20)
67

92 (42, 33, 17)
90 (70, 20)
83 (50, 33)

ML clusters^e; E = 0.2 (0.2–1.75); I = 5.5 (5–7)
ML clusters^e; E = 0.33 (0.2–1.75)

PRALIGN:

12	75
10	80
6	83

67 (2 × 33)
80 (60, 20)
67 (2 × 33)

50 (33, 17)
40
33

17
20
50

Window size = 20 (10–40); word size = 3
(3–5); MC = 1 (0–2); indel = 0; MD = 30%
(20%–50%)

NOTE.—All designations and abbreviations are as in tables 1 and 2.

^a See footnote "b" of table 2.

^b See footnote "c" of table 2.

^c See footnotes "d"–"g" of table 2.

^d See footnote "h" of table 2.

^e See footnote "i" of table 2.

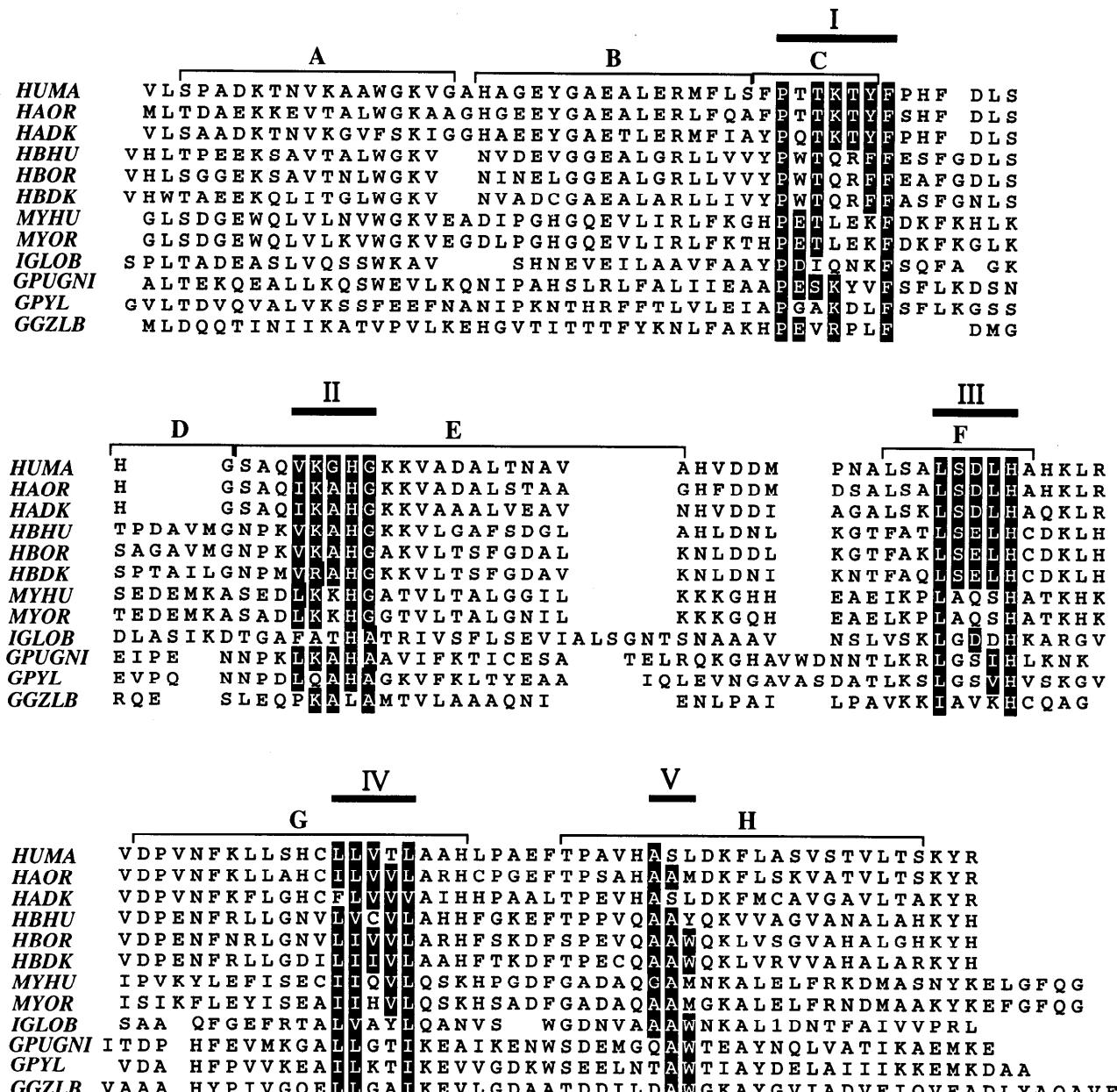


FIG. 1.—Multiple alignment of representative globin sequences. The five motifs scored for in the comparative analysis are indicated by blackened bars and the numerals I–V. Black/white reversals of columns within the motifs indicate the most conserved residues of the motifs and their conservative substitutions, based on the similarity scheme (F,Y), (M,L,I,V), (A,G), (T,S), (Q,N), (K,R), and (E,D). If the same number of matches occurs for more than one residue in a column, then one set is arbitrarily chosen for black/white reversal. The conserved helices of the globins are indicated by overlined regions and the letters A–H. The set of 12 sequences includes HAHU (human), HAOR (duckbill platypus), and HADK (duck) α -chain hemoglobins and HBHU (human), HBOR (duckbill platypus), and HBDK (duck) β -chain hemoglobins. MYHU (human) and MYOR (duckbill platypus) are myoglobins. The remaining hemoglobin sequences are IGLOB (insect, *Chironomus thummi*), GPYL (legume, yellow lupine), GPUGNI (nonlegume, swamp oak), and GGZLB (bacteria, *Vitreoscilla* sp.). The two other test sets of globin sequences are subsets of these sequences; set 10 = set 12 without HAOR and HBOR, and set 6 is comprised of HAHU, HBHU, MYHU, IGLOB, GPYL, and GGZLB.

The sequences of the four protein families tested display a wide range of motif density, motif conservation, and indels. The globins are highly conserved with few indels, and the five motifs range in size from three to

seven amino acids (fig. 1 and table 2). The kinase family has well-defined indel regions interspersed among eight highly conserved motifs, each of which varies from one to nine amino acid residues in size (fig. 2 and table 3).

The aspartic acid protease and RH sequences have the greatest range of motif density, motif conservation, and indels (figs. 2 and 3). The size of the three motifs of the protease is from three to five amino acid residues, and the four motifs of the RH data set vary from one to five amino acid residues (tables 4 and 5). These latter two tests are more difficult than either the globin or kinase tests.

Description of Alignment Methods Analyzed

Multiple Alignment Strategies

There are two basic software approaches in determining the similarity among proteins. The following global methods construct an alignment throughout the length of the entire sequence: AMULT (Barton and Sternberg 1987a, 1987b), DFALIGN (Feng and Doolittle 1987), MULTAL (Taylor 1987, 1988), MSA (Lipman et al. 1989), TULLA (Subbiah and Harrison 1989), CLUSTAL V (Higgins et al. 1992), and MWT (Kececioglu 1993). A subclass of global methods attempts first to identify an ordered series of motifs and then proceeds to align the intervening regions, e.g., GENALIGN (Martinez 1988), and ASSEMBLE (Vingron and Argos 1991). Local methods only attempt to identify an ordered series of motifs while ignoring regions between motifs, e.g., PIMA (Smith and Smith 1990, 1992), PRALIGN (Waterman and Jones 1990), and MACAW (Schuler et al. 1991). Brief descriptions of the basic algorithms, scoring matrices, and penalties for indels used in all the methods analyzed are presented below (table 1).

Global Methods

The diagram in figure 5 summarizes the basic implementation of the various algorithms employed in the nine different global multiple alignment methods analyzed (table 1) (Barton and Sternberg 1987a, 1987b; Feng and Doolittle 1987; Taylor 1987, 1988; Martinez 1988; Lipman et al. 1989; Subbiah and Harrison 1989; Vingron and Argos 1991; Higgins et al. 1992; Kececioglu 1993). Table 1 indicates the various algorithms employed by each method. In light of the computational expense of simultaneous comparison of protein sequences, all methods begin by comparing all sequences in a pairwise fashion. Several methods cluster the sequences into subalignments by using a similarity measure (GENALIGN and MULTAL) or a phylogenetic tree (CLUSTAL V, AMULT, and DFALIGN). GENALIGN, MULTAL, and CLUSTAL V subsequently align the clustered subalignments to one another by employing various consensus methods that reduce each subalignment to a single consensus sequence. Allowing the subalignments to be merged by aligning their con-

sensus sequences to one another produces a progressive multiple alignment. In addition, GENALIGN allows the user to chose either the Needleman-Wunsch (NW) or consensus word (CW) algorithms (for definitions, see the section Basic Algorithms) for the alignment, while CLUSTAL V permits the user to specify individual parameters for both the pairwise and multiple alignment stages. AMULT and DFALIGN produce a progressive multiple alignment directly from the clustering stage. AMULT then produces a final multiple alignment through optimization of the progressive multiple alignment. A novel feature of AMULT provides the option of producing a progressive multiple alignment directly from the pairwise ordering stage, bypassing the phylogenetic clustering stage. Two methods (MSA and TULLA) produce a progressive multiple alignment and then a final multiple alignment. The MSA method can also produce a final multiple alignment, bypassing the progressive multiple alignment stage, if the user supplies the upper bounds for all sequence pairs that is necessary for the multidimensional dynamic programming on a restricted space. ASSEMBLE and MWT produce a final multiple alignment directly from the pairwise analysis. The MSA and MWT methods differ from the others because they compute an optimal multiple alignment with respect to a well-defined multiple alignment scoring function. The source code for GENALIGN has been licensed to IntelliGenetics and, therefore, is no longer available. All other developers have made their source code available upon request, as is the standard practice in the scientific community.

The concept of a progressive multiple alignment has been suggested by several developers (Waterman and Perlitz 1984; Feng and Doolittle 1987; Taylor 1987). This approach begins with alignment of the two most closely related sequences (as determined by pairwise analysis) and subsequently adds the next closest sequence or sequence group to this initial pair. This process continues, in an iterative fashion, adjusting the positioning of indels in all sequences. The major shortcoming of this approach is that a bias may be introduced in the inference of the ordered series of motifs because of an overrepresentation of a subset of sequences. More recently developed methods, such as MSA, use a sequence-weighting scheme to correct for this potential problem (table 1) (Altschul et al. 1989).

Local Methods

We have analyzed three local multiple alignment methods (table 1). MACAW (*multiple alignment construction workbench*) automatically performs multiple alignment of input sequences and also provides a multiple alignment sequence editor (Schuler et al. 1991). This method begins with pairwise analysis of all se-

	I	II	III
CAPK	D Q F E R I K T L G T S F G R V M L V K H M E	T G N H Y A M K I L D K Q K V V K L K Q I E H	T L N E K R I L Q A V N F P F L V
MLCK	F S M N S K E A L G G K F G A V C T C T E K S	T G L K L A A K V I K Q T P K D K E M V M L E I E V M N Q L N H R N L I	
PSKH	A K Y D I K A L I C R G S F S R V V R V E H R A	T R Q P Y A I K M I E T K Y R E G R E V C E S E L R V L R R V R H A N I I	
CD28	A N Y K R L E K V G E G T Y G V V Y K A L D L R P G	Q G Q R V V A L K K I R L E S E D E G V P S T A I R E I S L L K E L K D D N I V	
WEE1	T R F R N V T L L G S G E F S E V F Q V E D P V E	K T L K Y A V K K L K V K F S G P K E R N R L L Q E V S I Q R A L K G H D H I V	
RAFI	S E V M L S T R I G G S F G T V Y K G K W H G D	V A V K I L K V V D P T P E Q F Q A F R N E V A V L R K T R H V N I L	
CMOS	E Q V C L L Q R L G A G G F G G S V Y K A T Y	R G V P V A I K Q V N K C T K N R L A S R R S F W A E L N V A R L R H D N I V	
CSRC	E S L R L E V K L G Q G C F G G E V W M G T W N	G T T R V A I K T L K P G N M S P E A F L Q E A Q V M K K L R H E K L V	
VFES	E D L V L G E Q I G R G N F G G E V F S G R L R A D	N T L V A V K S C R E T P P D I K A K F L Q E A K I K L Q Y S H P N I V	
PDGM	D Q L V L G R T L G S G A F G Q V V E A T A H G L S H S Q A T M K V A V K M L K S T A R S S E K Q A L M S E L Y G D L V D Y L H R N K H		
EGFR	T E F F K K I V K L G S G A F G T V Y K G L W I P E G E K V K I P V A I K E L R E A T S P K A N K E I L D E A Y V M A S V D N P H V C		
HSVK	M G F T I H G A L T P G S E G C V F D S S H P D	Y P Q R V I V K A G W Y T S T S H E A R L L R R L D H P A I L	
	IV	V	VI
CAPK	H S L D L I Y R D L K P E N L L I D Q Q G Y I Q V T	D F G F A K R V K G R T W T L C G T P E Y L A E E I I	R F S E P H A R F Y A A Q I V L T F E Y L
MLCK	H K M R V L H L D L K P E N I L C V N T T G H L V K I I	D F G L A R R Y N P N E K L K V N F G T P E F L S P E V	E V D T M V F V R Q I C D G I L F M
PSKH	H A L G I T H R D L K P E N L L Y Y H P G T D S K I I I T	D F G L A S A R K K G D D C L M K T T C G T P E Y I A F E V L	E R D A T R V L Q M V L D G V R Y L
CD28	H S H R I L H R D L K P Q N L L I N K D G N L K L G	D F G L A R A F G V P L R A Y T H E I V T L W Y R A F E V L	P L G A D I V V K K F M M Q L C K G I A Y C
WEE1	H H K N Y V H L D L K P A N V M I T F E G T L K I G	D F G M A S V W P V P R G M E R E G D C E Y I A F E V L	R L D E F R V W K I L V E A V G L G L Q F I
RAFI	H A K N I I H R D M K S N N I F L H E G L T V K I G	D F G C L A T V K S R W S G S Q Q V E Q P T G S V L W M A F E V I R M Q D N N	K F Q M F Q L I D I A R Q T A Q G M D Y I
CMOS	H S Q S I V H L D L K P A N I I L I S E Q D V C K I S	D F G C S E K L E D L L C F Q T P S Y P L G G T Y T H R A E L	L S L G K C L K Y S L D V V N G L L F I
CSRC	E R M N Y V H R D L R A A N I I L V G E N L V C K V A	D F G L A R L I E D N E Y T A R Q G A K F P I K W T A P E A A	R L P Q L V D M A A Q I A S G M A Y V
VFES	E S K C C I I H R D L A A R N C L V T E K N V L K I S	D F G M S R E A A D G I Y A A S G G L R Q V P V W T A P E A L	R L R M K T L L Q M V G D A A A G M E Y I
PDGM	A S K N C V H R D L A A R N V L I C E G K L V K I C	D F G L A R D I M R D S N Y I S K G S T Y L P L K W M A E S I	I G S Q Y L L N W C V Q I A K G M N Y I
EGFR	E D R R L V H R D L A A R N V L V K T P Q H V K I T	D F G L A K L G A E E K E Y H A E G G K V P I K W M A L E S I	P L G R P Q I A A V S R Q L L S A V D
HSVK	H R Q G I I H R D I K T E N I F I N T P E D I C L G	D F G A A C F V Q G S R S S P F P Y G I A G T I D T N A E E V	PLAGI
	VII		
CAPK	G Y N K A V D W W A L G V L I I Y E M A A G Y P P F F A	D Q P I Q I Y E K I V S G K V R F P S H	
MLCK	Q I S D K T D M W S L C G V I T Y M L L S G L S P F L G	D D D T E T L N N V L S G N W Y F D E E T F E A	
PSKH	P Y T N S V D M W A L G V I A Y I L L S G T M P F	E D D N R T R L Y R Q I L R G K Y S Y S G E P W P S	
CD28	Q Y S T G V D T W S I G C I F A E M C N R K P I F S G D S	E I D Q I F K I F R V L G T P N E A I W P D I V Y L P D F	
WEE1	L Y D K P A D I F S L G I T V F E A A A N I V L P	D N G Q S W Q K L R S G D L S D A P R L S S T D N	
RAFI	P F S F Q S D V Y S Y G I V L Y E L M T G E L P Y S	R D Q I I F M V G R G Y A S P D L S K L Y K	
CMOS	G V T P K A D I F S L G I T L W Q M T T K O A P Y S G	E R Q H I L Y A V V A Y D L R P S L S A	
CSRC	F T I K S D V W S F G I I L L T E L T T K G R V P Y P G M V N R E V L D Q	V E R G Y R M P C P P	
VFES	Y S S E S D V W S F G I I L L W E T F S L G A S P Y P N L S	N Q Q T R E F V E K G G R L P C P E	
PDGM	L Y T T L S D V W S F G I I L L W E I F T L G G T P Y P E L P	M N D Q F Y N A I K R G Y R M A Q P A	
EGFR	I Y T H Q S D V W S Y G V T V W L E M T F G S K P Y	D G I P A S E I S S I L E K G E R L P Q P P	
HSVK	P Y T T T V D I W S A G L V I F E T A V H N A S L F S A P R	G P K R G P C D S	
	VIII		
CAPK	F S S D L K D L L R N L	L Q V D L T K R F G N L K D G V N D I K N H K	
MLCK	V S D E A K D F V S N L	I V K E Q G A R M S A A Q C L A H P W L N N L	
PSKH	V S N L A K D F I D R L	L T V D P G A R M T A L Q A L R H P W V V S M	
CD28	S F P Q W R R K D L S Q V V P S L D P R G I D L L D K L	L A Y D P I N R I S A R R A I H P Y F Q E S	
WEE1	S L T S S S R E T P A N S I I G Q G G L D R V V E W M	L S P E P R N R P T I D Q I L A T D E V C W V	
RAFI	N C P K A M K R L V A D C V K K V K E E R P L F P Q I L S S I E L L Q H		
CMOS	F E D S L P G Q R L G D V I Q R C W R P S A A Q R P S A R L L V D L T S L K A		
CSRC	E C P E S L H D L M C Q C W R R D P E E R P T F E Y L Q A F L E D Y F T		
VFES	L C P D A V F R L M E Q C W A Y E P G Q R P S F S A I Y Q E L		
PDGM	H A S D E I Y E I M Q K C W E E K F E T R P P F S Q L V L L E R L L G E G K K K Y		
EGFR	I C T I D V Y M I M V K C W M I D A D S R P K F R E L I I E F S K M A R		
HSVK	Q I T R I I R Q A Q V H V D E F S P H P E S R L T S R Y R S R A A G N N R P P Y T R (P A W T R Y Y K M D I D V E Y L V C K A L T F D G A L R P S A A E L L C L P L F Q Q K)*		

Downloaded from www.jbc.org/ by guest on April 26, 2016

quences, to identify potential motifs. Only those motifs found in all pairwise alignments are coalesced into blocks that the user can then manipulate with the on-screen editor. The PIMA method begins with a pairwise analysis of all sequences, then constructs a tree on the basis of this order and derives a pattern at each node by using the progressive alignment approach (Smith and Smith 1990, 1992). This is continued in an iterative fashion until a root consensus pattern is achieved using the amino acid class hierarchy (see Scoring Matrices). PRALIGN is a method based on the CW approach (Waterman 1986; Waterman and Jones 1990). Words are found on the basis of user-specified word length (number of contiguous residues) and window length (number of consecutive residues to search within for a word) and motif density and motif conservation parameters (for definitions, see Methods Used for Comparative Analysis of Alignment Programs).

Basic Algorithms

The biologically interesting formulations of the multiple alignment problem are in the class of so-called NP-complete problems (i.e., nondeterministic polynomial time complete problems). This implies that algorithms that can find an optimal multiple alignment for *any set* of input sequences—called “exact algorithms”—are unlikely to be efficient. However, exact algorithms that can efficiently find an optimal alignment for *specific sets* of sequences exist, and some are known (Carrillo and Lipman 1988; Kececioglu 1993) and are included in this analysis (e.g., MSA and MWT). Algorithms that can efficiently find an alignment that is guaranteed to be close to the optimal alignment—called “approximation algorithms”—are possible, and some have recently been described (Gusfield 1993; Pevzner 1993). Whether the best alignment produced by these new algorithms includes the ordered series of motifs that define a given protein family remains to be determined. Only the algorithms and approaches implemented in the multiple alignment methods in this study will be briefly described.

The dot matrix approach has been used extensively in sequence analysis. In brief, a two-dimensional array

of two sequences is created, and a dot is placed for matches. In the ASSEMBLE method the dot matrix is initially employed as a filter to identify and retain only those motifs that are conserved among a given set of sequences, prior to the use of dynamic programming. States and Boguski (1990) have written an elegant history and detailed description of the various biological applications of the dot matrix method.

Most of the methods compared here employ dynamic programming, which finds an optimal alignment for two sequences on the basis of various scoring schemes. The scoring scheme is usually based on a value for matches and replacements (see below) and on a penalty for indels (see below). The major shortcoming of this approach, when applied to more than two sequences, is that it requires intensive computer time (CPU time) proportional to N^K , where K is the number of sequences, and N is their average length. In 1970, Needleman and Wunsch wrote the first dynamic programming algorithm for the global comparison of two sequences. In brief, a two-dimensional array of the sequences is employed to find maximal matches while penalizing for indels (Needleman and Wunsch 1970). This method has formed the basis of most of the subsequent extensions to higher-dimensional arrays for multiple sequence alignment. A significant reduction in CPU time for the case of two sequences, with little loss in sensitivity, was achieved by the use of the dot matrix method coupled to the NW algorithm, resulting in the Wilbur-Lipman (WL) algorithm (Wilbur and Lipman 1982). Another improvement to the NW algorithm, when extended to multiple sequences, was achieved by the use of pairwise alignments to restrict the search for optimal paths among multiple sequences, thus creating the Carrillo-Lipman (CL) algorithm (Carrillo and Lipman 1988).

Two of the three local multiple alignment methods analyzed here employ the Smith-Waterman (SW) algorithm (Smith and Waterman 1981). This algorithm was the first useful approach for identifying subsequences within larger sequences, and it allows for indels of arbitrary length within the subsequence. The use of this algorithm in the MACAW alignment editor, however,

FIG. 2.—Multiple alignment of representative eukaryotic kinase-protein sequences. The eight motifs scored for in the comparative analysis are indicated by blackened bars and the numerals I–VIII. CAPK (bovine cardiac muscle), MLCK (rat skeletal muscle), PSKH (HeLa cell), CD28 (*Saccharomyces cerevisiae*), and CMOS and RAF1 (human oncogenic proteins) are the sequences of serine/threonine-specific kinase proteins. WEE1 is a dual specificity kinase from *S. pombe*. CSRC (chicken oncogenic protein), VFES (feline sarcoma virus oncogenic protein), PDGMR (mouse, PDGF receptor), and EGFR (human, EGF receptor) are sequences of tyrosine-specific kinase proteins. HSVK is the herpes simplex-virus kinase. The asterisk and residues in parentheses indicate a HSVK duplication that provides a second conserved motif VIII. Numbers in parentheses indicate the number of amino acids in insertion/deletion positions not included in the alignment display. All other designations are as in fig. 1. The two other test sets of kinase sequences are subsets of these sequences; set 10 = set 12 without MLCK and CSRC, and set 6 is comprised of CAPK, CD28, WEE1, VFES, PDGMR, and EGFR.

does not allow the introduction of indels within a subsequence.

One global method (GENALIGN) and one local method (PRALIGN) are based on the CW approach to the multiple alignment problem (Karlin et al. 1983; Waterman 1986). It is assumed that the CWs defining a given protein family are unknown. All subsequences of a specific word size are then searched for within a given window among all the input sequences. Waterman and Jones (1990) have written a detailed description of

the CW approach applied to both DNA and protein sequences.

Scoring Matrices

Various types of amino acid exchange matrices are available to assist in aligning protein sequences (Fitch and Margoliash 1967; Dayhoff et al. 1978; Feng et al. 1985; Taylor 1986; Rao 1987; Risler et al. 1988). Values for replacing one residue with another are based on physical/chemical similarities,

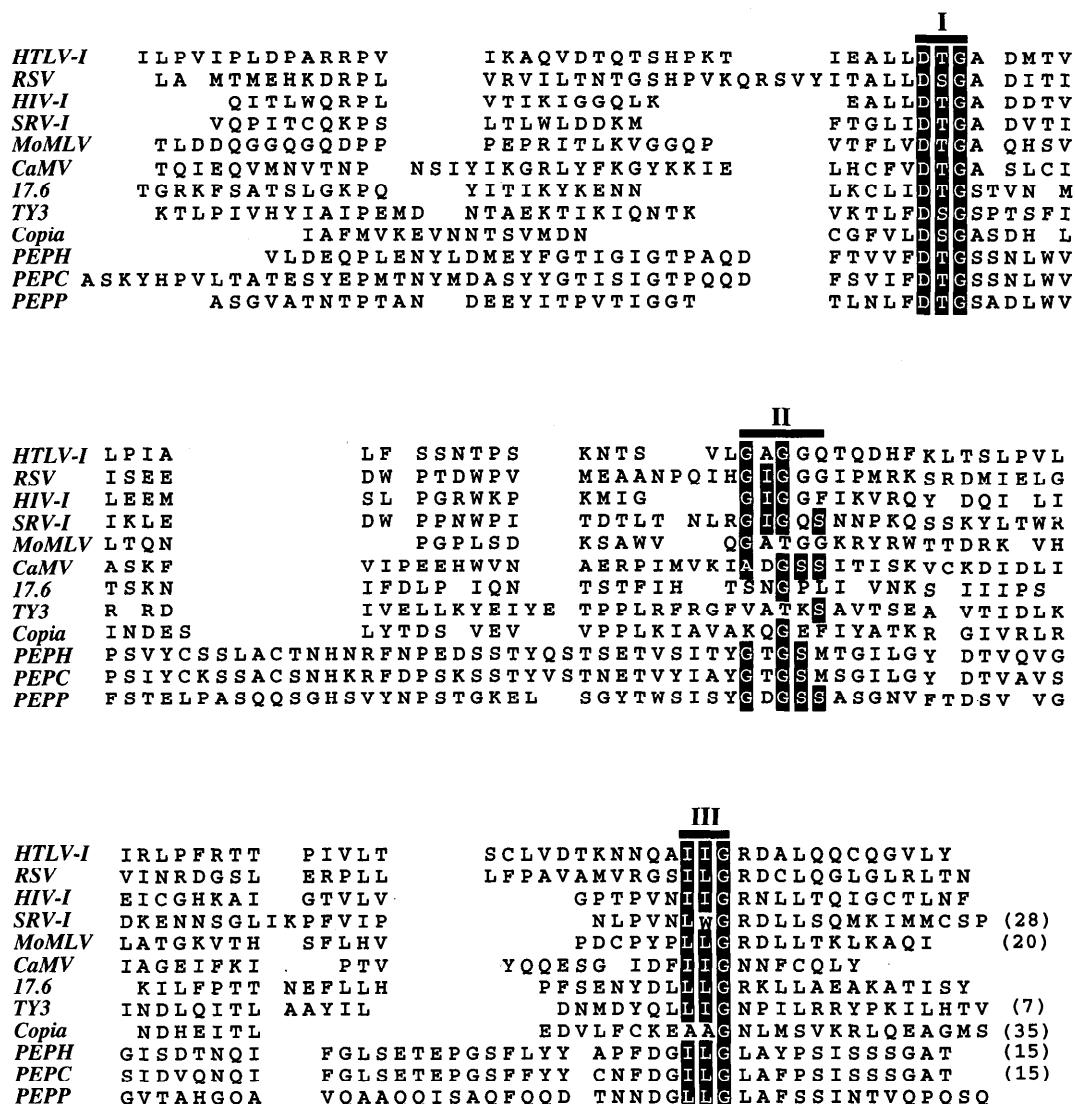


FIG. 3.—Multiple alignment of representative aspartic acid protease sequences. The three motifs scored for in the comparative analysis are indicated by blackened bars and the numerals I–III. The retrovirus family aspartic acid protease sequences are from the retroviruses HTLV-I (human T-cell leukemia virus, type I), RSV (Rous sarcoma virus), SRV-I (simian retrovirus, type I), HIV-I (human immunodeficiency virus, type I), and MoMLV (Moloney murine leukemia virus); the caulimovirus CaMV (cauliflower mosaic virus); and the retrotransposons Copia and 17.6 (*Drosophila melanogaster*) and TY3 (*Saccharomyces cerevisiae*). PEPH (human), PEPC (chicken), and PEPP (fungus, *Penicillium janthinellum*) are the amino-terminal half of pepsin sequences. All other designations are as in figs. 1 and 2. The two other test sets of aspartic acid protease sequences are subsets of these sequences; set 10 = set 12 without SRV-I and 17.6, and set 6 is comprised of PEPH, MoMLV, CaMV, COPIA, 17.6, and TY3.

	I			II			
HTLV-II	LDTAP	CLFS	DGS	PQ	KAAYVLWDQ	TILQQDITPLPS	HETHSAQKG ELL
SRV-I	LNNAL	LVFT	DGS	STG	MAAYTLAD	TTIKFQTN	LN SAQLVE LQ
RSV	PVPGP	TVFT	TDA	SSSTH	KGVVV	WREGPRWEIKEIAD	LGASVQQL EAR
HIV-II	IPGAE	TFYT	TDG	SCNRQSKEG	KAGYV	TDRGKDKVKLE	QTINQQA ELE
MoMLV	PDADH	TWYT	TDG	SSLQEGQR	KAGAAV	TTETEVIWAKALD	AGTSAQRA ELI
Ingi	PREHY	KLWT	TDG	VSLGE	KLGAAALLHRNNTLICAPKTGAGELSCSYRAECVAL	EIG	
CaMV	PEEKL	IIE	TDA	SDDYWGGML	KAIIKINEGT	NTELICRYASGSFKAAE	KNYHSNDK ETL
17.6	FTKKF	TLT	TDA	SDVALGAVL	RTLNEHE	INYSTIEK ELL	
Maup	FNNSTNL	QEPS	DSR	LLYR	KLSEEKHGLVPK	FLEKLRE IN	
HBV	RPGL	CQVFADAT			PTGWGLVM	GHQRMRGTFSA	PLPIHTA ELL
Copia	FENKI	IGYV	SDWAGSEIDR		KSTTGYLFKM	FDFNLICWNTKRQN	SVAASSTEAE
E.coli	MLKQVE	IFT	DGS	CLGNPG	PGGYGAIL	RYRGREKTFSA	TRTTNNRME LML

	III			IV			
HTLV-II	ALICGLR	AAKPWP	PSL	NIFLD	DSKYLIKYLH	SLAIGA	FL
SRV-I	ALIAVLS	AFTP	NQPL	NIYTDS	ASAYLAHSIP	LLETVAQI	K
RSV	AVAMALL	LWPT		NVVTDS	AFVAKM	LLKMGQE	G
HIV-II	AFAMALTD	SGPKV		NIIVDS	QYVM	G ISA SQP	T
MoMLV	ALTQALKMAE		GKKL	NVYTDS	RYAFATAHIH	GEIYRRGGLTS	E
Ingi	LQR LLK	WLPRYR	STPS RL	SIFSDS	LSMLT	ALQTGPLAV	T
CaMV	AVINTIK	KFSIYL	TPV HF	LIRTDS	DNTH	FKSFVNLY	
17.6	AIWWATK	TFRHYLL	GRHF	EISSD	HOPLS	WLYRMK	
Maup	FALDKVD			VTEIDS	SKSRLMKFSVSAA	DEVGTALKSLFKFRNS	
HBV	AACFARS	RSGAN		IIGTD	NSVVL	SRKY TSFPWLLGC	CAANW
Copia	YMALFEAV	REALWLK	FLLTSINIKLENPI	KIYEDDNQ	GCIS		
E.coli	AAIVAL	EALKEH	CEV	ILSTD	DSOYVRQ	G ITQWIHNWK KRGW	

FIG. 4.—Multiple alignment of representative RH sequences. The four motifs scored for in the comparative analysis are indicated by blackened bars and the numerals I-IV. The retroid family RH sequences are from the retroviruses HTLV-II (human T-cell leukemia virus, type I) and HIV-II (human immunodeficiency virus, type II); the hepadnavirus HBV (human hepatitis B virus, ayw strain); the retroposon Ingi (*T. brucei*), and the group II mitochondrial plasmid Maup (Mauriceville, 1c strain) of *Neurospora crassa*. *Escherichia coli* is the ribonuclease H from *E. coli*. Other abbreviations are as in fig. 3. All other designations are as in figs. 1 and 2. The two other test sets of RH sequences are subsets of these sequences; set 10 = set 12 without HBV and Maup, and set 6 is comprised of PEPH, MoMLV, CaMV, 17.6, and TY3.

ease of mutating one codon to another, and/or the observed frequency at which replacement occurs in closely related proteins. A widely accepted method for generating exchange matrices is the accepted point mutation (PAM) model (Dayhoff et al. 1978). To alleviate matrix bias we have evaluated all but two methods with a PAM250 matrix (PAM120 did not produce significantly different results). Although the method of Dayhoff provides scores for replacement between all amino acids, the highest scoring replacements are based on the similarity scheme (F, Y), (M, L, I, V), (A, G), (T, S), (Q, N), (K, R), and (E, D).

The amino acid class hierarchy is intrinsic to the PIMA method; therefore, this method cannot be evaluated with any other scoring scheme. This hierarchical classification scheme gives a score of three for identical residues, a score of two for some conservative replacements, and a score of one for broad-based similarities (e.g., all charged residues). Although this scheme groups the amino acids into hierarchical classes on the basis of side-chain physiochemical properties, it does not allow for all known conservative replacements (Smith and Smith 1990). The source code for GENALIGN is unavailable; therefore, we are unable to change the imbedded unitary matrix to the PAM matrix.

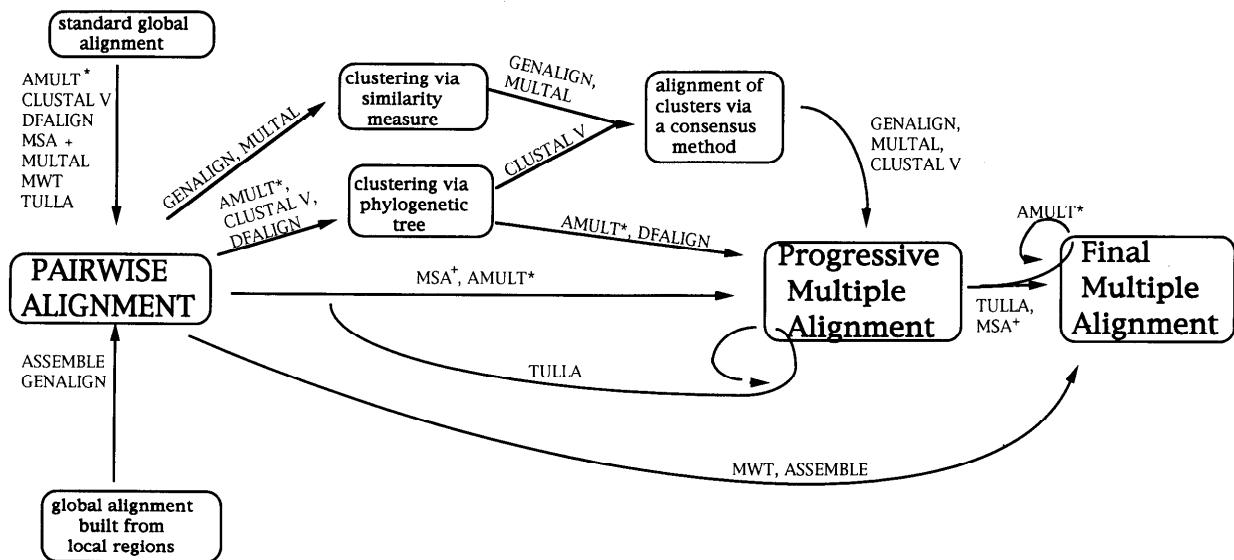


FIG. 5.—Schematic representation of the basic strategies employed by nine different global multiple alignment methods. All methods perform initial pairwise alignments and then progress through various stages, before producing a progressive or final multiple alignment. The loop on the TULLA and AMULT methods indicates that an optimization procedure can be performed on the multiple alignment at the indicated stage. All abbreviations are as in the text. The asterisks (*) indicate one of two user-specified strategies for the AMULT program. The plus sign (+) indicates that MSA uses the progressive multiple alignment strategy to provide the upper bounds for all sequence pairs in the multidimensional dynamic programming on a restricted space. The user may specify these upper bounds, thereby overriding the progressive multiple alignment step.

Insertions and Deletions

Alignment of protein sequences often requires the introduction of indels to maximize the similarity between sequences. There are basically two different methods for scoring indels. The most commonly used method assesses a constant, length-independent penalty (C). The second method charges a length-independent penalty for the initiation of the indel (I) and a length-dependent penalty for extending the indel (E). One of the methods analyzed in this study, TULLA, uses an indel score referred to as the relative gap weight (RGW) that assesses a constant indel penalty relative to how many sequences have this indel. The greater the number of sequences containing a common indel, the higher the penalty.

Parameters

The rate-limiting step of this study has been determining the appropriate user-specified parameters of each method for each data set. The number of user-specified parameters varies from method to method (from one to seven). Often the same parameter is called by different names in different programs. We have adopted a uniform parameter listing throughout this study, therefore, the indel penalty is the gap penalty, C is the constant, length-independent indel penalty, and I+E is the initial, length-independent plus the extension, length-dependent indel

penalty. In the ASSEMBLE program I+E is the “first” and “second” penalty, and in CLUSTAL V it is the “fixed” and “floating” penalty. Word size is called “ k -tuple” in CLUSTAL V and “amino acid residue length” in GENALIGN. The only parameter common to all methods is the indel penalty. In PRALIGN the I+E penalty is only applied to the word size, thus forming part of the motif conservation.

A range of parameter conditions has been explored for each method. Changes that have provided significantly better results, as judged by the motif identification criteria, when substituted for the default parameters, are indicated in tables 2–5. The software developers have also been given the opportunity to improve the results of the test of their methods by altering source code or by suggesting alternative parameter-range combinations. Few suggestions were forthcoming that improved the test results; although, those changes that resulted in improvement have been incorporated into this analysis.

Results

Although the program MSA correctly aligns the set of six globin sequences, it could not be tested further because of space requirements greater than the 40 megabytes of RAM and 40 megabytes+ swap (Lipman et al. 1989). The preliminary program MWT, which is an implementation of the exact algorithm for maximum weight-trace multiple alignment problem, could not

produce results at all with our test sets. We attribute this to the space limitations of our computer (Kececioglu 1993). By using a set of six globins with >50% identity, however, MWT produces the correct alignment (unpublished observation). An implementation of the approximation algorithm for MWT that is space efficient is in progress (J. Kececioglu, personal communication). Future testing will determine whether either MSA or MWT can correctly identify motifs that define a protein family. These two methods will not be considered further.

Our comparative analysis indicates three distinct types of problems in multiple sequence alignment. The most significant problem encountered is the inability to merge subsets of sequences in which motifs have been correctly identified, to provide a single multiple alignment (tables 2–5). The global method GENALIGN and the local method PRALIGN exhibit this problem for all data sets to varying degrees, depending both on the number of sequences and on which specific sequences are analyzed (tables 2–5). In the kinase test, several other methods—ASSEMBLE, CLUSTAL V, MULTAL, TULLA, and PIMA—exhibit this problem to a minor degree. In this case the problem stems from the inability to recognize single-residue motifs that are common between subsets (table 3 and fig. 2).

Both the protease and RH data sets have some motifs that display low motif conservation (e.g., fig. 3, motif II, and fig. 4, motif IV). Most of the methods exhibit varying degrees of inability to merge correctly aligned subsets of sequences from these more distantly related data sets (tables 4 and 5). It should be noted that an additional weighting parameter was developed for DFALIGN (D.-F. Feng and R. F. Doolittle, personal communication) to specifically correct this type of error. This parameter allows the user to specify an additional weight (a value of 2 or 3 is sufficient) to be added to the score for each identical match beginning with a user-specified sequence. For example, in the kinase test set a weight of 2 is added for each identical residue common between sequences beginning with the third sequence. Use of this parameter is absolutely necessary to achieve the scores of tables 3–5 for the DFALIGN program. Extreme caution should be exercised in the manipulation of this parameter even by expert users (R. F. Doolittle, personal communication).

The second problem is the degree to which the number of sequences in the test set affects the ability to recognize motifs. Most methods perform better with larger data sets. In some cases, however, even though the accuracy of identifying motifs increases with the number of sequences, the inability to merge correct subsets of the data set is introduced into the multiple alignment (tables 3–5, comparing sets of 10 vs. 12).

The third problem, sensitivity to specific sequences in the data sets, appears to be a more general problem. One might think that the degree to which a method could identify motifs would not vary significantly as a function of addition or deletion of sister sequences to the data set, but only in the globin test is this problem negligible. Sensitivity to specific sequences is most consistently exhibited by the global methods GENALIGN and AMULT and by the local method PIMA, although all methods suffered to a degree from this problem (tables 2–5).

Discussion

Protein sequences with >50% amino acid residue identity can usually be unambiguously aligned by many of the multiple alignment methods currently available. Among protein sequences with <30% identity, it can be fairly straightforward to find the ordered series of motifs when the motifs are well conserved and when few indels have occurred (table 3 and fig. 2). It is difficult, however, to discern the ordered series of motifs that define a protein family and to obtain an adequate global multiple alignment that can be used in subsequent phylogenetic inference, if the motifs are not well conserved and if significant indels have occurred (tables 4 and 5 and figs. 3 and 4).

We have identified three specific problems that are exhibited to various degrees by all the methods tested. The first, the inability to produce a single multiple alignment, could be due to an indel penalty that is too high. This seems unlikely, since we have varied the indel penalties in most methods without alleviating this problem. The extra parameter of the DFALIGN method, which allows the user to increase the weight for matches as the distance between sequences increases, suggests that the inability to produce a single multiple alignment from subsets could be addressed as a matrix problem. Perhaps identical residues common among distantly related protein sequences should have a higher value, especially if they occur in small contiguous runs. The point, in the divergence of a family of protein sequences, at which such an increase in the values of identities should take precedence over more standard matrix scores needs to be investigated. Currently, subsets are merged by adjusting the placement of indels and appropriately reducing or increasing the number of indels to produce a single multiple alignment as a final manual refinement.

The second problem, the sensitivity to the number of sequences, and the third problem, which specific sequences are in the test set, are serious problems. The increase from 6 sequences to 10 sequences, by the addition of sister sequences to the test data sets, usually increases the ability of most methods to identify motifs. This increase, however, is accompanied by the intro-

duction of the inability to merge correct subsets. The addition of only two more sister sequences to the 10-sequence set, however, causes a decrease in identification of motifs. This effect is most significant for the protease and RH tests (tables 4 and 5). Why so many of the methods are sensitive to sequence number and specificity is an area that warrants further investigation on the part of the software developers. Such shortcomings should warn biologists that variation in data sampling could lead to erroneous conclusions regarding the ordered series of motifs defining a protein family, as well as the phylogenetic history of the gene, when these methods are used.

It is surprising that the global methods perform better than the local methods in the correct identification of the ordered series of motifs present in the four different data sets analyzed (tables 2–5). In addition, methods (global or local) based on the CW approach perform poorly compared with all other methods. In light of these results the biologist-user should exercise caution in the use of local methods or CW methods, either local or global, to infer functional motifs.

It is obvious that a method that can identify an ordered series of motifs, in which individual motifs can vary in both motif density and motif conservation, is just the first stage of obtaining a structural or evolutionarily meaningful multiple protein-sequence alignment. Once this is achieved, the intervening regions of the ordered series of motifs must be aligned. Such an alignment can then be used for phylogenetic reconstruction, for classification of additional sequences, and for determining significantly different subsequences among the sequences that will provide additional information about functional properties, e.g., substrate specificity.

We are interested in the development of multiple alignment approaches that are designed to reconstruct the evolutionary relationships between proteins. Such approaches must not only take into account sequence identity and conservative substitution based on mutational frequencies and physical and chemical similarities of amino acids, but must also be able to describe regions of indels and duplication that can be very useful as phylogenetic markers. Methods that only detect highly conserved motifs, while useful for inferring function, are insufficient for phylogenetic analysis. If all that is detected between proteins are the functionally or structurally constrained residues and if such regions form the basis of phylogenetic reconstruction, then one runs the risk of inferring an incorrect tree topology because of the increased likelihood of parallel or convergent substitutions; this problem can be mitigated by considering sequence information conserved between more closely related relatives.

The area of computational biology that encompasses both sequence-search and alignment algorithms has created a plethora of methods. In only a few instances have developers attempted to evaluate the multiple alignments produced by their methods by comparing them with experimentally determined structures (Barton and Sternberg 1987a, 1987b; Subbiah and Harrison 1989). The field is now sufficiently developed for adequate testing of methods on real sequence data. It is no longer sufficient that algorithm developers merely propose yet another approach to these problems. It is incumbent upon the software developers to specify the limits of new methods on the basis of an adequate sampling of known protein families. Likewise it is the obligation of the analytical biologist to provide well-controlled tests and to suggest further directions for the development of new methods for sequence analysis. Perhaps developers could use the test sequences described here to test new approaches versus older ones. We hope this study not only serves as a guide for multiple protein-sequence methods for biologists, but that it also provides an overview of the problem and a language with which to communicate with the mathematicians, statisticians, and computer scientists in the field. This analysis also provides the algorithm developers with a more informed perspective on the nature of the biological pattern recognition in primary sequences.

The ability to infer the ordered series of motifs that define a protein family is not trivial. While the parameter values utilized in the various methods analyzed in this study may serve as a guide for inferring motifs in other protein sequences, they should in no way be considered as the *parameters* that will always find the motifs. The state-of-the-art strategy for the initial inference of the motifs defining a protein family from primary sequence analysis still requires the combination of multiple alignment methods and human pattern-recognition skills.

Acknowledgments

We would like to thank all the developers who provided their source code and assistance. We are grateful to Mark Boguski, John Kececioglu, George Gutman, and Jacques Perrault for constructive criticisms on the manuscript. Support for M.A.M. and T.K.V. was provided by NIH grant AI 28309. Support for W.M.F. was provided by NSF grant DEB-9096152.

LITERATURE CITED

- ALTSCHUL, S. F., R. J. CARROLL, and D. J. LIPMAN. 1989. Weights for data related by a tree. *J. Mol. Biol.* **207**:647–653.
- BARTON, G. J., and M. J. E. STERNBERG. 1987a. Evaluation and improvements in the automatic alignment of protein sequences. *Protein Eng.* **1**:89–94.

- . 1987b. A strategy for the rapid multiple alignment of protein sequences confidence levels from tertiary structure comparisons. *J. Mol. Biol.* **198**:327–337.
- BASHFORD, D., C. CHOTHIA, and A. M. LESK. 1987. Determinants of a protein fold unique features of the globin amino acid sequences. *J. Mol. Evol.* **196**:199–216.
- CARRILLO, H., and D. LIPMAN. 1988. The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.* **48**: 1073–1082.
- CHAN, S. C., A. K. C. WONG, and D. K. Y. CHIU. 1992. A survey of multiple sequence comparison methods. *Bull. Math. Biol.* **54**:563–598.
- DAVIES, J. F., Z. HOSTOMSKA, Z. HOSTOMSKY, S. R. JORDAN, and D. A. MATTHEWS. 1991. Crystal structure of the ribonuclease H domain of HIV-1 reverse transcriptase. *Science* **252**:88–95.
- DAYHOFF, M. O., R. M. SCHWARTZ, and B. C. ORCUTT. 1978. A model of evolutionary change in proteins. Pp. 345–352 in M. O. DAYHOFF, ed. *Atlas of protein sequence and structure*. National Biomedical Research Foundation, Washington, D.C.
- DOOLITTLE, R. F., D.-F. FENG, M. S. JOHNSON, and M. A. MCCLURE. 1989. Origins and evolutionary relationships of retroviruses. *Q. Rev. Biol.* **64**:1–30.
- FENG, D.-F., and R. F. DOOLITTLE. 1987. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evol.* **25**:351–360.
- FENG, D.-F., M. S. JOHNSON, and R. F. DOOLITTLE. 1985. Aligning amino acid sequences: comparison of commonly used methods. *J. Mol. Evol.* **21**:112–125.
- FITCH, W. M., and E. MARGOLIASH. 1967. Construction of phylogenetic trees. *Science* **155**:279–284.
- GUSFIELD, D. 1993. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bull. Math. Biol.* **55**:141–154.
- HANKS, S. K., and A. M. QUINN. 1991. Protein kinase catalytic domain sequence database: identification of conserved features of primary structure and classification of family members. *Methods Enzymol.* **200**:39–81.
- HIGGINS, D. G., A. J. BLEASBY, and R. FUCHS. 1992. CLUSTAL V: improved software for multiple sequence alignment. *Comput. Appl. Biosci.* **8**:189–191.
- JOHNSON, M. S., M. A. MCCLURE, D.-F. FENG, J. GRAY, and R. F. DOOLITTLE. 1986. Computer analysis of retroviral pol genes: assignment of enzymatic functions. *Proc. Natl. Acad. Sci. USA* **83**:7648–7652.
- KARLIN, S., G. GHANDOUR, F. OST, S. TAVARE, and L. J. KORN. 1983. New approaches for computer analysis of nucleic acid sequences. *Proc. Natl. Acad. Sci. USA* **80**:5660–5664.
- KATAYANAGI, K., M. MIYAGAWA, M. MATSUSHIMA, M. ISHIKAWA, S. KANAYA, M. IKEHARA, T. MATSUZAKI, and K. MORIKAWA. 1990. Three-dimensional structure of ribonuclease H from *E. coli*. *Nature* **347**:306–309.
- KECECIYOLU, J. 1993. The maximum weight trace problem in multiple sequence alignment. Pp. 106–119 in A. APOSTOLICO, M. C. Z. GALIL, and U. MANBER, eds. *The 4th symposium on combinatorial pattern matching*. Springer, Berlin.
- KNIGHTON, D. R., J. ZHENG, L. F. TEN EYCK, V. A. ASHFORD, N.-H. XUONG, S. S. TAYLOR, and J. M. SOWADSKI. 1991. Crystal structure of the catalytic subunit of cyclic adenosine monophosphate-dependent protein kinase. *Science* **254**: 407–414.
- LIPMAN, D. J., S. F. ALTSCHUL, and J. D. KECECIYOLU. 1989. A tool for multiple sequence alignment. *Proc. Natl. Acad. Sci. USA* **86**:4412–4415.
- MCCLURE, M. A. 1992. Sequence analysis of eukaryotic retroviral proteins. *Math. Comput. Modeling Int. J.* **16**:121–136.
- . 1993. Evolutionary history of reverse transcriptase. Pp. 425–444 in A. M. SKALKA and S. P. GOFF, eds. *Reverse transcriptase*. Cold Spring Harbor Laboratory, Cold Spring Harbor, N.Y.
- MARTINEZ, H. M. 1988. A flexible multiple sequence alignment program. *Nucleic Acids Res.* **16**:1683–1691.
- MILLER, M., M. JASKOLSKI, J. K. MOHANA RAO, J. LEIS, and A. WLODAWER. 1989. Crystal structure of a retroviral protease proves relationship to aspartic protease family. *Nature* **337**:576–579.
- MYERS, E. W. 1991. An overview of sequence comparison algorithms in molecular biology. *Tech. rep. TR 91-92*. University of Arizona, Tucson.
- NEEDLEMAN, S. B., and C. D. WUNSCH. 1970. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol.* **48**:443–453.
- PEARL, L. H., and W. R. TAYLOR. 1987. A structural model for the retroviral proteases. *Nature* **329**:351–354.
- PEVZNER, P. 1993. Multiple alignment, communication cost and graph matching. *SIAM J. Appl. Math.* **52**:1763–1779.
- RAO, J. K. M. 1987. New scoring matrix for amino acid residue exchanges based on residue characteristic physical parameters. *Int. J. Pept. Protein Res.* **29**:276–281.
- RISLER, J. L., M. O. DELORME, H. DELACROIX, and A. HENAUT. 1988. Amino acid substitutions in structurally related proteins: a pattern recognition approach: determination of a new and efficient scoring matrix. *J. Mol. Biol.* **204**:1019–1029.
- SCHULER, G. D., S. F. ALTSCHUL, and D. J. LIPMAN. 1991. A workbench for multiple alignment construction and analysis. *Proteins Structure Function Genet.* **9**:180–190.
- SMITH, R. F., and T. F. SMITH. 1990. Automatic generation of primary sequence patterns from sets of related protein sequences. *Proc. Natl. Acad. Sci. USA* **87**:118–122.
- . 1992. Pattern-induced multi-sequence alignment (PIMA) algorithm employing secondary structure-dependent gap penalties for use in comparative protein modeling. *Protein Eng.* **5**:35–41.
- SMITH, T. F., and M. S. WATERMAN. 1981. Identification of common molecular subsequences. *J. Mol. Biol.* **147**:195–197.
- STATES, D. J., and M. S. BOGUSKI. 1990. Similarity and homology. Pp. 89–157 in M. GRIBSKOV and J. DEVEREUX, eds. *Sequence analysis primer*. W. H. Freeman, New York.
- SUBBIAH, S., and S. C. HARRISON. 1989. A method for multiple sequence alignment with gaps. *J. Mol. Biol.* **209**:539–548.

- TANESE, N., and S. P. GOFF. 1988. Domain structure of the Moloney murine leukemia virus reverse transcriptase: mutational analysis and separate expression of the DNA polymerase and RNAase H activities. *Proc. Natl. Acad. Sci. USA* **85**:1777–1781.
- TANG, J., M. N. G. JAMES, I.-N. HSU, J. JENKINS, and T. BLUNDELL. 1978. Structural evidence for gene duplication in the evolution of acid proteases. *Nature* **271**:618–621.
- TAYLOR, W. R. 1986. Identification of protein sequence homology by consensus template alignment. *J. Mol. Biol.* **188**: 233–258.
- . 1987. Multiple sequence alignment by a pairwise algorithm. *Comput. Appl. Biosci.* **3**:81–87.
- . 1988. A flexible method to align large numbers of biological sequences. *J. Mol. Evol.* **28**:161–169.
- VINGRON, M., and P. ARGOS. 1991. Motif recognition and alignment for many sequences by comparison of dotmatrices. *J. Mol. Biol.* **218**:33–43.
- WATERMAN, M. S. 1986. Multiple sequence alignment by consensus. *Nucleic Acids Res.* **14**:9095–9102.
- WATERMAN, M. S., and R. JONES. 1990. Consensus methods for DNA and protein sequence alignment. *Methods Enzymol.* **183**:221–237.
- WATERMAN, M. S., and M. D. PERLWITZ. 1984. Line geometries for sequence comparison. *Bull. Math. Biol.* **46**:567–577.
- WILBUR, W. J., and D. J. LIPMAN. 1982. Rapid similarity searches of nucleic acid and protein data banks. *Proc. Natl. Acad. Sci. USA* **80**:726–730.

STANLEY A. SAWYER, reviewing editor

Received August 16, 1993

Accepted January 5, 1994

STRING 7—recent developments in the integration and prediction of protein interactions

Christian von Mering^{1,2,*}, Lars J. Jensen¹, Michael Kuhn¹, Samuel Chaffron^{1,2},
Tobias Doerks¹, Beate Krüger¹, Berend Snel³ and Peer Bork^{1,4}

¹European Molecular Biology Laboratory, Meyerhofstrasse 1, 69117 Heidelberg, Germany, ²University of Zurich,
Winterthurerstrasse 190, 8057 Zurich, Switzerland, ³Utrecht University, Padualaan 8, 3584 CH Utrecht,
The Netherlands and ⁴Max-Delbrück-Centre for Molecular Medicine, Robert-Rössle-Str. 10, 13092 Berlin, Germany

Received September 15, 2006; Revised and Accepted October 5, 2006

ABSTRACT

Information on protein–protein interactions is still mostly limited to a small number of model organisms, and originates from a wide variety of experimental and computational techniques. The database and online resource STRING generalizes access to protein interaction data, by integrating known and predicted interactions from a variety of sources. The underlying infrastructure includes a consistent body of completely sequenced genomes and exhaustive orthology classifications, based on which interaction evidence is transferred between organisms. Although primarily developed for protein interaction analysis, the resource has also been successfully applied to comparative genomics, phylogenetics and network studies, which are all facilitated by programmatic access to the database backend and the availability of compact download files. As of release 7, STRING has almost doubled to 373 distinct organisms, and contains more than 1.5 million proteins for which associations have been pre-computed. Novel features include AJAX-based web-navigation, inclusion of additional resources such as BioGRID, and detailed protein domain annotation. STRING is available at <http://string.embl.de/>

INTRODUCTION

A fully comprehensive view of all functionally relevant protein interactions is still not available for any species, not even for relatively simple, single-celled model organisms. However, this information is essential for a systems-level

understanding of cellular behavior, and it is needed in order to place the molecular functions of individual proteins into their cellular context.

For detecting direct physical binding between proteins, numerous small-scale and high-throughput experiments have been undertaken, and most of their reported interactions are available from dedicated interaction databases (1–4), as well as from multipurpose databases centered on specific model organisms (5–7). However, the growth of interaction data is severely lagging behind the pace of genome sequencing, so that for most genomes and proteins known to date no interaction data is available. Furthermore, proteins do not only interact physically: indirect associations such as genetic interactions or shared pathway memberships are equally important for a complete understanding of cellular function, but are for the most part not stored in interaction databases. Instead, they are available from a variety of pathway databases (8,9) and from the scientific literature.

The database STRING ('Search Tool for the Retrieval of Interacting Genes/Proteins') aims to collect, predict and unify most types of protein–protein associations, including direct and indirect associations. In order to cover organisms not yet addressed experimentally, STRING runs a set of prediction algorithms (10), and transfers known interactions from model organisms to other species based on predicted orthology of the respective proteins (11). STRING has grown from a purely predictive resource covering mainly prokaryotes (12) to a comprehensive tool integrating protein association information from all domains of life (Figure 1). Each interaction in the database is annotated with a benchmarked numerical confidence score, which can be used to filter the interaction network at any desired stringency. All data in STRING are stored in relational database tables. The interaction information is freely available for download, but download of the entire database content requires a license agreement to prevent redistribution (free for academic users who only access the previous version number).

*To whom correspondence should be addressed. Tel: +41 44 6353147; Fax: +41 44 6356864; Email: mering@molbio.unizh.ch

The authors wish it to be known that, in their opinion, the first two authors should be regarded as joint First Authors

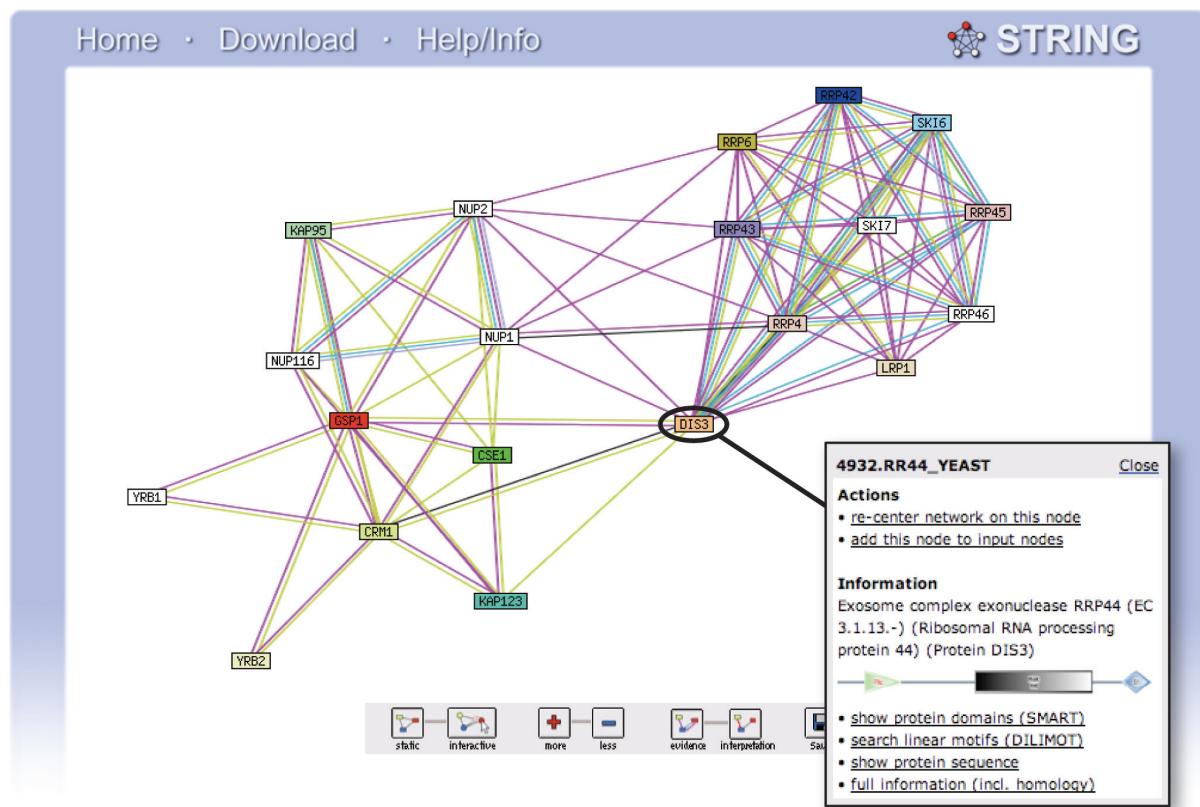


Figure 1. Protein interaction network in STRING. Screenshot from STRING showing a network of *Saccharomyces cerevisiae* proteins [the exosome complex, upper right, is seen weakly associated with proteins from nuclear transport, lower left, see also Ref. (26)]. The inset shows the context menu available for all STRING proteins—in the context menu, annotation and domain architecture are shown directly, and links to other databases and tools are available (22,23). In the network, links between proteins signify the various interaction data supporting the network, colored by evidence type (see STRING website for color legend).

KNOWN AND PREDICTED INTERACTIONS

Known interactions in STRING are primarily imported from existing excellent interaction databases (1–5,8,9), and are complemented by automated text mining of PubMed abstracts and several other bodies of scientific text [such as from Ref. (6)]. As is the case for all interactions in STRING, imported interactions are mapped onto a consistent set of proteins and identifiers, thereby facilitating comparison between datasets. STRING does not store specific details regarding splicing isoforms or post-translational modifications, but instead reduces protein isoforms to a single protein per locus (usually as defined by the longest known protein-coding transcript). This level of resolution enables efficient storage and is compatible with most prediction/transfer algorithms, which usually operate only at the level of the gene locus.

Known interactions are further complemented by *de novo* interaction predictions derived from several comparative genomics prediction algorithms that are mainly applicable to prokaryotes (13–19). These algorithms systematically compare genomes, searching for frequently observed gene neighborhoods, gene fusion events and similarities in gene occurrence across genomes. For each prediction algorithm, dedicated viewers of the genomic evidence are available in STRING.

Interaction evidence from model organisms is often useful for other organisms as well, especially when orthologs

interacting proteins can be clearly identified in the second organism. STRING systematically executes such orthology transfers, using both precomputed orthologs from the COG database (20), as well as a homology-based orthology scheme computed *de novo* (11). STRING can thus immediately predict a large number of interactions for any newly sequenced genome, as soon as it is included into the system. The combination of known, predicted and transferred interactions is unique, making STRING the most comprehensive interaction resource available to date, especially for organisms not addressed experimentally.

The homology data stored in STRING form the basis for the interaction transfers, and are the result of more than 7×10^{11} pairwise protein comparisons using the sensitive Smith–Waterman dynamic programming algorithm. This dataset is a very useful asset in itself [see also (21)], and can be accessed independently of the protein interaction networks by locally installing the STRING database files. Users of the website can also browse all of the homologs detected for any protein of interest, and can inspect alignments with very fast response times (Figure 2).

NEW FEATURES AND IMPROVEMENTS IN STRING 7

The network viewer in STRING (Figure 1) is the central information source and navigation hub for the user. It has

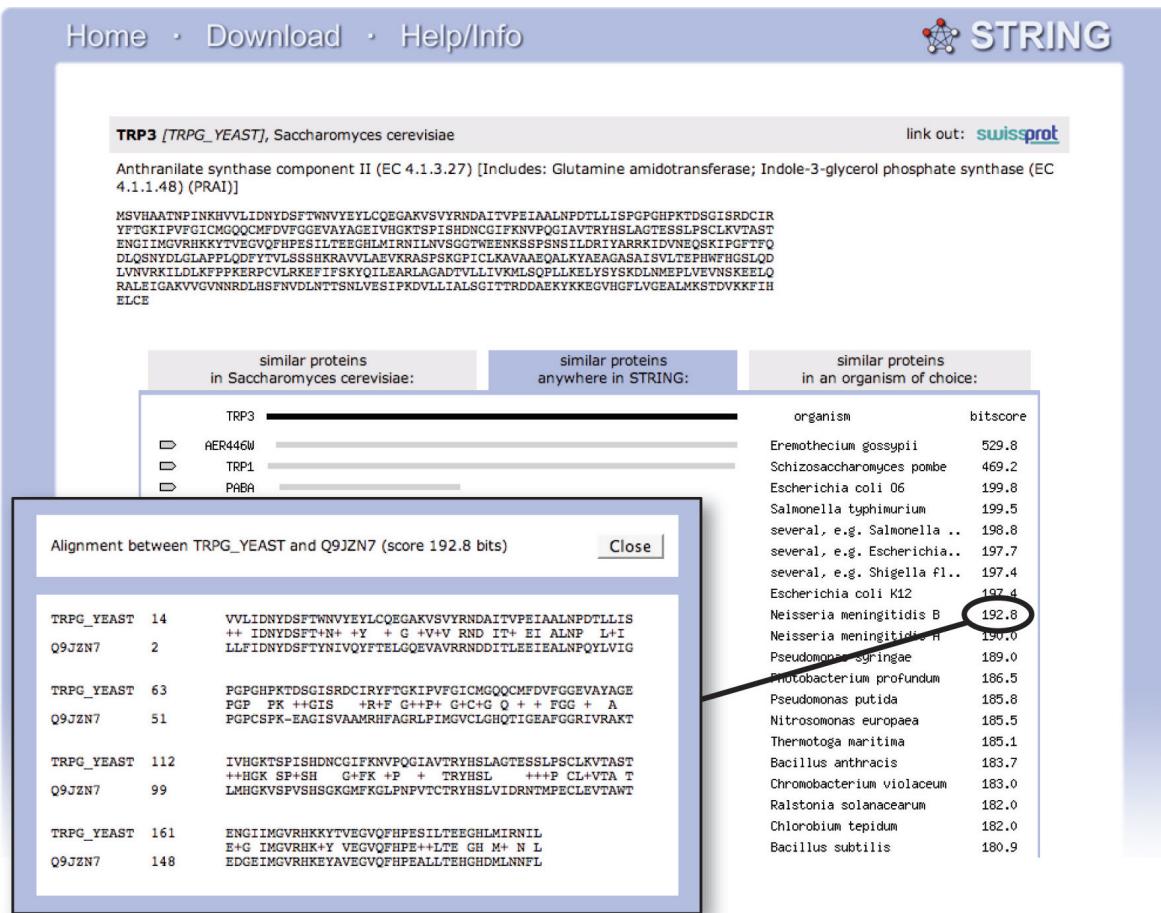


Figure 2. Precomputed homology relations and alignments. For most genomes contained in STRING, sensitive all-against-all homology searches using the Smith–Waterman algorithm are included. These form the basis for assigning orthologs and transferring interaction information, but are also available directly to the user. Because they are stored in a relational database, access to homologs and alignments for any protein of interest is possible without the usual waiting time.

been extended through a context-sensitive menu-box, which displays associated information for any protein in the network. This menu includes a graphical summary of protein domains and features, and allows the user to link out to other external resources such as the motif discovery tool DILIMOT (22). STRING is now also tightly integrated with the SMART protein architecture research tool (23). With the latter it shares a common set of genomes and proteins, for which consistent results are pre-computed and stored. This enables automatic interlinking between both resources (SMART includes interaction previews, and STRING includes domain architecture previews). The topology and evolution of interaction networks can thus be studied both at the level of proteins as well as at the level of individual domains.

Since the last update (11), STRING has grown substantially both in terms of data sources and number of organisms covered. Five new databases are included [MINT, HPRD, BioGRID, DIP and Reactome (2–5,8)], as well as 194 new organisms. Especially due to this latter increase in completely sequenced organisms, the architecture of STRING had to be substantially upgraded so that it can accommodate present and future growth. With respect to the user interface, this required changes in the viewers for the genomic context

data, which could no longer show all of the genomes simultaneously by default. Instead, STRING uses a phylogenetic tree of species to collapse redundant genomes; this tree has been derived from concatenated alignments of a small number of universal protein families (24). Users can navigate the tree by expanding or collapsing its sub-branches, thus choosing which organisms to focus on. AJAX technology ('Asynchronous JavaScript and XML') is then used to fetch the requested information into the existing, pre-loaded browser page, thus increasing usability and speed.

With respect to the underlying database structure, changes were necessary in the way homology data and interaction transfers are stored. Both can no longer be computed and stored in an 'all-against-all' fashion, because of their quadratic scaling with the number of genomes. Beginning with version 7, STRING therefore adopts a two-layered approach when accommodating fully sequenced genomes (Figure 3): important model organisms and those for which experimental data are available from the 'core genomes', all other genomes form the periphery. Within the core, homology searches and interaction transfers are still executed in an all-against-all fashion, whereas for peripheral genomes only searches against the core are included. These and other changes in STRING dramatically improve the scalability of the resource,

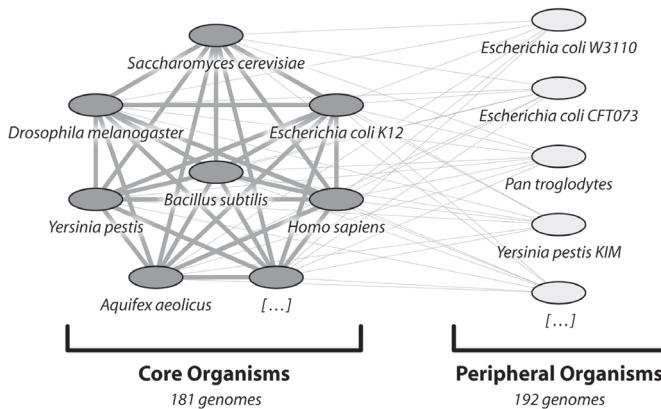


Figure 3. Organisms covered by STRING. STRING currently contains 373 fully sequenced organisms. These are divided into ‘Core Organisms’ and ‘Peripheral Organisms’. The former include all important model organisms for which experimental data are available, as well as selected representatives for cases of redundant genome sequencing (e.g. when several closely related strains of a bacterial species have been sequenced, only one strain is included). The ‘Peripheral Organisms’ form the remainder; they tend to be somewhat redundant, and usually have little more than genomic sequence information annotated. For the core organisms, homology relations and interaction transfers are fully computed, whereas the peripheral organisms are only connected to the core but not among themselves (the graphic shows only a small selection of organisms; lines indicate homology searches and interaction transfers). This architecture allows STRING to encompass all sequenced genomes, while still keeping database size and computation time within reasonable limits.

leading to faster update cycles even when the number of sequenced genomes is to increase as fast as currently projected. Together with future plans to increase the scope and specificity of the stored interaction information, STRING should thus continue to facilitate not only network research but also wider projects that range from phylogenetics to metagenomics (24,25).

ACKNOWLEDGEMENTS

The authors wish to thank Dianna Fisk from the *Saccharomyces* Genome Database for access to the Gene Summary Paragraphs, and Toby Gibson, Martijn Huynen, Victor Neduvia, Rune Linding and members of the Bork group for continued feedback and discussions. This work was supported in part by grants from the Bundesministerium für Forschung und Bildung, Germany, as well as through the ADIT Integrated Project, contract number LSHB-CT-2005-511065, and through the BioSapiens Network of Excellence, contract number LSHG-CT-2003-503265, both funded by the European Commission FP6 Programme. Funding to pay the Open Access publication charges for this article was provided by the University of Zurich, through its Research Priority Program ‘Systems Biology and Functional Genomics’.

Conflict of interest statement. None declared.

REFERENCES

- Alfaroan,C., Andrade,C.E., Anthony,K., Bahroos,N., Bajec,M., Bantoft,K., Betel,D., Bobechko,B., Boutilier,K., Burgess,E. *et al.* (2005) The biomolecular interaction network database and related tools 2005 update. *Nucleic Acids Res.*, **33**, D418–D424.
- Salwinski,L., Miller,C.S., Smith,A.J., Pettit,F.K., Bowie,J.U. and Eisenberg,D. (2004) The Database of Interacting Proteins: 2004 update. *Nucleic Acids Res.*, **32**, D449–D451.
- Zanzoni,A., Montecchi-Palazzi,L., Quondamatteo,F., Ausiello,G., Helmer-Citterich,M. and Cesareni,G. (2002) MINT: a Molecular INTeraction database. *FEBS Lett.*, **513**, 135–140.
- Stark,C., Breitkreutz,B.J., Reguly,T., Boucher,L., Breitkreutz,A. and Tyers,M. (2006) BioGRID: a general repository for interaction datasets. *Nucleic Acids Res.*, **34**, D535–D539.
- Mishra,G.R., Suresh,M., Kumaran,K., Kannabiran,N., Suresh,S.., Bala,P., Shivakumar,K., Anuradha,N., Reddy,R., Raghavan,T.M. *et al.* (2006) Human protein reference database—2006 update. *Nucleic Acids Res.*, **34**, D411–D414.
- Hirschman,J.E., Balakrishnan,R., Christie,K.R., Costanzo,M.C., Dwight,S.S., Engel,S.R., Fisk,D.G., Hong,E.L., Livstone,M.S., Nash,R. *et al.* (2006) Genome Snapshot: a new resource at the *Saccharomyces* Genome Database (SGD) presenting an overview of the *Saccharomyces cerevisiae* genome. *Nucleic Acids Res.*, **34**, D442–D445.
- Schwarz,E.M., Antoshechkin,I., Bastiani,C., Bieri,T., Blasius,D., Canaran,P., Chan,J., Chen,N., Chen,W.J., Davis,P. *et al.* (2006) WormBase: better software, richer content. *Nucleic Acids Res.*, **34**, D475–D478.
- Joshi-Tope,G., Gillespie,M., Vaastrik,I., D'Eustachio,P., Schmidt,E., de Bono,B., Jassal,B., Gopinath,G.R., Wu,G.R., Matthews,L. *et al.* (2005) Reactome: a knowledgebase of biological pathways. *Nucleic Acids Res.*, **33**, D428–D432.
- Kanehisa,M., Goto,S., Hattori,M., Aoki-Kinoshita,K.F., Itoh,M., Kawashima,S., Katayama,T., Araki,M. and Hirakawa,M. (2006) From genomics to chemical genomics: new developments in KEGG. *Nucleic Acids Res.*, **34**, D354–D357.
- von Mering,C., Huynen,M., Jaeggi,D., Schmidt,S., Bork,P. and Snel,B. (2003) STRING: a database of predicted functional associations between proteins. *Nucleic Acids Res.*, **31**, 258–261.
- von Mering,C., Jensen,L.J., Snel,B., Hooper,S.D., Krupp,M., Foglierini,M., Jouffre,N., Huynen,M.A. and Bork,P. (2005) STRING: known and predicted protein–protein associations, integrated and transferred across organisms. *Nucleic Acids Res.*, **33**, D433–D437.
- Snel,B., Lehmann,G., Bork,P. and Huynen,M.A. (2000) STRING: a web-server to retrieve and display the repeatedly occurring neighbourhood of a gene. *Nucleic Acids Res.*, **28**, 3442–3444.
- Valencia,A. and Pazos,F. (2002) Computational methods for the prediction of protein interactions. *Curr. Opin. Struct. Biol.*, **12**, 368–373.
- Huynen,M.A. and Bork,P. (1998) Measuring genome evolution. *Proc. Natl Acad. Sci. USA*, **95**, 5849–5856.
- Pellegrini,M., Marcotte,E.M., Thompson,M.J., Eisenberg,D. and Yeates,T.O. (1999) Assigning protein functions by comparative genome analysis: protein phylogenetic profiles. *Proc. Natl Acad. Sci. USA*, **96**, 4285–4288.
- Enright,A.J., Iliopoulos,I., Kyriakis,N.C. and Ouzounis,C.A. (1999) Protein interaction maps for complete genomes based on gene fusion events. *Nature*, **402**, 86–90.
- Marcotte,E.M., Pellegrini,M., Ng,H.L., Rice,D.W., Yeates,T.O. and Eisenberg,D. (1999) Detecting protein function and protein–protein interactions from genome sequences. *Science*, **285**, 751–753.
- Dandekar,T., Snel,B., Huynen,M. and Bork,P. (1998) Conservation of gene order: a fingerprint of proteins that physically interact. *Trends Biochem. Sci.*, **23**, 324–328.
- Overbeek,R., Fonstein,M., D’Souza,M., Pusch,G.D. and Maltsev,N. (1999) The use of gene clusters to infer functional coupling. *Proc. Natl Acad. Sci. USA*, **96**, 2896–2901.
- Tatusov,R.L., Fedorova,N.D., Jackson,J.D., Jacobs,A.R., Kiryutin,B., Koonin,E.V., Krylov,D.M., Mazumder,R., Mekhedov,S.L., Nikolskaya,A.N. *et al.* (2003) The COG database: an updated version includes eukaryotes. *BMC Bioinformatics*, **4**, 41.
- Rattei,T., Arnold,R., Tischler,P., Lindner,D., Stumpflen,V. and Mewes,H.W. (2006) SIMAP: the similarity matrix of proteins. *Nucleic Acids Res.*, **34**, D252–D256.
- Neduvia,V. and Russell,R.B. (2006) DILIMOT: discovery of linear motifs in proteins. *Nucleic Acids Res.*, **34**, W350–W355.
- Letunic,I., Copley,R.R., Pils,B., Pinkert,S., Schultz,J. and Bork,P. (2006) SMART 5: domains in the context of genomes and networks. *Nucleic Acids Res.*, **34**, D257–D260.

24. Ciccarelli,F.D., Doerks,T., von Mering,C., Creevey,C.J., Snel,B. and Bork,P. (2006) Toward automatic reconstruction of a highly resolved tree of life. *Science*, **311**, 1283–1287.
25. Tringe,S.G., von Mering,C., Kobayashi,A., Salamov,A.A., Chen,K., Chang,H.W., Podar,M., Short,J.M., Mathur,E.J., Detter,J.C. *et al.* (2005) Comparative metagenomics of microbial communities. *Science*, **308**, 554–557.
26. Dimaano,C. and Ullman,K.S. (2004) Nucleocytoplasmic transport: integrating mRNA production and turnover with export through the nuclear pore. *Mol. Cell. Biol.*, **24**, 3069–3076.

WEEK 9 READINGS

WEEK 10 READINGS

RESEARCH

Open Access



CrossMark

The genome of the Gulf pipefish enables understanding of evolutionary innovations

C. M. Small^{1†}, S. Bassham^{1†}, J. Catchen^{1,2†}, A. Amores³, A. M. Fuiten¹, R. S. Brown^{1,4}, A. G. Jones⁵ and W. A. Cresko^{1*}

Abstract

Background: Evolutionary origins of derived morphologies ultimately stem from changes in protein structure, gene regulation, and gene content. A well-assembled, annotated reference genome is a central resource for pursuing these molecular phenomena underlying phenotypic evolution. We explored the genome of the Gulf pipefish (*Syngnathus scovelli*), which belongs to family Syngnathidae (pipefishes, seahorses, and seadragons). These fishes have dramatically derived bodies and a remarkable novelty among vertebrates, the male brood pouch.

Results: We produce a reference genome, condensed into chromosomes, for the Gulf pipefish. Gene losses and other changes have occurred in pipefish *hox* and *dlx* clusters and in the *tbx* and *pitx* gene families, candidate mechanisms for the evolution of syngnathid traits, including an elongated axis and the loss of ribs, pelvic fins, and teeth. We measure gene expression changes in pregnant versus non-pregnant brood pouch tissue and characterize the genomic organization of duplicated metalloprotease genes (*patristacins*) recruited into the function of this novel structure. Phylogenetic inference using ultraconserved sequences provides an alternative hypothesis for the relationship between orders Syngnathiformes and Scombriformes. Comparisons of chromosome structure among percomorphs show that chromosome number in a pipefish ancestor became reduced via chromosomal fusions.

Conclusions: The collected findings from this first syngnathid reference genome open a window into the genomic underpinnings of highly derived morphologies, demonstrating that de novo production of high quality and useful reference genomes is within reach of even small research groups.

Keywords: *Syngnathus scovelli*, Syngnathidae, Male pregnancy, Genome assembly, Evolution, Differential expression, Gene loss, Novel traits

Background

Evolutionary novelties adorn the tree of life and yet their genetic origins remain a problem for biologists. The Modern Synthesis sparsely addressed novel traits but rationalized their incidence with neo-Darwinian models of gradual change via accumulation of many small-effect mutations [1]. Contemporary perspectives are more accepting of discontinuous morphological change [2], underlain by genetic changes diverse in nature. These changes may include point mutations as well as gross changes like gains and losses of genes or their regulatory elements, but the common thread is their effect on developmental systems. Indeed, the origin of novelties is

now routinely viewed through the lens of evolutionary developmental biology, with an emphasis on how gene regulatory networks arise de novo or are modified from ancient ones [3] to orchestrate novel gene expression in development [4].

This modern genetic and developmental understanding of novel traits is an extremely difficult objective without quality genomic resources. Past genome sequencing efforts have been the purview of large, well-populated research communities generally focused on producing a resource beneficial for biomedical research. In the midst of the current sequencing technology revolution, however, the door is open for small research groups to produce genome resources for a variety of other questions, including those in ecology, conservation biology, evolutionary biology, and population genomics. As new evolutionary lineages are sampled, a valuable by-product is that novel reference genomes can augment

* Correspondence: wcresco@uoregon.edu

†Equal contributors

¹Institute of Ecology and Evolution, University of Oregon, Eugene, OR 97403, USA

Full list of author information is available at the end of the article

the study of other existing model genomes, in the way the spotted gar (*Lepisosteus oculatus*) genome aids in bridging between the tetrapod and teleost model organisms [5]. We set out to genomically enable the study of novel body plan and reproductive character evolution in syngnathid fishes (pipefishes, seahorses, and seadragons) by generating a high-quality reference genome for the Gulf pipefish, *Syngnathus scovelli*.

Syngnathid fishes are widely recognized for their highly divergent body plans [6–8], including the elongate form of many pipefishes (Fig. 1), the upright body axis and reduced craniocervical angle of seahorses, and the highly cryptic morphology of the seadragons. Derived characters such as leafy appendages, prehensile tails, and bony body armor are common across the family and, in many cases, have evolved independently in multiple lineages [6, 8, 9]. A truly striking evolutionary innovation shared by all syngnathid fishes is the somatic brooding of offspring by males, crowned by those lineages that have evolved complex, pouch-like structures for the maintenance of homeostasis during pregnancy [10–13]. In total, these remarkable characters make syngnathids an exceptional clade for the study of evolutionary novelty. The Gulf pipefish represents the group well, given its recent history as a choice subject for evolutionary genetic and behavioral studies [14–17], its abundance and amenability to experimental work, and its embodiment of many of the derived syngnathid traits.

Comparative genomics and evolutionary developmental approaches to effectively study the evolution of new forms, such as the diversification of the syngnathid body

plan or the origin of male pregnancy, require advanced genomic tools. The centerpiece of each toolkit is a properly assembled, well annotated genome model, which can be directly compared at the sequence and structural levels to other species and efficiently mined to design molecular tools for manipulative genetic studies. To this end, we produced an annotated chromosome-level genome model [5] for *S. scovelli* by integrating a 176X-coverage, short-read genome assembly with a linkage map constructed from RAD-seq markers. We used this tool to reveal features of chromosome structure evolution, to investigate pipefish lineage-specific losses of genes associated with morphological development, to infer the likely phylogenetic position of the syngnathids in the tree of ray-finned fishes, and to describe a unique cluster of tandemly duplicated *patristacins* [18] that demonstrate conspicuous expression changes in the brood pouch during male pregnancy. Others have reviewed the approaches best suited to small-scale genome projects [19], but our intention here is to provide a biological case study and methodological template for success, motivated by the desire to better understand how novelties arise. We expect our experiences to be of interest to similarly sized research groups ready to reap the benefits of a reference genome in their own pursuits of biological discovery.

Results

The pipefish genome assembly is of high quality and completeness

The only published estimate of Gulf pipefish genome size is based on Feulgen staining [20], from which a

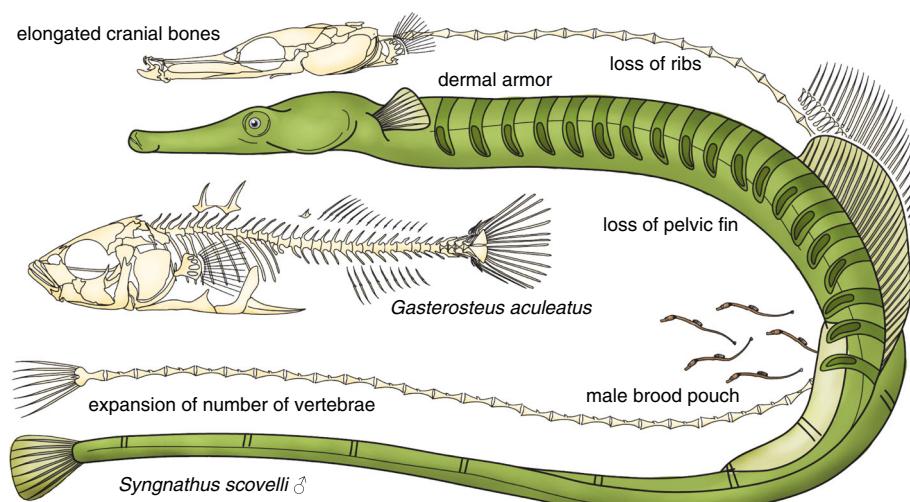


Fig. 1 A cartoon representation of key derived traits in pipefishes and their relatives. Syngnathid fishes such as the Gulf pipefish have increased numbers of vertebrae and an elongated head, are missing pelvic fins and ribs, and have an evolutionarily novel structure, the male brood pouch. Shown for comparison is the axial skeleton of a percomorph with more typical morphology, a threespine stickleback. Note that not all derived syngnathid skeletal features are depicted in this cartoon. For detailed, anatomical illustrations of syngnathid skeleton attributes, please see other studies [144, 145]

haploid genome size of 523.23 Mb was calculated for the species. We obtained a short read k-mer-based genome length estimate of 351.44 Mb using ALLPATHS-LG [21]. Using the RAD markers from our genetic map to estimate the number of RAD sites per scaffold and infer the amount of sequence missing from the assembly by estimating the number of missing RAD sites, we obtained an estimated genome size of 334 Mb. These data suggest that, consistent with the k-mer-based estimate, no more than approximately 27 Mb, or 8% of sequence, is missing from the assembly (not including repetitive sequence) and that the Feulgen estimate is likely too large.

We assembled overlapping and mate-pair Illumina paired-end 100 nt reads (176X total coverage of 351 Mb) into 2123 scaffolds, yielding an assembly length of 307.02 Mb with 6.58% gaps. Contig and scaffold N50 were 32.24 kb and 640.41 kb, respectively, and the maximum scaffold size was 6.71 Mb. An analysis of core eukaryotic genes (CEGs) using CEGMA [22] revealed that our assembly contained complete information for 245 of 248 CEGs and “partial” information for the remaining three CEGs. These assembly quality metrics are comparable to other recently published, high-quality, scaffold-level genomes for fishes. Table 1 presents a side-by-side comparison of the Gulf pipefish assembly with several other published ray-finned fish assemblies.

The genome assembly of *S. scovelli* is comparable in quality to three recently published fish reference genomes. Shown in Table 1 are assembly statistics calculated from scaffold-level genome assemblies, considering scaffolds 1000 nt and longer, except for the 248-gene CEGMA analysis, which was applied to all scaffolds. Assembly versions are *N. furzeri* GCA_000878545.1 [23], *A. mexicanus* GCA_000372685.1 [24], and *L. oculatus* GCF_000242695.1 [5]

Using MAKER [25], we initially generated 37,696 total protein-coding gene annotations, but we retained only 20,834 of these based on biological evidence from protein databases, RNA-sequencing (RNA-seq) data, or protein domain detection. After manual annotation correction for several genes of interest, the final annotation included 20,841 protein-coding genes. Mean and

median protein sequence length were 539.55 and 386.00 amino acids, respectively.

A genetic map integrates 87% of the genome assembly into chromosomes

To order and orient scaffolds and to unite them into chromosomes, we generated an F1 pseudo-test cross genetic linkage map from a cross of wild *S. scovelli* with 108 progeny. Of 21,680 RAD tags, 4779 polymorphic tags were informative and met our criteria for inclusion in the genetic map (see “Methods”). The genetic map readily coalesced into 22 distinct linkage groups (see Additional file 1: Figure S1 for schematics of the consensus genetic map). Markers could be aligned to 553 scaffolds, thereby tying nearly 266.3 Mb – 87% – to chromosome models (see Additional file 2: SH1, which tabulates markers and scaffolds in the map). A total of 271 scaffolds (49%) were anchored at more than one map position with two or more markers, which allowed us to assign an orientation. Unplaced scaffolds tended to be shorter and more depauperate of annotated genes, on average, than scaffolds incorporated into chromosomes (see Additional file 1: Figure S2 for plotted lengths and gene densities of the scaffolds). Possibly the same sequence characteristics that make assembly difficult – a higher occurrence of repetitive DNA – could help explain the lower gene density of these smaller scaffolds. There were few initial conflicts between the genome assembly and the linkage map and none that could not be ruled out as artefactual due to poor support. For instance, three scaffolds were initially tied to more than one linkage group; in all three cases, however, only a single marker, with equivalent alignments to multiple locations, created this conflict and could be reasonably ruled incorrect, particularly when patterns of conserved synteny were taken into account. There were also apparent within-linkage group conflicts, which in most cases could be resolved by movement of markers without any cost to the linkage map. In total, five scaffolds where conflicts remained were split by our software Chromonomer (see “Methods”) to reconcile the map and the assembly; in each of these cases, a small scaffold (1.2 to 3.1 kb) was inserted into a gap in a larger scaffold. Only

Table 1 Scaffold-level assembly statistics for the Gulf pipefish genome

Genome	Scaffolds (n)	Longest scaffold	Scaffold N50	Contig N50	Assembly length	Gaps in assembly (%)	CEGs complete (%)
Gulf pipefish (<i>Syngnathus scovelli</i>)	2104	6.7 Mb	640.4 kb	32.2 kb	307.0 Mb	6.6	98.8
African turquoise killifish (<i>Nothobranchius furzeri</i>)	29,054	0.7 Mb	119.7 kb	8.7 kb	1010.9 Mb	7.7	94.8
Blind cave fish (<i>Astyanax mexicanus</i>)	10,542	9.8 Mb	1775.3 kb	14.7 kb	1191.1 Mb	19.1	87.9
Spotted gar (<i>Lepisosteus oculatus</i>)	2105	21.3 Mb	6928.1 kb	68.3 kb	945.8 Mb	8.1	90.7

the largest of these small scaffolds contained an annotated gene, and in that case, its insertion into the larger scaffold agreed with the relative position of its ortholog in other teleost genomes.

Chromosome evolution is revealed by patterns of conserved synteny

Evidence based on ancestral state reconstruction supports an ancestral chromosome number of 24 in the teleosts [26]. Though chromosome number has been shown to vary across the broad group of Syngnathidae, the 22 linkage groups that coalesced in this linkage map in *S. scovelli* accords well with published karyotypes for two other species in *Syngnathus*, *S. abaster*, and *S. typhle* [27]. Using a genome-wide synteny analysis, we investigated how this change from the ancestral chromosome number likely occurred. Genes are called syntenic when they lie on the same chromosome or chromosomal segment and a pair of compared genomes show “conserved synteny” when orthologous genes that are syntenic in one genome also lie together, though not necessarily in the same gene order, in the comparator genome. The pattern of conserved synteny between Gulf pipefish and other teleosts, such as southern platyfish (*Xiphophorus maculatus*), which has the ancestral number of chromosomes (Fig. 2a), suggests that the reduced chromosome number in *Syngnathus* resulted simply from two chromosomal fusions (Fig. 2b). Two large blocks covering the length of one linkage group in *S. scovelli* have strong conserved synteny of orthologs along both platyfish LG 1 and 24, respectively, and another pair of blocks covering all of a second pipefish linkage group are orthologous to platyfish LG 14 and 23 (Fig. 2b). The resulting pipefish chromosomes, which we here name LG 1 and 14 to reflect this orthology, are the largest in the genome. Several scaffolds linked to pipefish LG1 and LG14 contain genes orthologous to the two ancestral chromosomes that constitute each of them (Fig. 2b), suggesting that intrachromosomal rearrangements have blended the original margins of the chromosomes since they became fused.

Other within-chromosome rearrangements relative to various teleost reference genomes can be confidently inferred using the pipefish assembly and linkage map, where they provide mutual support. It is beyond the scope of this paper to catalogue such chromosomal differences and is the subject of other studies. As an example, however, pipefish LG 16 can be used to illustrate a subset of these rearrangements because all scaffolds that map to this linkage group are ordered and all but two very small scaffolds are oriented, with strong map support. Here, likely inversions and transpositions can be discerned in a comparison between pipefish and platyfish, based on stretches of conserved synteny of protein coding genes (Fig. 2c).

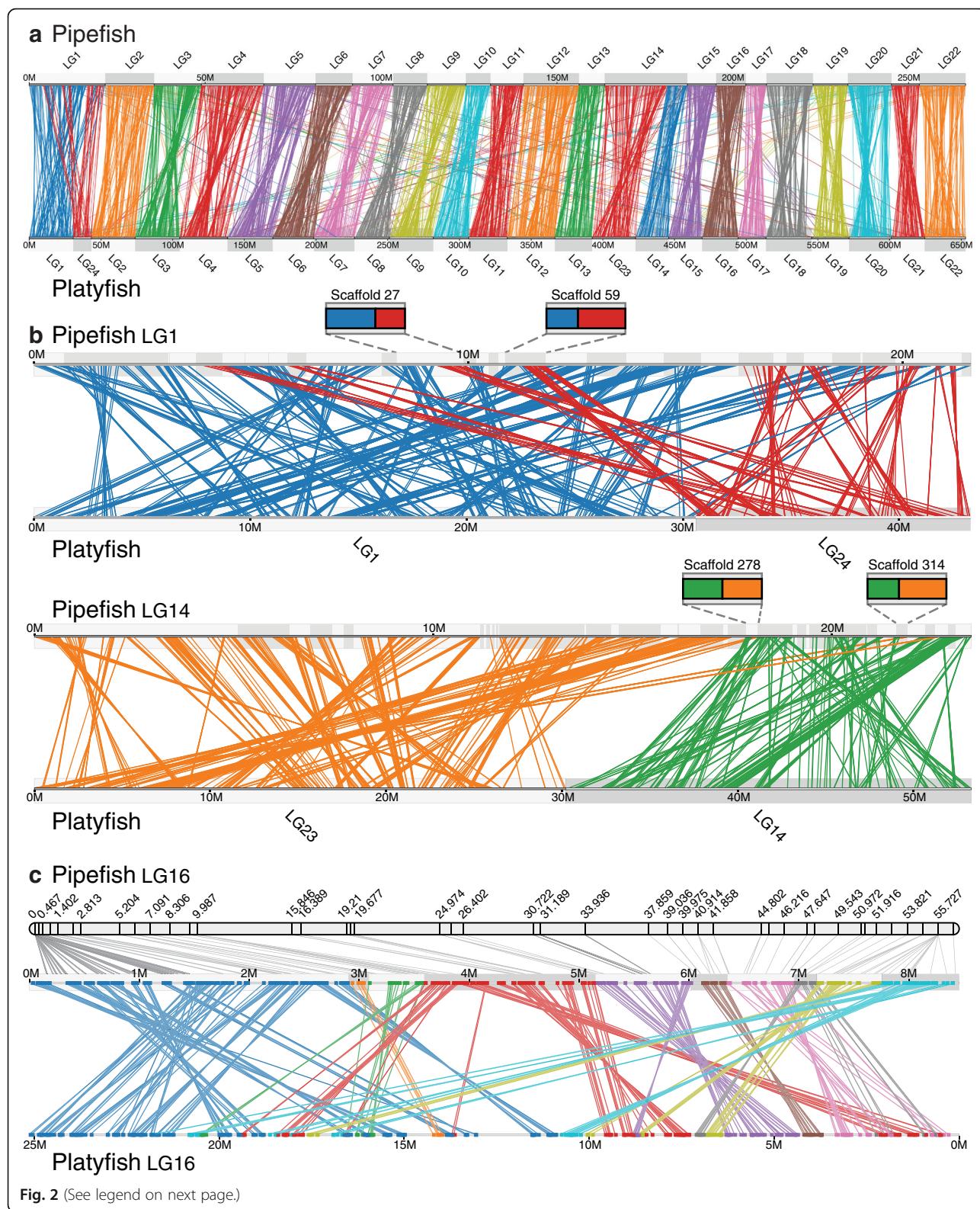
Phylogenomic analysis supports an alternative hypothesis for the position of syngnathiform fishes among the Percomorpha

Knowing the phylogenetic placement of syngnathid fishes relative to other teleosts with sequenced genomes is critical for using comparative genomic approaches to polarize the evolution of traits in the Syngnathidae. Conflicting hypotheses regarding the origin of syngnathid fishes and their relatives are a barrier to this understanding, and resolving phylogenetic relationships for the crown clade of teleosts (Superorder Percomorpha) in general has been a problem [28–30].

Ultraconserved elements (UCEs) offer a genome-wide alternative to small panels of nuclear and mitochondrial phylogenetic markers because they exist by the hundreds or thousands in vertebrate genomes, are often easily identifiable as well-conserved, single-copy orthologs that contain divergent regions, and can be used to address hypotheses over a broad range of phylogenetic scales [31]. Faircloth et al. [32] used UCEs to produce a well-supported phylogeny at both deep and shallow time scales for ray-finned fishes. We added to this dataset UCEs from Gulf pipefish, Pacific bluefin tuna (*Thunnus orientalis*), and southern platyfish and performed phylogenetic analysis. Interestingly, our phylogenomic analysis provides an alternative hypothesis regarding the relationships among Scombriformes (tunas and their relatives) and Syngnathiformes (Syngnathid fishes and their relatives). Briefly, the two orders would not be interpreted as a monophyletic clade from our topology, in contrast to conclusions based on trees inferred by others [29, 30, 33]. Statistical support for clades bracketing this region of the topology was high (Fig. 3), but should be interpreted with caution given evidence that phylogenetic discordance across different regions of the genome can limit the accuracy of species-level inferences based on concatenated sequence data [34, 35]. We recovered all relationships reported by Faircloth et al. [32] and found, consistent with previous studies [29, 30, 33], that the Syngnathiformes are not nested within the clade containing species commonly used in genetic and genomic studies (i.e. medaka, platyfish, stickleback, and pufferfish). Given this phylogenetic hypothesis for the origin of syngnathids, the Gulf pipefish genome fills a useful outgroup role in comparative genomics studies using these model species. The currently understood relationships also highlight a need for phylogenetic analyses including fish lineages that diverged just prior to origin of the syngnathids, in order to help understand the unusual derived traits in the Syngnathidae.

Convergent and unique gene losses have occurred in the pipefish *hox* clusters

The *hox* clusters, which include tandem arrays of homeobox genes interspersed with non-coding RNAs



(See figure on previous page.)

Fig. 2 Chromosomal rearrangements inferred from a conserved synteny comparison. **a** Pipefish and platyfish chromosomes are broadly congruent. Strings connecting orthologous genes between the species' genomes are colored by pipefish chromosome. **b** Pipefish LG 1 and 14 are each orthologous to two platyfish chromosomes, likely because chromosome fusions occurred in the syngnathid lineage. Several scaffolds from fused chromosomes 1 and from 14, including those shown in the insets, show blocks of conserved synteny to both "ancestral" chromosomes in platyfish (LG 1 and 24 or LG 14 and 23). This pattern indicates that some number of intra-chromosomal rearrangements blended segments across the chromosomal junction after the chromosomes fused. Strings connecting orthologs are color-coded by platyfish chromosome. Pipefish scaffolds are shown in alternately shaded rectangles along the chromosome. **c** On LG 16, differences in the orientation and location of orthologous gene blocks suggest inversions and transpositions have occurred since the last common ancestor of pipefish and platyfish. Strings connecting orthologous genes are colored according to the pipefish scaffold each gene resides on. Support for scaffold order and orientation can be seen in the linkage map for pipefish LG 16, shown above

that regulate *hox* and other genes, are critical for patterning the body axis and paired appendages (reviewed in [36–38]). Pipefish have elongated bodies, including more trunk and especially more caudal vertebrae than relatives like medaka and threespine stickleback, and they lack pelvic fins, key examples of derived traits depicted in cartoon form in Fig. 1. We therefore scrutinized the gene content of the *hox* clusters for differences from pipefish's percomorph relatives (including

pufferfish, medaka, stickleback, and tuna). Just as in many other gene families, differential loss of *hox* genes among lineages followed the whole genome duplication that occurred near the base of the teleost lineage (e.g. [39]). Gulf pipefish appears to share some of these losses with other percomorph fishes, to the exclusion of the outgroup lineage zebrafish (Fig. 4). A parsimonious interpretation of the pattern of losses suggests that *hoxb10a*, *hoxb8b*, *hoxd13a*, the entire *hoxcb* cluster, and

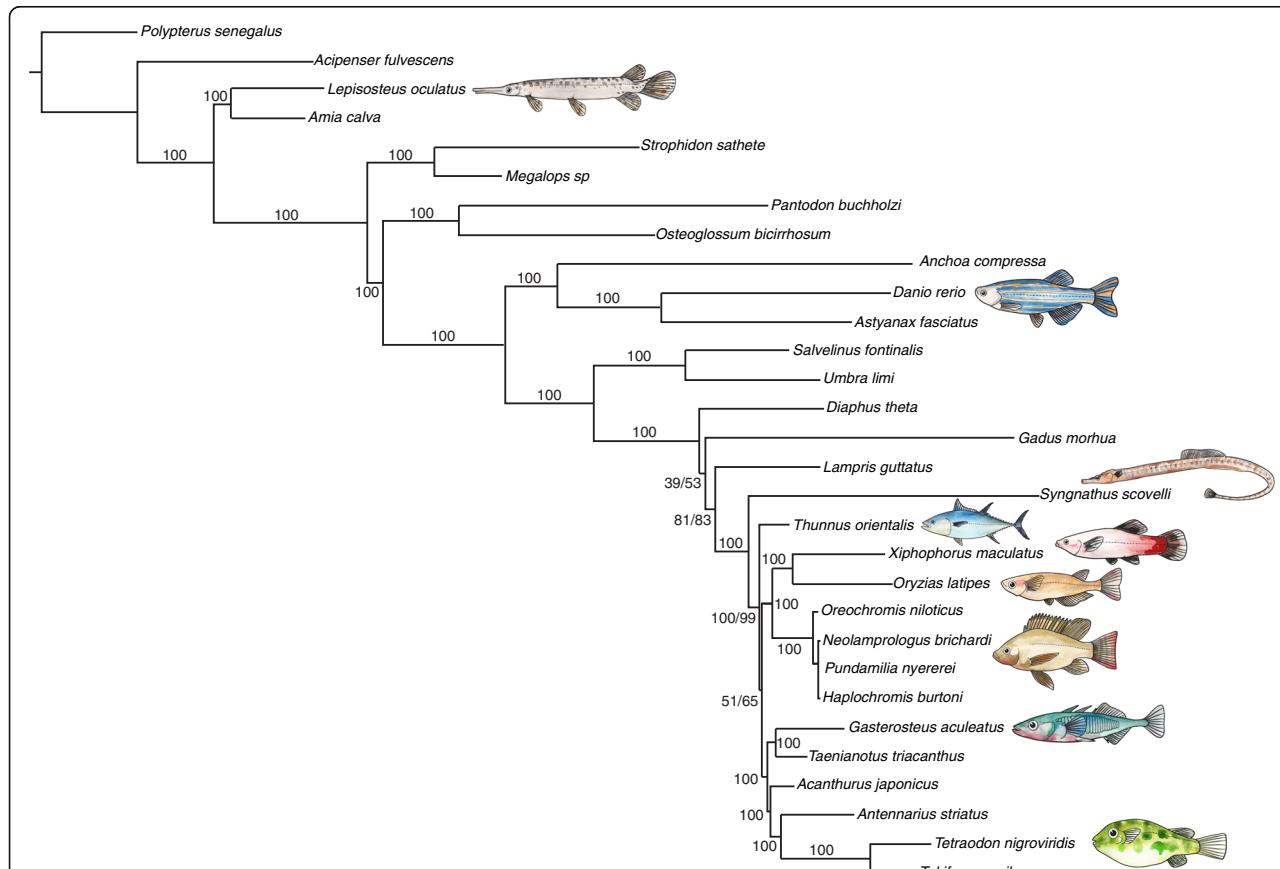


Fig. 3 Phylogenomic inference supports a syngnathiform clade distinct from the clade containing commonly studied fish models. A well-supported maximum likelihood tree of UCEs places Syngnathiformes as an outgroup relative to fellow percomorph species used as genetic models, consistent with previous work regarding the molecular systematics of Percomorpha [29, 30, 33]. Note, however, that our topology is not consistent with a monophyletic group including Syngnathiformes and Scombriformes, as previously reported. Bootstrap and SH-aLRT support is listed for each node; a single number is listed where both values agree

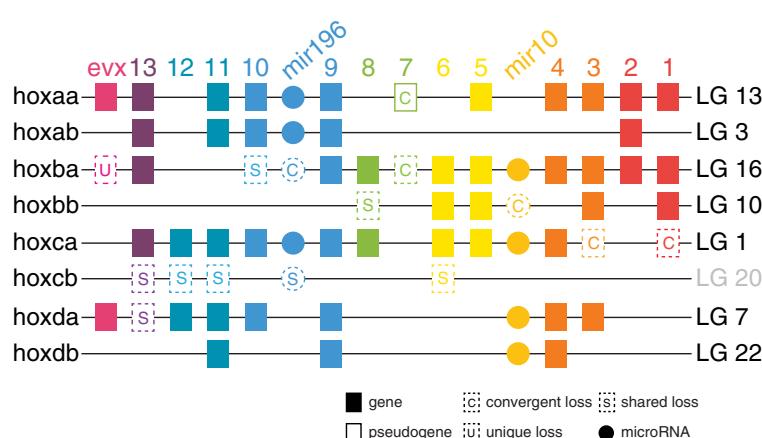


Fig. 4 The pipefish *hox* clusters have experienced convergent and unique gene losses. A cartoon of the *hox* clusters in *S. scovelli*, with boxes representing genes arranged along chromosome segments of different linkage groups, summarizes gene content changes relative to other teleosts. Seven gene losses, of both coding and non-coding genes, are here labeled shared losses among the compared percomorph lineages because these genes are retained by the non-percomorph outgroup, zebrafish. Six other pipefish gene losses are inferred to be convergent losses with respect to some members of Percomorpha because other species that are not pipefish sister lineages have also lost these genes. *hox* cluster-associated *evenskipped* gene *eve1* (a member of the *evx* paralogy group) is missing in pipefish, a loss that has not been reported in other teleosts. Though percomorphs likely share the loss of the *hoxcb* cluster, comparison via conserved synteny with zebrafish shows that the orthologous region is on pipefish LG 20

mir196c were absent in the common ancestor of pipefish and other percomorphs. Several other *hox* cluster genes have been lost in pipefish as well as in some but not all model percomorphs; based on the topology of the phylogenetic tree in Fig. 3 and those inferred by others [29, 30, 33], we conclude that these losses are likely to be convergent (Fig. 4). These include *hoxa7a*, *hoxb7a*, *hoxc3a*, *hoxc1a*, *mir196b* in the *hoxba* cluster, and *mir10a* in the *hoxbb* cluster. For example, *hoxb7a* was likely lost independently at least three times (in pufferfish, medaka, and pipefish), but it is still present in stickleback and tuna. *hoxa7a* was lost independently in both pipefish and pufferfish, leaving both lineages with no *hox7* paralog in any cluster. By contrast, zebrafish and all of the other percomorphs surveyed here retain either *hoxa7a* or *hoxb7a* or they have both of these genes. There is a remnant of the pipefish *hoxa7a* sequence, found between *hoxa5a* and *hoxa9a*; it is likely a pseudogene, as there is no trace of the sequence for the homeobox-containing second exon and an early stop codon in the first exon is predicted also to eliminate the hexapeptide. In addition to these losses, the pipefish *hoxba* cluster remarkably no longer has *evenskipped* gene *eve1*, a gene that is present in zebrafish and all other percomorphs compared here (Fig. 4). We detected pipefish sequences for orthologs of long non-coding RNA genes *hotairm1* between *hoxa1a* and *hoxa2a*, and *hottip* between *evx1* and *hoxa13a* (not shown). *hotairm1* is missing in zebrafish and so far unreported in any teleost (though annotated in the Ensembl reference genome for spotted gar, an actinopterygian basal to the teleosts).

Syngnathus scovelli dlx gene clusters are missing deeply conserved non-coding elements

The vertebrate *dlx* genes, a family of homeobox transcription factors important for patterning the central nervous system, head skeleton, and limbs, are arranged in tandem pairs associated with specific *hox* clusters. Some percomorphs, like stickleback and pufferfish, retain *dlx1/2a*, *dlx3/4a*, *dlx3/4b*, and *dlx5/6a* clusters, while medaka appears to lack a *dlx3/4a* cluster, and zebrafish (a non-percomorph) has lost *dlx3a* but has retained an unpaired *dlx2b* not found in percomorphs [40]. We found the four typical percomorph clusters, totaling eight genes, in the Gulf pipefish genome and performed a search via mVISTA [41, 42] for conserved non-coding elements (CNEs) within the *dlx* clusters by comparing sequences from mammals and other teleosts. We found that pipefish retains some non-coding elements conserved between mammals and teleosts, as well as other CNEs shared only among teleosts [40, 43] (Fig. 5; see Additional file 1: Figure S3 for VISTA comparisons of the *dlx3/4a*, *dlx3/4b*, and *dlx5/6a* clusters). For example, we identified pipefish orthologs of two inter-*dlx* CNEs (Fig. 5) that were found previously to be conserved between mouse, zebrafish, and pufferfish and that were shown to direct reporter gene expression in subsets of *dlx* domains [43]. A third CNE that was not functionally tested but was conserved in both zebrafish and pufferfish [43] is not preserved in pipefish. We identified two other notable losses in this pipefish cluster: *S. scovelli* has lost an inter-*dlx1/2a* CNE that we find conserved in the other percomorphs, and it also lacks an

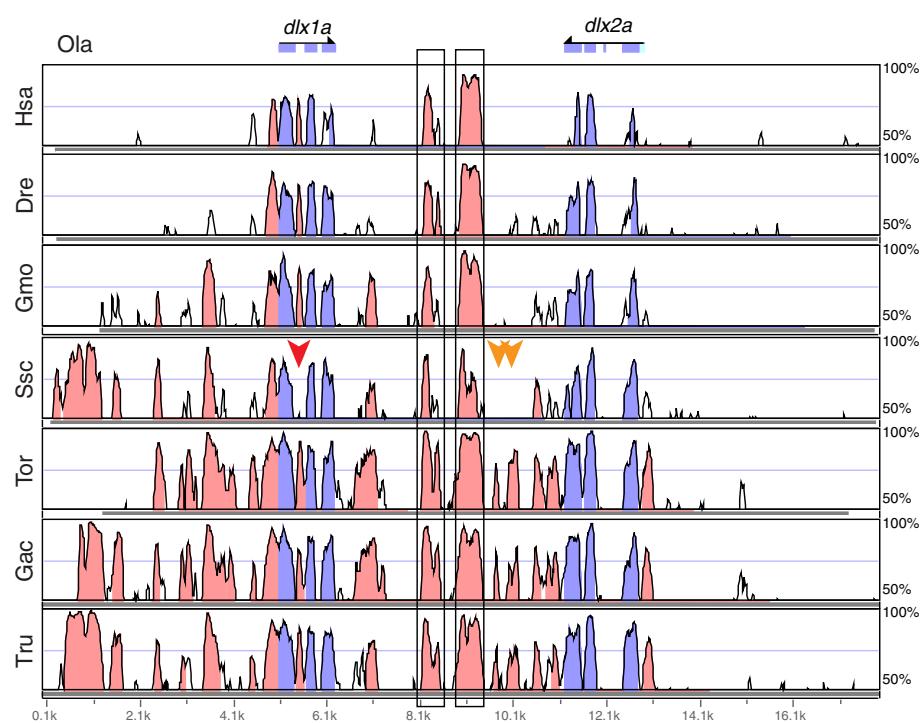


Fig. 5 Three CNEs are not detectable in the pipefish *dlx1a-dlx2a* cluster. One CNE present in other teleosts and mammals is missing from a gapless region between exon 1 and -2 in the *S. scovelli* assembly (red arrow). Two other CNEs in the *dlx* intergenic region that are conserved among percomorphs are also missing from this region in pipefish (orange arrows). Two CNEs previously shown to direct reporter gene expression in murine *Dlx* expression domains are boxed [43]. Exons are highlighted in blue, CNEs in pink. The reference, Ola, is medaka; Hsa, human; Dre, zebrafish; Gmo, cod; Ssc, pipefish; Tor, tuna; Gac, stickleback; Tru, pufferfish

element in the intron between coding exon 1 and exon 2 of *dlx1a*, a CNE that is conserved in both mammals and other teleosts. There are no gaps in the assembly in these regions of the pipefish genome. Several other CNEs are missing from other clusters, including two elements on either side of the last exon of *dlx4a* that are, notably, conserved between other percomorphs such as pufferfish and stickleback and cod, a non-percomorph (Additional file 1: Figure S3).

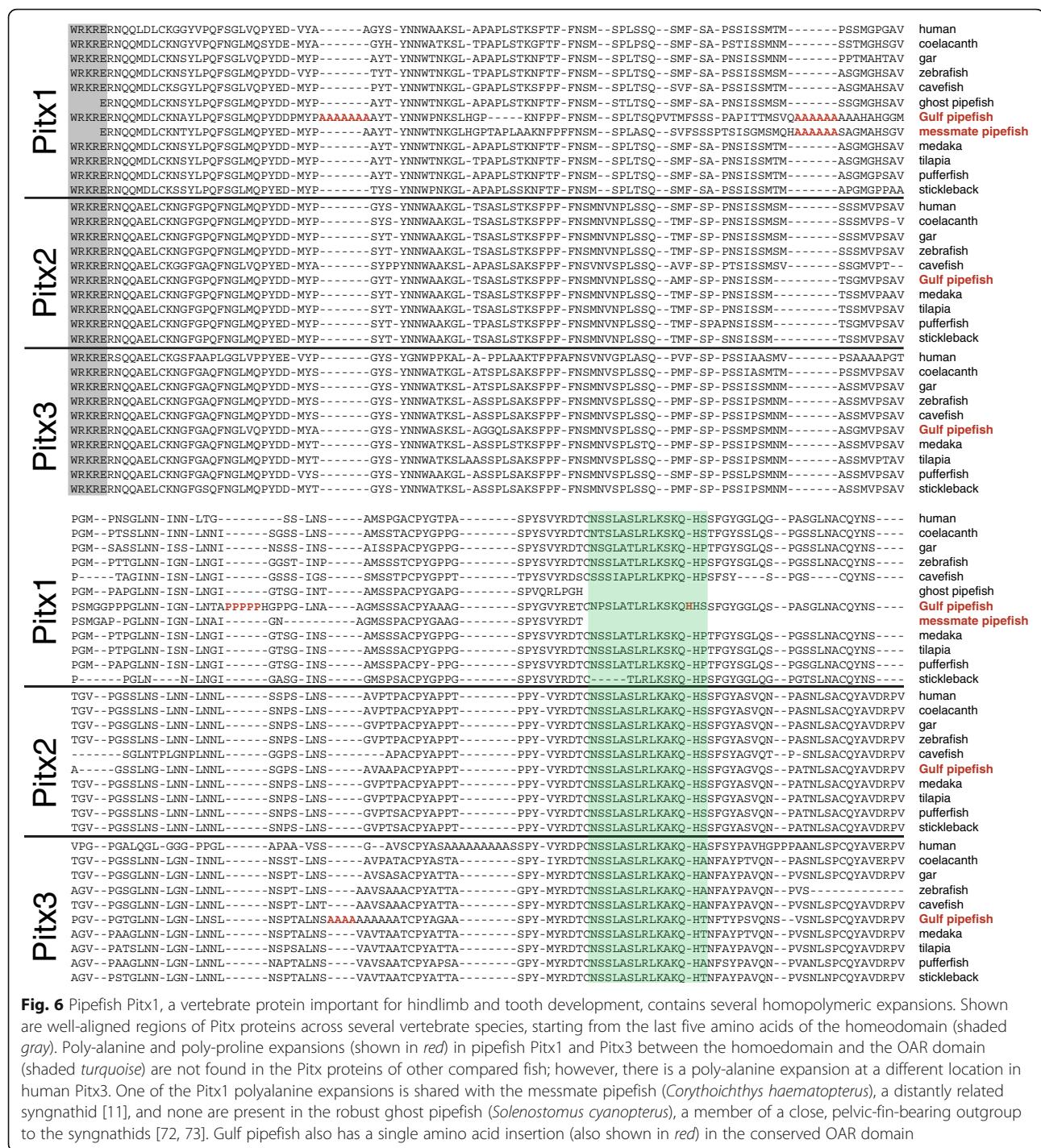
Syngnathid hindlimb loss implicates modification of the *tbx4-pitx1* pathway

Pipefish, seahorses, and seadragons all lack paired pelvic fins. *tbx4*, *pitx1*, and *pitx2* are genes at the top of the regulatory cascade described in vertebrate hindlimb development, including teleosts that have pelvic fins [44–46]. We found no trace of the protein-coding sequence for *tbx4* in the pipefish genome assembly. The genomic segments flanking *tbx4* were also not identified, as pipefish orthologs of genes adjacent to *tbx4* in other teleosts were either undetected, as in the case of *tbx2b*, or were on small scaffolds not anchored to the genetic map. TBLASTN also failed to identify *tbx4* among our de novo assembled gene transcripts generated from RNA-seq data. Gulf pipefish *pitx1* is present in the assembly but divergent. The predicted

pipefish Pitx1 amino acid sequence, supported by transcriptome sequencing, contains homopolymeric expansions of alanine and proline, and an amino acid insertion in the conserved OAR domain not seen in orthologs from other fish lineages or from human (Fig. 6). A fragment amplified with degenerate polymerase chain reaction (PCR) primers shows that a second syngnathid species, the messmate pipefish (*Corythoichthys haematopterus*), shares one of the alanine expansions (Fig. 6). Both Gulf pipefish and human Pitx3, a protein associated more strongly with eye and neural development than limb development [47, 48] also have polyalanine runs in different locations from those found in Pitx1. Pitx2 aligns well with other fish orthologs and apparently contains no homopolymeric expansions.

Pregnancy-specific gene expression in the brood pouch is widespread and reflects regulation of the innate immune system

We aligned to the annotated genome RNA-seq data from six pregnant male brood pouches (excluding embryonic tissue) and six non-pregnant male pouches. Based on these digital gene expression data, the transcriptional landscape of male brooding tissues differed substantially as a consequence of pregnancy, as 26.19% of the total



multivariate dissimilarity among the 12 individual transcriptomes was explained by pregnancy status (Additional file 1: Figure S4a; perMANOVA: $F_{1,11} = 3.55$, $p = 0.004$). Univariate tests of differential expression between pregnant and non-pregnant males revealed different transcript abundances for 1145 genes of 15,253 genes (false discovery rate (FDR) = 0.1) expressed robustly across at least four of 12 individuals. In total, 526 genes were pregnancy-

enriched and 619 were pregnancy-depressed, demonstrating fold change differences as extreme as 215 (Tables 2 and 3; see Additional file 2: SH2 for a complete tabulation of differentially expressed genes).

We identified several KEGG pathways enriched for genes subject to strong pregnancy-specific expression patterns, including “complement and coagulation cascades,” “cytokine-cytokine receptor interaction,” “calcium

Table 2 List of the top 15 pregnancy-enriched pouch tissue genes

Gene ID	Fold change	CPM	p value	Gene description	KO ID
SSCG00000006913	15.66	7.22	2.13E-24	WNT1-inducible-signaling pathway protein 2 isoform X2	K06827
SSCG00000005974	21.04	6869.88	1.87E-18	patristacin, partial	K08778
SSCG00000007802	4.15	93.44	7.69E-16	podocan	
SSCG00000014514	3.15	46.38	1.45E-15	fos-related antigen 2-like	
SSCG00000015977	12.38	229.24	1.39E-14	myocilin-like	
SSCG00000006209	6.53	4.72	4.91E-14	dickkopf-related protein 2	K02165
SSCG00000007875	2.93	188.72	8.81E-14	neuroepithelial cell-transforming gene 1 protein	
SSCG00000013720	5.13	233.89	3.85E-13	lipopolysaccharide-binding protein/bactericidal permeability-increasing protein	
SSCG00000011252	2.88	72.11	2.72E-12	beta-galactoside alpha-2,6-sialyltransferase 1-like isoform X1	K00778
SSCG00000004944	6.64	29.73	7.33E-12	collagen alpha-2(VI) chain-like	K06238
SSCG00000006480	3.10	18.93	1.81E-11	CTTNBP2 N-terminal-like protein	
SSCG00000013244	2.30	34.04	2.10E-11	LIM domain transcription factor LMO4-B-like	
SSCG00000004636	3.22	386.88	3.62E-11	NA	
SSCG00000002072	29.24	1.59	3.77E-11	potassium channel subfamily K member 2-like	K04913
SSCG00000007792	5.21	7.06	4.20E-11	excitatory amino acid transporter 5-like	K05618

Included are the fold change (pregnant/non-pregnant), average expression level across 12 pouch libraries in copies per million (CPM), edgeR negative binomial exact test p value, gene description from top BLASTP hit, and the assigned KEGG orthology ID for each pipefish gene. See Additional file 2 SH2 for the full list

signaling,” and “neuroactive ligand-receptor interaction” (See Additional file 2: SH3 for a full tabulation of KEGG pathways enriched for differentially expressed genes). Many pipefish genes within the first two of these pathways, which include innate immune system cascades, were expressed at higher levels in pregnant, relative to non-pregnant, pouch tissues. For example, members of the complement membrane attack complex (MAC), which are cell membrane pore-forming toxins [49] (reviewed in [50]), tended to be expressed at higher levels in pregnant

males (Additional file 1: Figure S5a, S6a). Pro-inflammatory chemokines Il8, Cxcl9, Cxcl10, and Cxcl12 of the Cxc subfamily were also expressed at higher levels in pregnant males, as were several members of the Cc subfamily (Additional file 1: Figure S5b). Not all transcriptional signatures of the immune system reflected this pattern, however. A suite of genes belonging to the natural killer cell cytotoxicity response pathway, for example, was expressed at higher levels in non-pregnant males (Additional file 1: Figure S4d). Furthermore, genes in

Table 3 List of the top 15 pregnancy-depressed pouch tissue genes

Gene ID	Fold change	CPM	p value	Gene description	KO ID
SSCG00000006879	27.36	56.49	7.91E-43	Serine/threonine-protein kinase WNK2	K08867
SSCG00000018539	12.37	15.96	2.04E-26	FXYD domain-containing ion transport regulator 12	
SSCG00000007973	4.73	53.34	1.66E-24	A disintegrin and metalloproteinase with thrombospondin motifs 6, partial	K08621
SSCG00000013585	10.78	19.10	1.07E-23	Tetratricopeptide repeat protein 18	
SSCG00000005985	214.58	652.27	7.29E-23	patristacin, partial	K08076
SSCG00000008728	14.12	6.03	2.22E-22	Uridine-cytidine kinase-like 1	K00876
SSCG00000000969	4.32	19.82	1.25E-17	ras-like protein family member 11A	K07852
SSCG00000017729	6.14	359.52	1.71E-17	nidogen-2-like isoform X5	K06826
SSCG00000004506	6.00	12.98	4.08E-17	syntaxin-2-like isoform X1	K08486
SSCG00000010275	14.47	3.28	1.00E-16	acid-sensing ion channel 1	
SSCG00000016046	6.75	8.51	1.51E-16	leucine-rich repeat-containing protein 4-like	K16351
SSCG00000014649	10.15	7.67	1.77E-16	homeobox protein MSX-2-like	K09341
SSCG00000019217	66.66	3.26	1.82E-16	leucine-rich repeat-containing protein 3-like	
SSCG00000007661	5.19	24.20	2.23E-16	cytochrome P450 27C1-like	K17951
SSCG00000005388	19.81	1.44	5.60E-16	glutamate receptor ionotropic, delta-2 isoform X5	K05207

Included are the fold change (non-pregnant/pregnant), average expression level across 12 pouch libraries in copies per million (CPM), edgeR negative binomial exact test p value, gene description from top BLASTP hit, and the assigned KEGG orthology ID for each pipefish gene. See Additional file 2 SH2 for the full list

KEGG pathways associated with the adaptive immune system, including “antigen processing and presentation,” “T cell receptor signaling pathway,” and “B cell receptor signaling pathway,” were transcriptionally less sensitive to pregnancy status than those in innate immunity KEGG pathways (Additional file 1: Figure S6b). Consistent with a characterization of the immune gene repertoire in *Syngnathus typhle* [51], we failed to detect MHC class II alpha and beta chain genes in the genome of *S. scovelli*, so the potential for some functionality of the adaptive immune system in this pipefish genus may be limited in general.

Gene Ontology terms overrepresented among pregnancy-enriched genes included those related to the complement system, coagulation, and immunity, consistent with the KEGG analysis, but we also identified terms related to hemopoiesis, homeostasis, proteolysis, and others (Additional file 2: SH5). GO terms overrepresented among pregnancy-depressed genes included those related to developmental processes, cell-to-extracellular matrix (ECM) adhesion, and protein glycosylation (Additional file 2: SH6).

Lineage-specific duplication of *patristacins* associated with male pregnancy

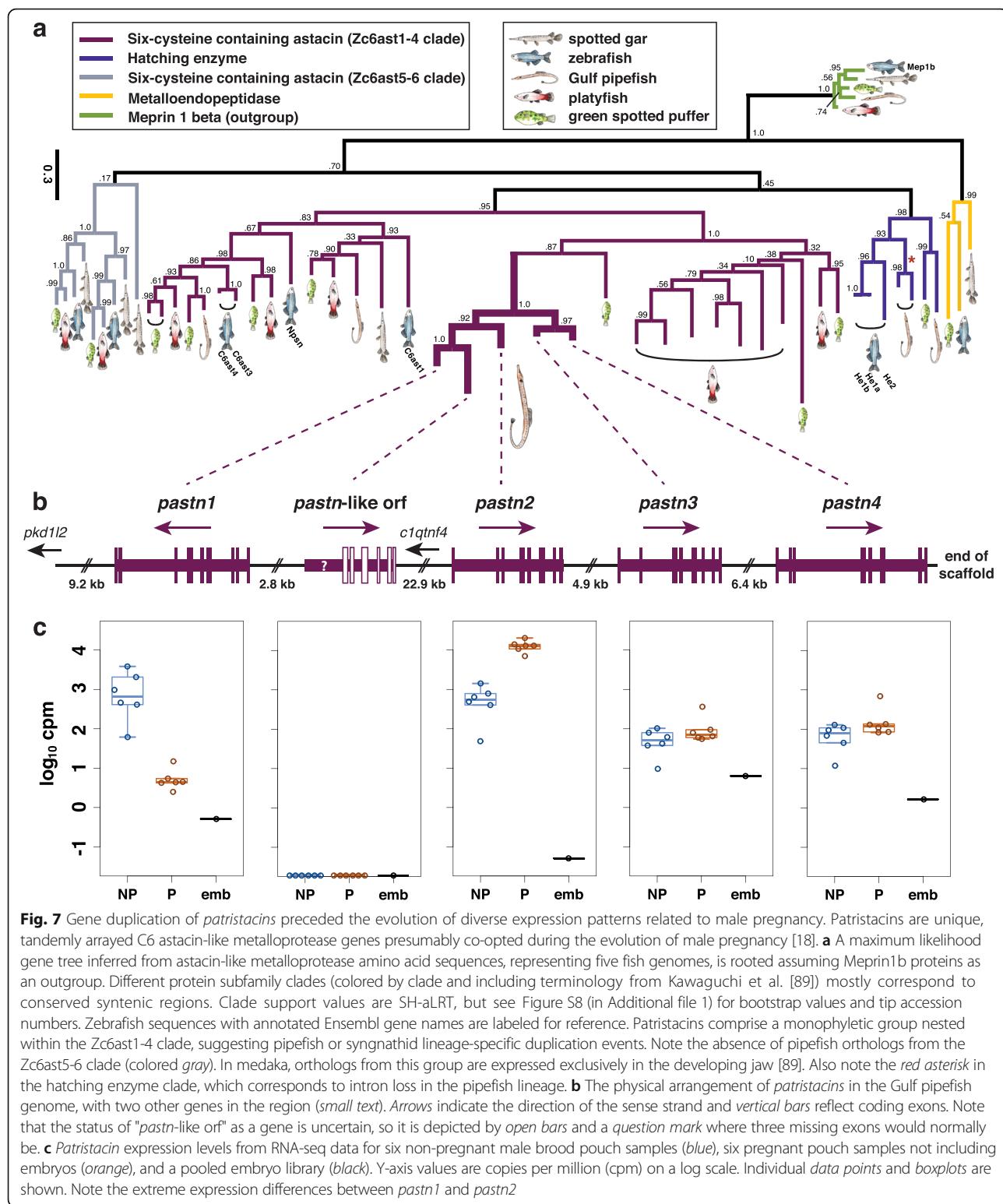
As documented previously in *S. scovelli* and *S. floridae* [52], two similar astacin-like metalloproteases demonstrated strikingly opposite patterns of gene expression: one markedly pregnancy-enriched and the other highly pregnancy-depressed (Table 2, Table 3, Fig. 7b, c). We here find that these “*patristacins*” [18] are adjacent genes belonging to a small cluster of duplicates that includes two additional *patristacins* expressed at lower levels in the brooding tissues at the stages examined (Fig. 7b, c). This cluster, located on scaffold 62 of pipefish LG4, also included a fifth, partial coding sequence for which we could identify neither a likely start methionine nor the first three typical *patristacin* exons. A phylogenetic analysis including astacin-like metalloprotease sequences from global searches of five ray-finned fish genomes suggests that the *patristacin* cluster is a gene family expansion unique to the lineage leading to syngnathids (Fig. 7a). We found protein-coding genes from platyfish and green spotted puffer genomes that share a recent common ancestor with *patristacins*, but these sequences were not nested within the *patristacin* subclade. Furthermore, *patristacins* and their closest homologs most likely diverged via gene duplication from the subfamily of 6-cysteine astacins that includes zebrafish *nephrosin*, given the topology of our current gene tree and that all paralogs share the same genomic region on pipefish LG4.

Discussion

Despite the explosive teleost species radiation over the last 300 million years, these fishes have been conservative in karyotype evolution relative even to the much younger mammalian lineage, with the majority of teleost species having a haploid number of 24 or 25 [53]. Variations from the inferred ancestral number of 24 [26] do exist across the teleost radiation, stemming from chromosome duplications, fissions, and fusions. We have shown that two chromosomal fusions in an ancestor of *Syngnathus scovelli* have likely led to a haploid karyotype of 22 (Fig. 2a and b). Comparisons of sequenced genomes suggest that interchromosomal rearrangements (translocations) are relatively uncommon in teleosts [53] and this is reflected in the striking one-to-one correspondence of chromosomes across most of the genome between Gulf pipefish and other percomorphs, such as southern platyfish (Fig. 2a). The stability of teleost genomes simplifies comparisons and increases confidence in correctly determining orthology of genes and chromosome segments based on observed patterns of conserved synteny. We have exploited the exceptional conservation of synteny among sequenced teleosts to explore the evolution and behavior of genes that might play a role in syngnathid innovations.

The remarkable morphology of syngnathids was noted in “The History of Animals” by Aristotle, who construed the peculiar phenomenon of pipefish live birth as a splitting open of the body. Prior to our characterization of the Gulf pipefish genome, however, with the exception of a few transcriptomic resources [51, 52, 54], virtually no information existed for how key developmental genes and their modification might be responsible for derived syngnathid phenotypes. Now, with the availability of the genome of *Syngnathus scovelli*, and likely other related genomes soon to follow, we expect researchers interested in the developmental genetic underpinnings of novel vertebrate morphologies to make the critical experimental connections between genomic differences in syngnathids and their functional consequences. In anticipation of exciting functional genomics work enabled by the latest genome editing approaches [55, 56], here we highlight a few especially promising examples of molecular signatures with implications for hallmark traits of pipefishes, seahorses, and their relatives.

We explored the constitution of the syngnathid *hox* genes because these Vertebrate Hox clusters are tandem arrays of transcription factor genes with many developmental roles, including segmental identity in the axis and in limb morphogenesis (reviewed in [36, 57]). Our investigation of Gulf pipefish *hox* cluster content revealed that the evolution of an elongated, ribless body was not accompanied by drastic reorganization of the *hox* genes. While there are multiple losses of pipefish



hox genes and the *hox*-regulating microRNA genes that are interspersed among them, many of these same genes have been lost from other percomorphs that have less modified skeletons (Fig. 4).

Two gene losses from the Gulf pipefish *hox* clusters stand out, however. The loss of *evel* is unique among described teleost *hox* clusters. This gene belongs to the *even-skipped* (*evx*) gene family, whose members reside at

the ends of particular clusters. In zebrafish embryogenesis, the *hoxba* cluster-associated *evel* gene is expressed during gastrulation and in the extending tail tip; its knockdown suppresses trunk and tail development, prompting the experimentalists to suggest *evel* acts as a posterior organizer [58] (but see [59] for another interpretation). It is therefore remarkable that *evel* is deleted in pipefish (Fig. 4). It is possible that some of these early ontogenetic functions of *evel* have been distributed to the remaining two pipefish *evx* genes or otherwise compensated for. However, syngnathids have neither oral nor pharyngeal teeth, consistent with evolutionary loss of *evel*, the only reported *evx* gene that is expressed during teleost tooth development [60, 61]. In addition, it appears that pufferfish and pipefish lineages have independently lost all copies of *hox7*, a paralogy group that when experimentally knocked out in mouse causes reduction and mispatterning of ribs [62]; consistent with this biological role for *hox7*, both pufferfish and pipefish lack ribs.

A uniting trait of the Syngnathidae is an absence of pelvic fins. Two other percomorphs that have evolutionarily lost pelvic fins appear to have done so by alteration of a hindlimb-positioning *hoxd9a* expression boundary (pufferfish [63]) or by loss of *pitx1* expression in the developing hindlimb (freshwater threespine stickleback [64, 65]). *Pitx1*, a transcription factor, directly activates initial expression of *tbx4* in the hindlimb primordium [66] and *tbx4* is required for initial limb bud outgrowth [67]. We found that pipefish *pitx1* has an amino acid insertion in the OAR, a functional domain thought to modulate DNA binding [68], and unusual homopolymeric alanine and proline repeat expansions between the homeodomain and OAR (Fig. 6). Homopolymers are known to cause several developmental diseases in humans (reviewed in [69]) and to affect subcellular localization, protein-protein interaction, and transcriptional regulation [70, 71]. In particular, expansions of alanine and proline homopolymers within transcription factors can modulate the proteins' ability to regulate transcription of gene targets. A distantly related pipefish species, the messmate pipefish, shares one of the homopolymeric repeats (Fig. 6), suggesting that this divergence of *pitx1* began early in the syngnathid lineage. It is conceivable that changes in the amino acid sequence of syngnathid *Pitx1* have had functional consequences for the protein's interaction with its gene targets (such as *tbx4*), affecting hindlimb development. We found no pipefish ortholog of *tbx4*. Failure to find pipefish *tbx4* in the genome assembly does not necessarily mean the gene has been evolutionarily lost; however, the possible loss of this gene with an apparently narrow developmental role in teleosts – in hindlimb development [46] – is consistent with the evolutionary loss of the hindlimb itself in syngnathids. Loss of the pelvic fins in a

syngnathid ancestor may have occurred shortly before or after the origin of the lineage, because the closest extant relatives – the ghost pipefishes (Family Solenostomidae) [72, 73] – have large, clasping pelvic fins in which females brood the embryos [74]. Interestingly, *Pitx1* in robust ghost pipefish (*Solenostomus cyanopterus*) lacks the homopolymeric repeats described above (Fig. 6).

A family of homeodomain transcription factors important for limb, brain, and craniofacial development, the *Dlx* genes, are arranged in gene pairs associated with specific Hox clusters. Within and near the *Dlx* gene pairs are CNEs recognizable by alignment among sequences from even distantly related vertebrates. Several teleost *dlx* clusters, for example, have CNEs in common with mammals [40, 75]. Putatively these CNEs are preserved because they have a function, perhaps in regulating gene expression of the *dlx* genes themselves. For instance, two CNEs that fall between *dlx1* and *dlx2* and that are conserved between teleosts and mammals direct reporter gene expression in the developing forebrain and first and second pharyngeal arches in murine [43] and in zebrafish [75] embryos. We found that pipefish has retained these two ancient CNEs but has apparently lost a third element that is as deeply conserved (i.e. between mammals and teleosts), from within an intron of *dlx1a*. In addition, at least two more CNEs in the intergenic region of *dlx1/2a* that are conserved among other percomorphs are lost or diverged beyond recognition in pipefish (Fig. 5). Experimental mutation of mouse *Dlx1/2* genes creates defects in the development of pharyngeal arch derivatives, such as the mandible and teeth [76]. Knockdown of these genes in zebrafish causes embryos with shortened faces and mispatterning of first and second arch cartilages and a reduced ethmoid (a cartilage of the ventral neurocranium) [77]. In addition, *dlx2* genes are expressed in developing teeth in cichlids, catfish, and cyprinids [78–80], and *dlx2a* is expressed in migrating neural crest that will form the anterior pharyngeal arch cartilages [77, 81]. Pipefish embryos show modified development of the anterior skull including cartilage derivatives of the first and second pharyngeal arches, particularly elongation of the hyosymplectic (a cartilage of the second arch), as well as unusual early curvature and later elongation of the ethmoid cartilage (see Additional file 1: Figure S7 for a view of pipefish craniofacial development), implicating changes in expression of early acting genes such as *dlx2a*, involved in cranial neural crest survival and patterning. Functional testing in other teleosts could reveal whether the CNEs here shown to be erased in pipefish are functional units that modulate expression of the *dlx1/2a* cluster genes and possibly affect pharyngeal arch or tooth development.

Male pregnancy in syngnathid fishes is a true example of evolutionary novelty. In many lineages, including *S.*

scovelli, males gestate developing embryos in a tightly regulated environment defined by a complex brood pouch. Extensive cellular and developmental changes in the pouch occur leading up to and during pregnancy, including proliferation of epithelial cells, development of specialized secretory cells, and angiogenesis [10, 82, 83]. These specializations are likely the consequence of adaptation, as they enable functions directly relevant to fitness, including solute, gas, and nutrient delivery to a male's brood [12, 13, 84], as well as immune priming of offspring [85]. Consistent with this functional diversity, our genome-based analysis of male pregnancy in *S. scovelli* revealed a transcriptionally rich brood pouch in which over 73% of annotated genes were expressed robustly and over 1000 were differentially expressed as a consequence of pregnancy (Additional file 2: SH2). Previous studies, based on de novo transcriptome assemblies, characterized pregnancy-specific gene expression in pipefish species of *Syngnathus* [52] and in the seahorse *Hippocampus abdominalis* [54], but lack of a reference genome in those surveys limited insights into the transcriptional breadth of the pouch and single gene resolution for transcript abundance measurements. Our differential expression analysis comparing early-stage pregnant to non-pregnant male pouch tissue echoes many of the patterns described in the comprehensive seahorse study [54], including evidence for positive regulation of developmental processes, lipid transport, homeostasis, and the immune system during pregnancy. Interestingly, we noted a more pronounced signature of pregnancy-specific gene expression for innate, relative to adaptive, immune pathways in Gulf pipefish (Additional file 1: Figure S6). This observation is likely in part a consequence of pipefishes in *Syngnathus* having lost important genetic components of MHC class II mediated immunity [51], although MHC class I components remain intact. Syngnathid fathers face unique demands with respect to immunity and pregnancy, given that the brood pouch is a non-urogenital organ more directly exposed to the environment than internal uterine structures of other vertebrates. A seemingly difficult balance among pathogen control, maintenance of beneficial microbes, and mitigation of attack against non-self (embryonic) tissues must therefore be struck. Although future work regarding the details of this balance will be required to say so, perhaps a uniquely fine-tuned division of labor between innate and adaptive immunity has been an evolutionary outcome of male pregnancy, a balance we hypothesize differs across syngnathid lineages with varying brood pouch complexity.

The significance of gene duplication to adaptation and biological diversification in general is continually of interest to evolutionary biologists [86–88]. We identified at least four clustered members of the *patristacin* gene

subfamily on a single scaffold of LG4 in the Gulf pipefish genome (Fig. 7). Given the striking patterns of gene expression for *pastn1* and *pastn2* with respect to pregnancy, it is possible that gene duplication followed by neo- or subfunctionalization played a key role in the evolution of male pregnancy, although surveys of other syngnathid genomes and those of their closest relatives are needed to test this hypothesis. Our interpretation of the evolution of *patristacins* is distinct from that of Harlin-Cognato et al. [18], who suggested that one *patristacin*, identified without the advantage of a complete *S. scovelli* genome, took on a novel role in male pregnancy by a spatiotemporal shift in gene expression and not via gene duplication. Our genome-wide approach has provided additional information, however, by revealing the complete coding sequence for multiple *patristacin* paralogs in *S. scovelli*. Because the two *patristacins* with exceptional pregnancy-specific gene expression (*pastn1* and *pastn2*) likely diverged by gene duplication after pipefish separated from the other fish lineages in our comparison, we provide evidence for a role of relatively recent gene duplication in *patristacin* evolution. Our phylogenetic analysis highlights a second, large expansion of *patristacin*-like genes in the genome of *Xiphophorus maculatus*, suggestive of high duplicate retention in multiple live-bearing fish lineages.

The specific functional roles *patristacins* play in male pregnancy are currently unknown, but our current phylogenetic understanding of their place among teleost Astacin-like metalloproteases suggests that they may be more functionally similar to Nephrosin-like proteins than hatching enzyme components (Fig. 7a, Additional file 1: Figure S8). Kawaguchi et al. [89] showed, for example, that medaka 6-cysteine astacin genes *mc6ast1* and *mc6ast2*, orthologs of zebrafish *c6ast1* and zebrafish *c6ast3/4*, respectively, were expressed in a wide range of tissues, in contrast to medaka hatching enzymes, which were expressed exclusively in pre-hatching embryos. Another member of this gene subclade, *cimp1*, is expressed epithelially in the developing cichlid jaw and may play a role in ECM turnover during development [90]. We hypothesize that *patristacins* evolved from an already transcriptionally promiscuous ancestor and now, following subsequent duplication events, work in concert to regulate the remodeling of the pouch epithelium necessary for the sustenance of pregnancy. Our characterization here of their structural organization and expression patterns in the brood pouch will inform and facilitate future functional studies of these gene duplicates and their specific roles in male pregnancy.

Conclusions

We present the first annotated reference genome assembly, organized into chromosomes, for a syngnathid fish.

Our comparisons of the Gulf pipefish genome to other fish genomes reveal two chromosomal fusions in the syngnathid lineage. We provide additional evidence suggesting that syngnathiform fishes are an outgroup relative to fellow percomorph fishes commonly used in comparative genomics studies. The Gulf pipefish genome will therefore serve as a useful comparator in studies that aim to understand rates of genome evolution among percomorphs for which there are existing genomic resources. We show that losses of both genes and CNEs have occurred in pipefish gene families important for vertebrate craniofacial, tooth, hindlimb, and axial development, all features that are highly modified in syngnathids. In addition, we detail aspects of the molecular biology of male pregnancy, a unique and unifying feature of the pipefish, seahorses, and seadragons; in particular, we exploited the annotated Gulf pipefish genome and transcriptional profiling to show how pregnancy is associated with clear changes in gene expression in the male brood pouch tissue, a broad example being regulation of the innate immune system and a specific example being regulation of duplicated *patristacins*.

Methods

Genome sequencing libraries and genome sequence assembly

We isolated genomic DNA from a single adult male pipefish purchased from Gulf Specimen Marine Laboratories, Inc. (Panacea, FL, USA) in 2010 using standard organic extraction. We generated four different 100 nt paired-end Illumina libraries for whole genome shotgun assembly: (1) a short (~180 bp) insert length library; (2) a 2.5–5 kb insert length jumping library; (3) a 5–10 kb insert length jumping library; and (4) a 11–15 kb insert length jumping library. To construct the 180 bp library, we sheared 1 μg of genomic DNA to less than 500 bp using sonication in a Bioruptor (Diagenode) and size selected fragments by agarose gel electrophoresis, followed by end repair of the fragments, addition of adenosine overhangs, ligation of Illumina sequencing adapters, and 12 cycles of PCR amplification with Phusion polymerase (NEB). We used the Illumina Nextera Matepair Sample Preparation Kit (Illumina, cat. #FC-132-1001) to generate the three jumping libraries. Briefly, we performed a single fragmentation reaction using 5 ng of genomic DNA, selected the three aforementioned fragment size ranges using agarose gel electrophoresis, and performed the remaining library preparation steps in parallel, including circularization, shearing by Bioruptor (30 s on, 60 s off, for 15 min), streptavidin bead pull-down, end repair, addition of adenosine overhangs, Illumina indexed adapter ligation, and 15 cycles of PCR amplification. We sequenced the short-insert library (two lanes) and three jumping libraries (all in one lane) on an

Illumina HiSeq2000 at the University of Oregon Genomics Core Facility (UOGCF).

To minimize the inclusion of sequencing adaptors, sequencing errors, and repetitive DNA sequences in the assembly process, we used tools from the Stacks software suite [91, 92] to adaptor-trim and discard low-quality read pairs (*process_shortreads*) and filter pairs containing abundant k-mers (*kmer_filter*). Remaining were 238.6 million overlap pairs, 3.5 million 11–15 kb mate-pairs, 21.6 M 5–10 kb mate-pairs, and 44.4 M 2.5–5 kb mate-pairs, which we used for assembly with ALLPATHS-LG [21]. Because initial k-mer spectrum analyses suggested a highly polymorphic genome, we ran ALLPATHS-LG with HAPLOIDIFY = TRUE. To assess completeness of the assembly with respect to CEGs, we used CEGMA [22]. For a summary of all Illumina sequencing data used in the assembly, see Additional file 3.

We confirmed several apparent pipefish gene losses via comparison among preliminary genome assemblies derived from independently constructed molecular libraries and generated using SGA [93] and Velvet [94] and via targeted Sanger sequencing. Briefly, SGA and Velvet assemblies incorporated a shotgun genomic DNA library with an insert length of 470 nt, sequenced independently with 120 nt, 100 nt, and 80 nt paired-end Illumina reads. For the SGA assembly, the overlap value was optimized to 70 during the contig construction phase. Scaffolding was performed using SSPACE [95], with the three mate-pair libraries mentioned above and an additional 2–8 kb mate-pair library. These analyses filled seven small gaps in the range of 51–1753 nt in the *hoxba*, *hoxbb*, *hoxca*, and *hoxda* clusters. The degraded nature of *hoxa7a* was also confirmed by Sanger sequencing.

RNA-seq libraries and transcriptome assemblies

Embryo and fry transcriptome

Embryos, flushed from the pouch of lab-reared pregnant males, and fry were euthanized in Tricaine-S and stored in RNA-Later (Ambion). Tissue including the head to just posterior to the pectoral fin was dissected and pooled from 17 embryos (including 15 at 8 days post fertilization (dpf) and 2 at 10 dpf) and from 18 fry (including 2 at 16 dpf and 16 at 17 dpf). Double-stranded complementary DNA (cDNA) was produced from these tissues via standard methods including RiboPure Kit (Ambion) for total RNA isolation, MicroPoly(A)Purist Kit (Ambion) for messenger RNA (mRNA) enrichment, mostly hexameric Random Primers (ThermoFisher, #48190-011) and Superscript III reverse transcriptase (Invitrogen) for first strand synthesis, and Random Primers with Klenow exo-DNA polymerase (Epicentre). Paired-end Illumina sequencing libraries were created using standard methods including mechanical shearing of the cDNA and TA ligation of adaptors (top, 5'

ACACTCTTCCTACACGACGCTCTCCGATC*T3'; bottom, 5'Phos-GATCGGAAGAGCGGTTCAGCAG-GAATGCCGAG3'), slab gel size fractionation to isolate fragments in the 200–500 bp range, and amplification using Illumina-compatible primers (5'AATGATACGGC-GACCACCGAGATCTACACTCTTCCTACACGACG CTCTCCGATCT3' and P2 reverse primer, 5'CAAG-CAGAAGACGGCATACGAGATCGGTCTCGGCATTC CTGCTGAACCGCTCTCCGATCT3'). The library was sequenced on an Illumina GAIx platform to produce 60 nt paired-end reads and on an Illumina HiSeq2000 platform to produce 100 nt paired-end reads (see Additional file 3 for details).

Male brood pouch

Six non-pregnant and six early-stage pregnant adult males were captured from Redfish Bay, TX, USA (Lat: 27.86795057508745, Long: -97.08869218576297), transported to the laboratory, and euthanized as described above approximately 24 h after capture. We carefully dissected all brooding tissues, including the pouch "flaps" and epithelium, but excluding all embryonic tissue in the case of pregnant males. We fixed tissues in RNA-Later (Ambion) before freezing, homogenized by pestle upon thawing, and isolated total RNA using Trizol Reagent (Invitrogen) and RNeasy MinElute columns (Qiagen). A unique RNA-seq library was generated for each individual from 1 ug of total RNA using the TruSeq RNA v2 Kit (Illumina) and the 12 mRNA-seq libraries were sequenced across two lanes of Illumina HiSeq 2000, generating 100 nt paired-end reads.

De novo transcriptome assemblies

We removed low-quality and adaptor sequences from RNA-seq reads using *process_shortreads* from Stacks [91, 92], overlapped paired-end reads using FLASH [96], and performed rare k-mer filtering and digital normalization using *kmer_filter* from Stacks. We then generated two separate de novo transcriptome assemblies (one for each tissue type) from the cleaned, filtered RNA-seq data using Trinity [97] with *-min_kmer_cov* set to 3.

Genome annotation

Prior to genome annotation, the assembly was soft-masked for repetitive elements and areas of low complexity with RepeatMasker [98] using a custom Gulf pipefish library created by RepeatModeler [99], Repbase repeat libraries [100], and a list of known transposable elements provided by MAKER [25]. In total 15.36% of the genome assembly was masked by RepeatMasker. Repetitive elements were annotated with RepeatModeler. Hidden Markov models (HMMs) for gene prediction were generated by SNAP [101] and Augustus [102] and

were iteratively trained for the assembly using MAKER as described by Cantarel et al. [103]. Training was performed on the five largest scaffolds and two additional scaffolds that were UTR rich, totaling 25 Mb. Evidence used by MAKER for annotation included Gulf pipefish mRNA-seq transcriptomes from embryonic head tissue and brood pouch tissue (assembled with Trinity – see above), protein sequences from threespine stickleback (*Gasterosteus aculeatus*), zebrafish (*Danio rerio*), medaka (*Oryzias latipes*), and tilapia (*Oreochromis niloticus*) (downloaded from Ensembl: Broad S1, GRCz10, HdrR, Orenil1.0, respectively), and all Uniprot/swissprot proteins [104].

We filtered the annotations by MAKER to include evidence-based annotations with assembled transcriptome or protein support and those ab initio gene predictions that contained protein family domains as detected with InterProScan [105]. Gene annotations were manually refined for *hox*, astacin-like metalloprotease, and *pitx* genes. For each annotated amino acid sequence we queried the NCBI nr database using BLASTP and compiled the results for the top BLASTP hit per gene in Additional file 2: SH6.

Linkage map and map integration

Mapping cross

For the genetic cross, wild male and female *S. scovelli* were captured from Redfish Bay and maintained in the lab. A total of six sequential broods from a single mated pair, totaling 108 F1 progeny, including fry from the brood pouch plus 15 collected just prior to emergence, were gathered and flash frozen over a span of 4 months. Genomic DNA was isolated from individual progeny and from their parents via the Qiagen DNeasy Kit. RAD-seq libraries were made using the restriction enzyme SbfI as in Baird et al. [106], Hohenlohe et al. [107], and Etter et al. [108] with the Illumina-compatible, bar-coded P1 adapters and primer types used in Hohenlohe et al. [109] and the P2 adapter type used in Hohenlohe et al. [107]. Single-end reads of 100 nt were produced from two lanes on an Illumina HiSeq2000 (see Additional file 3 for details). The parents were sequenced to greater depth than progeny (see below) to make an accurate catalog of diploid genotypes possible in the cross.

Marker genotyping

The two lanes of Illumina data resulted in 367,085,475 raw reads which were analyzed using the software, Stacks [91, 92]. Using the *process_radtags* program, reads were demultiplexed according to barcode and discarded if the barcode could not be determined after correcting for sequencing error, if the restriction enzyme cut site was not intact, or if the sequencing quality was too degraded. The 218,309,324 remaining reads were

analyzed by the Stacks de novo pipeline to assemble and genotype the RAD loci. A minimum of three identical reads ($-m\ 3$) was required to form a “stack” or putative allele in each individual, up to five differences were allowed when merging stacks into putative loci ($-M\ 5$) and up to 3 differences were allowed when merging loci from different individuals into the catalog ($-n\ 3$) to accommodate fixed differences between the cross parents. The *genotypes* program from Stacks was used to export data in a CP cross-format for use in JoinMap and the genotypes were uploaded to the Stacks web interface. Genotype data with markers present in at least 75 of the 108 individual progeny were exported from the web interface for linkage analysis.

Map construction

Linkage analysis was performed with JoinMap 4.1 [110] using only markers that were present in at least 75 of the 108 individual progeny. Markers were initially grouped in JoinMap 4.1 using the “independence LOD” parameter under “population grouping” at a minimum LOD value of 15.0, and markers that remained unlinked at $LOD < 15$ were excluded. Marker sets were partitioned into paternal and maternal markers to enable the construction of sex-specific linkage maps. Marker ordering was performed using the Maximum Likelihood (ML) algorithm in JoinMap 4.1 with default parameters. Supposed double recombinants were identified using the “genotype probabilities” feature in JoinMap 4.1 and by visual inspection of the colorized graphical genotypes in the male, female, and consensus maps. After visual inspection of the individual sequences in the web interface of Stacks, markers were manually corrected as needed in the web interface and re-exported. For example, if a double recombinant was a homozygote with a small number of sequences, the genotype was eliminated because it might represent a heterozygote with no sequences for the second allele. Conversely, if the double recombinant was a heterozygote with only one sequence for the second allele, the genotype was eliminated because the second sequence could be sequencing error. The new dataset with corrected genotypes was loaded again into JoinMap 4.1 and the process was repeated until no suspect genotypes were identified. The “expected recombination count” feature in JoinMap 4.1 was used to identify individuals with higher than expected recombination events; marker order was visually inspected and, when necessary, optimized by moving a marker or sets of markers to a new map position that reduced the number of recombination events. When a marker or sets of markers could be in multiple map positions, the markers were moved to a position congruent with their physically aligned scaffold location if there was no cost to the map.

Integrating the assembly and the linkage map

The 4375 markers from the linkage analysis were integrated with the assembled pipefish scaffolds to create a chromonome using the software, Chromonomer (<http://catchenlab.life.illinois.edu/chromonomer/>). Markers were aligned to the set of assembled pipefish scaffolds using GSnap [111], requiring unique alignments, allowing up to five mismatches ($-m\ 5$), counting gaps as four mismatches ($-i\ 4$), and requiring 99% of the RAD locus to align ($-min-coverage = 0.99$). The AGP file produced by ALLPATHS-LG that describes the assembly, the linkage group, and map position of the markers in the map, the alignments of the markers to the scaffolds, and the FASTA file containing the sequence from the assembly are all fed into Chromonomer, which integrates them in the following way. First, markers are arrayed along the scaffolds they are aligned to and scaffolds that have markers from more than one linkage group are identified (no scaffolds were split between linkage groups). A coherent ordering of markers must be found for each scaffold so that physical basepair and map position are consistent among all markers for that scaffold. Markers that are out of order with respect to the map or scaffold are discarded (unless it is the last marker holding a scaffold into the map). Of the 4375 markers, 649 were excluded in this phase, leaving 3726 markers in the final “chromonome.” If a scaffold spans more than one map position, and physical order is the same as map order, the orientation of the scaffold is positive. If physical and map order are inverted, the scaffold is considered in negative orientation and the sequence is reverse complemented. Otherwise orientation is unknown and the scaffold remains in positive orientation by default. Scaffolds are then hung from the linkage group they occur on, according to map position. Ordered markers may place the scaffold in more than one place within the linkage group, that is, one or more scaffolds occur within the focal scaffold according to the linkage map. This can be due to an incorrect assembly join or because a smaller scaffold is filling a gap in a larger scaffold. In these cases, the scaffold is split at the largest gap that can be found between the markers in the map that indicate where the split must occur. Starting with 553 scaffolds, five scaffolds were split one time each for a total of 558 scaffolds in the chromonome. Sequence from the scaffolds is then concatenated into chromosomes according to the orientation and integrated order with standard 100 bp gaps placed in between each join resulting in a chromonome of 266,330,253 bp (53.6Kb scaffold join gaps) with 40,734,039 bp of sequence remaining in unintegrated scaffolds. Finally, the genome annotation is translated to the new chromonome providing a genome-level ordering of genes for use in conserved synteny analysis and new AGP, FASTA, and GFF files are generated to describe the chromonome.

Conserved synteny analysis

In order to visualize evolutionarily conserved gene neighborhoods, i.e. conserved synteny, we used the Synolog software (Catchen, unpublished). We used Synolog to identify orthologs between the Gulf pipefish, threespine stickleback, medaka, green spotted pufferfish (*Tetraodon nigroviridis*), zebrafish, spotted gar, and southern platyfish and to identify conserved gene neighborhoods pairwise between the different species. Genome-wide images of conserved synteny were drawn by Synolog by combining the conserved synteny blocks across the genome and incorporating the integrated linkage map/assembly output by Chromonomer where appropriate (Fig. 2c). Protein gene models for each non-pipefish species were downloaded from Ensembl. While Synolog is a new and independent implementation, the algorithm to identify conserved synteny and the biological inferences stemming from its application are as described in Catchen, et al. [112].

Phylogenomic analysis using ultraconserved elements

We added UCEs from Gulf pipefish, Pacific bluefin tuna, and southern platyfish genomes to an existing UCE dataset containing sequences for 27 actinopterygian fishes and published by Faircloth et al. [32]. To retrieve each of the 491 UCEs from the three genomes above, we generated a consensus sequence of each alignment from Faircloth et al. [32] using *em_cons* from EMBOSS [113], searched for each consensus sequence in each genome using LASTZ [114], and extracted unique search hits from each genome using BEDTools [115]. For this we used the tuna reference genome available from http://nrifs.fra.affrc.go.jp/ResearchCenter/5_AG/genomes/Tuna_DNAMicroarray/index.html and the platyfish genome from Ensembl. We obtained 457, 453, and 479 single-copy UCEs for Gulf pipefish, tuna, and platyfish, respectively. A multiple sequence alignment for each UCE was generated using MAFFT v7 [116] with options –localpair and –maxiterate 1000, and minor manual adjustments were made when necessary.

We performed substitution model selection for each UCE alignment using the corrected Akaike Information Criterion, as implemented in jModeltest-2.1.10 [117, 118]. The GTR + gamma model was selected for the largest percentage of the total aligned sequence data. We concatenated UCE alignments, ordering them so that the loci having the same best-fitting substitution model were grouped together. We proceeded with a partitioned phylogenetic analysis using the concatenated alignment (153,032 nt total), and the GTR + gamma model for all partitions. Maximum likelihood (ML) phylogenetic inferences were conducted with RAxML version 8.2.4 [119] using default

settings. We produced a consensus ML tree using the rapid bootstrap search algorithm described in Stamatakis et al. [120]. Briefly, 1000 rapid bootstrap searches were conducted, followed by fast ML searches on 200 of these, followed by a slow ML search on the 10 best fast ML trees. Clade confidence was assessed with SH-aLRT support values and bootstrap replicate frequencies. We specified *Polypterus senegalus* as the outgroup for tree rooting.

Characterization of *hox* clusters

hox gene content

Teleost *hox* gene sequences acquired from Ensembl were used as queries for BLAST searches of the final Gulf pipefish genome assembly using Geneious (version 8.0.5). Exon boundaries were annotated by hand using alignments with the query *hox* genes. The *hox* genes annotated in the Gulf pipefish assembly were then BLAST-searched against the NCBI NR sequence database to confirm gene identity using Geneious (version 8.0.5). Additionally, *hox* genes were identified, following the method outlined above, in the Pacific bluefin tuna genome (see genome source above) [121].

hox cluster microRNAs and long non-coding RNAs within the *hox* cluster were identified using VISTA analyses based on CNEs within *hox* clusters across Gulf pipefish, threespine stickleback, mouse (*Mus musculus*), spotted gar, zebrafish, Pacific bluefin tuna, medaka, and fugu (*Takifugu rubripes*) [41, 42, 122–124]. We aligned primary miRBase [125] microRNA sequences from stickleback, zebrafish, medaka, and fugu to *S. scovelli* *hox* regions using MUSCLE [126] to supplement annotations. The hairpin loops of the annotated microRNAs were confirmed using RNAfold (<http://rna.tbi.univie.ac.at/cgi-bin/RNAWebSuite/RNAfold.cgi>). When known *hox* cluster microRNAs were not detected in the Gulf pipefish genome, we further confirmed absence of the conserved seed sequence, which was the case for *mir196b* between *hoxb13a* and *hoxb9a* and *mir10a* between *hoxb5b* and *hoxb3b*. All conserved non-coding sequences annotated within the Gulf pipefish *hox* cluster were queried against miRBase Sequence Databases (Release 21) for mature miRNA chordate sequences and miRNA chordate hairpins (downloaded from miRBase) using BBMapSkimmer [127] for further identification of microRNAs. Kmer index size was set to 7, max indel set to 0, approximate minimum alignment identity set to 0.50, secondary site score ratio set to 0.25, behavior on ambiguously-mapped reads set to retain all top-scoring sites, and maximum number of total alignments to print per read set to 4 million. See Additional file 2: SH7 for scaffold locations and sequences of microRNAs and long non-coding genes.

Characterization of *dlx* CNEs

CNEs between *dlx1* and *dlx2*, between *dlx3* and *dlx4*, and between *dlx5* and *dlx6* were identified using mVISTA analyses based on levels of sequence conservation within *dlx* clusters across Gulf pipefish, Atlantic cod, threespine stickleback, zebrafish, human, Pacific bluefin tuna, medaka, and fugu [41, 42, 122–124]. Sequences were downloaded from Ensembl for cod, stickleback, zebrafish, human, medaka, and fugu. Tuna sequences were downloaded from the reference genome source cited above. Medaka was set as the reference sequence for the *dlx1/2* and *dlx5/6* comparisons and stickleback was the reference for the *dlx3/4* comparisons. Alignment of each sequence from these species were aligned using the shuffle-LAGAN algorithm through the mVISTA website under default parameters. See Additional file 2: SH7 for scaffold locations of CNEs.

Characterization of pelvic fin development candidates

Pitx1, Pitx2, and Pitx3 protein sequences were obtained from our pipefish annotation, Ensembl, and Genbank (in the case of stickleback Pitx1) for human, coelacanth (*Latimeria chalumnae*), spotted gar, zebrafish, blind cavefish (*Astyanax mexicanus*), medaka, tilapia, green spotted pufferfish, and threespine stickleback, and aligned using MAFFT (with default settings). To isolate DNA fragments for Sanger sequencing of *pitx1* from the messmate pipefish (*Corythoichthys haematopterus*) and the robust ghost pipefish (*Solenostomus cyanopterus*) genomic DNA, we designed degenerate PCR primers (in IUPAC notation, forward 5'-CGGAGCGCAACCAGCARATGGA-3' and reverse 5'-GGACGACGACATGSCSCWGTGAT-3') for amplification using Phusion DNA polymerase (New England Biolabs) in Phusion HF buffer, and an annealing temperature of 55 °C.

Because *tbx4* was not represented in the pipefish genome annotation, we attempted to determine its location in the genome assembly manually by using a targeted profile HMM generated from several aligned teleost Tbx4 protein sequences. HMM-based approaches are more sensitive than BLAST-based approaches when searching for divergent homologs [128], a possible scenario when a gene has evolved rapidly or has degenerated. Briefly, we used an alignment of Ensembl Tbx4 sequences from spotted gar, zebrafish, medaka, southern platyfish, threespine stickleback, green spotted pufferfish, and tilapia to generate a profile HMM with hmmer2 [129], then searched for sequences in the Gulf pipefish genome with this model using the genewisedb program of wise2 (<http://www.ebi.ac.uk/~birney/wise2/>) with default search settings.

Differential expression analysis

We aligned adaptor- and low-quality-trimmed, forward reads from the 12 brood pouch RNA-seq libraries to the annotated Gulf pipefish genome using GSnap [111]. We counted the number of uniquely mapped reads per exonic region of each annotated gene using HTSeq-count [130] and used the counts to test for differential gene expression between pregnant and non-pregnant males using the negative binomial exact test [131], after TMM normalization, implemented by the R/Bioconductor package edgeR [132]. We limited differential expression analysis to those genes with at least one read per million counted (cpm) in at least four of the 12 fish, which reduced the dataset to 15,253 genes.

To connect genes annotated in the pipefish genome with putative functional information, we mapped the pipefish amino acid sequences to KEGG Orthology (KO) entries [133] using the KEGG Automatic Annotation Server [134]. We then identified KEGG PATHWAYS enriched for pipefish KOs with extreme log₂ fold change values from the pregnancy differential expression analysis using the R/Bioconductor package GAGE [135]. To visualize individual members of KEGG PATHWAYS enriched for pregnancy-sensitive genes we used the R/Bioconductor package Pathview [136]. We also used Ensembl IDs for putative *D. rerio* orthologs of Gulf pipefish genes to test for overrepresentation of PANTHER GO-slim Biological Process terms among pregnancy-enriched and pregnancy-depressed genes using binomial tests implemented by the online resource PANTHER (pantherdb.org), [137, 138]. For the overrepresentation tests, we used all genes tested for differential expression (see above) and matched with a zebrafish ortholog as the comparison set. To interpret the results of overrepresentation tests for pregnancy-enriched and pregnancy-depressed sets we only considered GO-Slim terms represented in the comparison set by at least five genes and we controlled the FDR at 0.1 as in Benjamini and Hochberg [139]. Results for these overrepresentation tests are in Additional file 2: SH4 and Additional file 2: SH5.

To visualize and quantify multivariate differences among individual brooding tissue samples in transcript space, we calculated Bray-Curtis dissimilarity based on TMM-normalized cpm values, performed non-metric multidimensional scaling (nMDS), and conducted permutation-based multivariate analysis of variance (perMANOVA) to test for a global transcriptional effect of pregnancy status, all using the R package vegan [140]. Similarly, to visualize clustering of genes and pouch libraries via co-expression patterns, we generated heatmaps for all pouch-expressed genes and several immune system related KEGG pathways. Ward clustering was used, based on Euclidean distance calculated from scaled, log₂-transformed cpm values, implemented by the R function hclust.

Unless noted otherwise, all additional analyses related to the gene expression were conducted using core packages within the statistical programming language R [141].

Characterization of *patristacins*

Previous work identified members of the astacin-like metalloprotease gene family as candidates for playing a functional role in male pregnancy [18, 52]. We confirmed extreme transcriptional differences for two of these *patristacins* between brood pouch tissue of pregnant and non-pregnant males (see “Differential expression analysis” section) and set out to characterize the distribution of this gene family in the Gulf pipefish and other teleost genomes. We compared protein sequences from pipefish gene annotations bearing similarity to *patristacins* against the Ensembl zebrafish GRCz10 protein set using BLAST and discovered that all similar zebrafish homologs belong to Ensembl protein family ENSFM00500000270265 (choriolytic enzymes). We used all actinopterygian fish sequences from this Ensembl protein family alignment to generate a HMM profile using hmmer2 [129], then searched for similar sequences in the Gulf pipefish genome using the genewisedb program of wise2 (<http://www.ebi.ac.uk/~birney/wise2/>) with default search settings. These protein family-specific annotations allowed us to both correct and supplement initial MAKER annotations as necessary. Most of the *S. scovelli* astacin-like metalloproteases annotated in this manner, including at least four tandemly arrayed *patristacins* on scaffold 62, shared high sequence similarity with zebrafish homologs from Ensembl protein family ENSFM00500000270265. Six of the *S. scovelli* astacin-like metalloproteases were most similar to three additional Ensembl protein families, including ENSFM00500000282854 (Metalloendopeptidases), ENSFM00570000851071 (Bone morphogenetic 1/Tolloid-like proteins), and ENSFM00500000270104 (Meprins).

To identify potential *patristacin* orthologs and/or close paralogs in several teleost genomes, we repeated the HMM search using a hmmer2 profile generated from an alignment of the four pipefish *patristacins*, but included the Gulf pipefish assembly, and the Ensembl genomes of spotted gar, zebrafish, platyfish, and green spotted pufferfish as targets. Hits from these searches were used to understand the evolution of *patristacins* in the syngnathid lineage. Excluding hits that corresponded to the more distantly paralogous Bmp1/Tolloid-like and Merprin proteins [142], with the exception of Meprin1b as an outgroup (see Fig. 7), we aligned all unique astacin-like amino acid sequences from the aforementioned actinopterygii genomes with MAFFT v7 [116] using options –localpair and –maxiterate 1000. We then made manual adjustments to the alignment by removing non-conserved residues at the ends, yielding a final alignment of 55 sequences, covering 269 amino acids. We used the PhyML 3.0 web

server [143] for Akaike Information Criterion model selection and ML phylogenetic inference. The WAG + G + I + F model was selected and we proceeded with two separate evaluations of ML tree clade support: PhyML’s fast SH-like aLRT and 500 bootstrap replicates.

Additional files

Additional file 1: This pdf file contains the following supplementary figures: S1–S8. Legends for these figures are presented at the beginning of Additional file 1. (PDF 3.97 mb)

Additional file 2: This xlsx file contains the following supplementary spreadsheets, each included as a separate tab in a single Microsoft Excel File: SH1–SH7. Descriptions for these spreadsheets are presented at the beginning of Additional file 2. (XLSX 2.47 mb)

Additional file 3: This doc file is a comprehensive summary of Illumina data used for the Gulf pipefish genome project. A diverse collection of short-read sequencing data was used to understand the genome of *S. scovelli*. Included are descriptions of library types, raw read counts, and analyses in which the different libraries were used. (DOCX 90 kb)

Acknowledgements

We thank Ingo Braasch and John Postlethwait for helpful comments early in the project, Mark Currey, Emily Buck, and Nicole Nishimura for laboratory help, Jason Sydes for bioinformatic help, and John Willis, Sarah Flanagan, and Thomas Desvignes for technical consultation. Paula Carlson (Dallas World Aquarium) and Atsushi Sogabe kindly assisted with acquisition of *S. robustus* and *C. haematopterus* specimens.

Funding

This work was funded by an NSF EAGER grant (DEB-1038587) to WAC, an NIH R24 (RR032670) to WAC et al., an NSF grant (DEB-1119261) to AGJ, and an NSF DDIG (DEB-1110709) to CMS and AGJ. CMS was also supported by NIH grant P50GM098911 to WAC et al. AA was also supported in part by NIH R01 (R01OD011116) to J. Postlethwait. SB and JC were supported by NIH NRSA fellowships (F32 GM078949) and (F32GM095213), respectively.

Availability of data and materials

Upon publication, all raw sequencing data described in this study will be available via the NCBI Sequencing Read Archive (SRA) through BioProject ID PRJNA355893 and BioSample accessions SAMN06094711, SAMN06094712, SAMN06094713, SAMN06094714, SAMN06094715, and SAMN06094716. Additional files necessary for understanding the structure and content of the *S. scovelli* genome (e.g. assembly, annotation, map, etc.) will be hosted by the Cresko Lab web server (<http://creskolab.uoregon.edu>).

Authors’ contributions

CMS, SB, and JC are equal co-authors. SB and WAC initiated the project. CMS, SB, AGJ, and WAC obtained funding. CMS, SB, JC, AGJ, and WAC designed the project; CMS, SB, and JC wrote the manuscript; CMS, SB, JC, RB, and AF produced sequencing libraries and/or produced gene annotations; CMS, JC, and AF performed genome assemblies; CMS, SB, JC, AA, and AF analyzed genomic, genetic map, and/or RNA-seq data; JC wrote custom software; RB performed morphological analysis of embryos; AGJ and WAC commented on the manuscript. WAC supervised the overall progress of the project. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Consent for publication

Not applicable.

Ethics approval and consent to participate

All research protocols involving live fish were approved by the Texas A&M University and University of Oregon Institutional Animal Care and Use Committees (2011-51 and 13-07, respectively).

Author details

¹Institute of Ecology and Evolution, University of Oregon, Eugene, OR 97403, USA. ²Present address: Department of Animal Biology, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA. ³Institute of Neuroscience, University of Oregon, Eugene, OR 97403, USA. ⁴Present address: Oregon Health & Science University, Portland, OR 97239, USA. ⁵Department of Biology, Texas A&M University, College Station, TX 77843, USA.

Received: 11 August 2016 Accepted: 5 December 2016

Published online: 20 December 2016

References

- Mayr E. The emergence of evolutionary novelties. In: Tax S, editor. *Evolution After Darwin*, vol. 1. Chicago, IL: The University of Chicago; 1960. p. 349–80.
- Muller GB, Wagner GP. Novelty in evolution - restructuring the concept. *Annu Rev Ecol Syst*. 1991;22:229–56.
- Shubin N, Tabin C, Carroll S. Deep homology and the origins of evolutionary novelty. *Nature*. 2009;457:818–23.
- Wagner GP, Lynch VJ. Evolutionary novelties. *Curr Biol*. 2010;20:R48–52.
- Braasch I, Gehrke AR, Smith JJ, Kawasaki K, Manousaki T, Pasquier J, et al. The spotted gar genome illuminates vertebrate evolution and facilitates human-teleost comparisons. *Nat Genet*. 2016;48:427–37.
- Herald ES. From pipefish to seahorse - a study of phylogenetic relationships. *Proc Calif Acad Sci*. 1959;29:465–73.
- Teske PR, Beheregaray LB. Evolution of seahorses' upright posture was linked to Oligocene expansion of seagrass habitats. *Biol Lett*. 2009;5:521–3.
- Wilson NG, Rouse GW. Convergent camouflage and the non-monophyly of 'seadragons' (Syngnathidae: Teleostei): suggestions for a revised taxonomy of syngnathids. *Zool Scr*. 2010;39:551–8.
- Neutens C, Adriaens D, Christiaens J, De Kegel B, Dierick M, Boistel R, et al. Grasping convergent evolution in syngnathids: a unique tale of tails. *J Anat*. 2014;224:710–23.
- Carcupino M. Functional significance of the male brood pouch in the reproductive strategies of pipefishes and seahorses: a morphological and ultrastructural comparative study on three anatomically different pouches. *J Fish Biol*. 2002;61:1465–80.
- Wilson AB, Ahnesjo I, Vincent AC, Meyer A. The dynamics of male brooding, mating patterns, and sex roles in pipefishes and seahorses (family Syngnathidae). *Evolution*. 2003;57:1374–86.
- Ripley JL. Osmoregulatory role of the paternal brood pouch for two Syngnathus species. *Comp Biochem Physiol A Mol Integr Physiol*. 2009;154:98–104.
- Ripley JL, Foran CM. Direct evidence for embryonic uptake of paternally-derived nutrients in two pipefishes (Syngnathidae: Syngnathus spp.). *J Comp Physiol B*. 2009;179:325–33.
- Jones AG, Walker D, Avise JC. Genetic evidence for extreme polyandry and extraordinary sex-role reversal in a pipefish. *Proc Biol Sci*. 2001;268:2531–5.
- Hoffman EA, Mobley KB, Jones AG. Male pregnancy and the evolution of body segmentation in seahorses and pipefishes. *Evolution*. 2006;60:404–10.
- Paczolt KA, Jones AG. Post-copulatory sexual selection and sexual conflict in the evolution of male pregnancy. *Nature*. 2010;464:401–4.
- Flanagan SP, Johnson JB, Rose E, Jones AG. Sexual selection on female ornaments in the sex-role-reversed Gulf pipefish (Syngnathus scovelli). *J Evol Biol*. 2014;27:2457–67.
- Harlin-Cognato A, Hoffman EA, Jones AG. Gene cooption without duplication during the evolution of a male-pregnancy gene in pipefish. *Proc Natl Acad Sci U S A*. 2006;103:19407–12.
- Ekblom R, Wolf JB. A field guide to whole-genome sequencing, assembly and annotation. *Evol Appl*. 2014;7:1026–42.
- Hardie DC, Hebert PDN. Genome-size evolution in fishes. *Can J Fish Aquat Sci*. 2004;61:1636–46.
- Gnerre S, Maccallum I, Przybylski D, Ribeiro FJ, Burton JN, Walker BJ, et al. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proc Natl Acad Sci U S A*. 2011;108:1513–8.
- Parra G, Bradnam K, Ning Z, Keane T, Korf I. Assessing the gene space in draft genomes. *Nucleic Acids Res*. 2009;37:289–97.
- Valenzano DR, Benayoun BA, Singh PP, Zhang E, Etter PD, Hu CK, et al. The African turquoise killifish genome provides insights into evolution and genetic architecture of lifespan. *Cell*. 2015;163:1539–54.
- McGaugh SE, Gross JB, Aken B, Blin M, Borowsky R, Chalopin D, et al. The cavefish genome reveals candidate genes for eye loss. *Nat Commun*. 2014;5:5307.
- Holt C, Yandell M. MAKER2: an annotation pipeline and genome-database management tool for second-generation genome projects. *BMC Bioinformatics*. 2011;12:491.
- Mank JE, Avise JC. Phylogenetic conservation of chromosome numbers in Actinopterygian fishes. *Genetica*. 2006;127:321–7.
- Vitturi R, Libertini A, Campolmi M, Calderazzo F, Mazzola A. Conventional karyotype, nucleolar organizer regions and genome size in five Mediterranean species of Syngnathidae (Pisces, Syngnathiformes). *J Fish Biol*. 1998;52:677–87.
- Nelson GJ. Phylogeny of major fish groups. In: Fernholm B, Bremer K, Brundin L, Jörnvall H, Rutberg L, Wanntorp HE, editors. *The hierarchy of life: molecules and morphology in phylogenetic analysis: proceedings from Nobel Symposium 70 held at Alfred Nobel's Björkborn, Karlskoga, Sweden, August 29–September 2, 1988*. Amsterdam: Excerpta Medica. Sole distributors for the USA and Canada, Elsevier Science Pub. Co.; 1989. International congress series.
- Betancur RR, Broughton RE, Wiley EO, Carpenter K, Lopez JA, Li C, et al. The tree of life and a new classification of bony fishes. *PLoS Curr*. 2013;5.
- Sanciangco MD, Carpenter KE, Betancur RR. Phylogenetic placement of enigmatic percomorph families (Teleostei: Percomorphaceae). *Mol Phylogenet Evol*. 2016;94:565–76.
- Faircloth BC, McCormack JE, Crawford NG, Harvey MG, Brumfield RT, Glenn TC. Ultraconserved elements anchor thousands of genetic markers spanning multiple evolutionary timescales. *Syst Biol*. 2012;61:717–26.
- Faircloth BC, Sorenson L, Santini F, Alfaro ME. A phylogenomic perspective on the radiation of ray-finned fishes based upon targeted sequencing of ultraconserved elements (UCEs). *PLoS One*. 2013;8, e65923.
- Near TJ, Dornburg A, Eytan RI, Keck BP, Smith WL, Kuhn KL, et al. Phylogeny and tempo of diversification in the superradiation of spiny-rayed fishes. *Proc Natl Acad Sci U S A*. 2013;110:12738–43.
- Edwards SV, Liu L, Pearl DK. High-resolution species trees without concatenation. *Proc Natl Acad Sci U S A*. 2007;104:5936–41.
- Kubatko LS, Degnan JH. Inconsistency of phylogenetic estimates from concatenated data under coalescence. *Syst Biol*. 2007;56:17–24.
- Zakany J, Duboule D. The role of Hox genes during vertebrate limb development. *Curr Opin Genet Dev*. 2007;17:359–66.
- Mallo M, Wellik DM, Deschamps J. Hox genes and regional patterning of the vertebrate body plan. *Dev Biol*. 2010;344:7–15.
- Mallo M, Alonso CR. The regulation of Hox gene expression during animal development. *Development*. 2013;140:3951–63.
- Amores A, Force A, Yan YL, Joly L, Amemiya C, Fritz A, et al. Zebrafish hox clusters and vertebrate genome evolution. *Science*. 1998;282:1711–4.
- Renz AJ, Gunter HM, Fischer JM, Qiu H, Meyer A, Kuraku S. Ancestral and derived attributes of the dlx gene repertoire, cluster structure and expression patterns in an African cichlid fish. *EvoDevo*. 2011;2:1.
- Mayor C, Brudno M, Schwartz JR, Poliakov A, Rubin EM, Frazer KA, et al. VISTA: visualizing global DNA sequence alignments of arbitrary length. *Bioinformatics*. 2000;16:1046–7.
- Frazer KA, Pachter L, Poliakov A, Rubin EM, Dubchak I. VISTA: computational tools for comparative genomics. *Nucleic Acids Res*. 2004;32:W273–9.
- Ghanem N, Jarinova O, Amores A, Long Q, Hatch G, Park BK, et al. Regulatory roles of conserved intergenic domains in vertebrate Dlx bigene clusters. *Genome Res*. 2003;13:533–43.
- Marciel A. Pitx1 and Pitx2 are required for development of hindlimb buds. *Development*. 2003;130:45–55.
- Naiche LA. Loss of Tbx4 blocks hindlimb development and affects vascularization and fusion of the allantois. *Development*. 2003;130:2681–93.
- Don EK, de Jong-Curtain TA, Doggett K, Hall TE, Heng B, Badrock AP, et al. Genetic basis of hindlimb loss in a naturally occurring vertebrate model. *Biol Open*. 2016;5:359–66.
- Semina EV, Ferrell RE, Mintz-Hittner HA, Bitoun P, Alward WLM, Reiter RS, et al. A novel homeobox gene PITX3 is mutated in families with autosomal-dominant cataracts and ASMD. *Nat Genet*. 1998;19:167–70.
- Shi X, Bosenko DV, Zinkevich NS, Foley S, Hyde DR, Semina EV, et al. Zebrafish pitx3 is necessary for normal lens and retinal development. *Mech Dev*. 2005;122:513–27.

49. Humphrey JH, Dourmashkin RR. The lesions in cell membranes caused by complement. *Adv Immunol.* 1969;11:75–115.
50. McCormack R, de Armas L, Shiratsuchi M, Podack ER. Killing machines: three pore-forming proteins of the immune system. *Immunol Res.* 2013;57:268–78.
51. Haase D, Roth O, Kalbe M, Schmiedeskamp G, Scharsack JP, Rosenstiel P, et al. Absence of major histocompatibility complex class II mediated immunity in pipefish, *Syngnathus typhle*: evidence from deep transcriptome sequencing. *Biol Lett.* 2013;9:20130044.
52. Small CM, Harlin-Cognato AD, Jones AG. Functional similarity and molecular divergence of a novel reproductive transcriptome in two male-pregnant Syngnathus pipefish species. *Ecol Evol.* 2013;3:4092–108.
53. Naruse K, Tanaka M, Mita K, Shima A, Postlethwait J, Mitani H. A medaka gene map: the trace of ancestral vertebrate proto-chromosomes revealed by comparative gene mapping. *Genome Res.* 2004;14:820–8.
54. Whittington CM, Griffith OW, Qi W, Thompson MB, Wilson AB. Seahorse brood pouch transcriptome reveals common genes associated with vertebrate pregnancy. *Mol Biol Evol.* 2015;32:3114–31.
55. Boettcher M, McManus MT. Choosing the right tool for the job: RNAi, TALEN, or CRISPR. *Mol Cell.* 2015;58:575–85.
56. Sternberg SH, Doudna JA. Expanding the biologist's toolkit with CRISPR-Cas9. *Mol Cell.* 2015;58:568–74.
57. Alexander T, Nolte C, Krumlauf R. Hox genes and segmentation of the hindbrain and axial skeleton. *Annu Rev Cell Dev Biol.* 2009;25:431–56.
58. Cruz C, Maegawa S, Weinberg ES, Wilson SW, Dawid IB, Kudoh T. Induction and patterning of trunk and tail neural ectoderm by the homeobox gene eve1 in zebrafish embryos. *Proc Natl Acad Sci U S A.* 2010;107:3564–9.
59. Seebald JL, Szeto DP. Zebrafish eve1 regulates the lateral and ventral fates of mesodermal progenitor cells at the onset of gastrulation. *Dev Biol.* 2011;349:78–89.
60. Laurenti P, Thaeron C, Allizard F, Huysseune A, Sire JY. Cellular expression of eve1 suggests its requirement for the differentiation of the ameloblasts and for the initiation and morphogenesis of the first tooth in the zebrafish (*Danio rerio*). *Dev Dyn.* 2004;230:727–33.
61. Debiais-Thibaudeau M, Borday-Birraux V, Germon I, Bourrat F, Metcalfe CJ, Casane D, et al. Development of oral and pharyngeal teeth in the medaka (*Oryzias latipes*): comparison of morphology and expression of eve1 gene. *J Exp Zool B Mol Dev Evol.* 2007;308:693–708.
62. Chen F, Greer J, Capecchi MR. Analysis of Hoxa7/Hoxb7 mutants suggests periodicity in the generation of the different sets of vertebrae. *Mech Dev.* 1998;77:49–57.
63. Tanaka M, Hale LA, Amores A, Yan YL, Cresko WA, Suzuki T, et al. Developmental genetic basis for the evolution of pelvic fin loss in the pufferfish *Takifugu rubripes*. *Dev Biol.* 2005;281:227–39.
64. Shapiro MD, Marks ME, Peichel CL, Blackman BK, Nereng KS, Jonsson B, et al. Genetic and developmental basis of evolutionary pelvic reduction in threespine sticklebacks. *Nature.* 2004;428:717–23.
65. Chan YF, Marks ME, Jones FC, Villarreal Jr G, Shapiro MD, Brady SD, et al. Adaptive evolution of pelvic reduction in sticklebacks by recurrent deletion of a Pitx1 enhancer. *Science.* 2010;327:302–5.
66. Logan M, Tabin CJ. Role of Pitx1 upstream of Tbx4 in specification of hindlimb identity. *Science.* 1999;283:1736–9.
67. Naiche LA, Papaioannou VE. Tbx4 is not required for hindlimb identity or post-bud hindlimb outgrowth. *Development.* 2007;134:93–103.
68. Brouwer A, ten Berge D, Wiegerink R, Meijlink F. The OAR/aristaless domain of the homeodomain protein Cart1 has an attenuating role in vivo. *Mech Dev.* 2003;120:241–52.
69. Brown LY, Brown SA. Alanine tracts: the expanding story of human illness and trinucleotide repeats. *Trends Genet.* 2004;20:51–8.
70. Galant R, Carroll SB. Evolution of a transcriptional repression domain in an insect Hox protein. *Nature.* 2002;415:910–3.
71. Oma Y, Kino Y, Sasagawa N, Ishiiura S. Intracellular localization of homopolymeric amino acid-containing proteins expressed in mammalian cells. *J Biol Chem.* 2004;279:21217–22.
72. Kawahara R, Miya M, Mabuchi K, Lavoué S, Inoue JG, Satoh TP, et al. Interrelationships of the 11 gasterosteiform families (sticklebacks, pipefishes, and their relatives): a new perspective based on whole mitogenome sequences from 75 higher teleosts. *Mol Phylogenet Evol.* 2008;46:224–36.
73. Hamilton H, Saarman N, Short G, Sellas AB, Moore B, Hoang T, et al. Molecular phylogeny and patterns of diversification in Syngnathid fishes. *Mol Phylogenet Evol.* 2016. doi:10.1016/j.ympev.2016.10.003.
74. Playfair RL, Günther ACLG. The fishes of Zanzibar, with a list of the fishes of the whole East coast of Africa. London: John van Voorst; 1866.
75. MacDonald RB, Debiais-Thibaudeau M, Talbot JC, Ekker M. The relationship between dlx and gad1 expression indicates highly conserved genetic pathways in the zebrafish forebrain. *Dev Dyn.* 2010;239:2298–306.
76. Qiu M, Bulfone A, Martinez S, Meneses JJ, Shimamura K, Pedersen RA, et al. Null mutation of Dlx-2 results in abnormal morphogenesis of proximal first and second branchial arch derivatives and abnormal differentiation in the forebrain. *Genes Dev.* 1995;9:2523–38.
77. Sperber SM, Saxena V, Hatch G, Ekker M. Zebrafish dlx2a contributes to hindbrain neural crest survival, is necessary for differentiation of sensory ganglia and functions with dlx1a in maturation of the arch cartilage elements. *Dev Biol.* 2008;314:59–70.
78. Jackman WR, Draper BW, Stock DW. Fgf signaling is required for zebrafish tooth development. *Dev Biol.* 2004;274:139–57.
79. Stock DW, Jackman WR, Trapani J. Developmental genetic mechanisms of evolutionary tooth loss in cypriniform fishes. *Development.* 2006;133:3127–37.
80. Fraser GJ, Hulsey CD, Bloomquist RF, Uyesugi K, Manley NR, Streelman JT. An ancient gene network is co-opted for teeth on old and new jaws. *PLoS Biol.* 2009;7, e31.
81. Akimenko MA, Ekker M, Wegner J, Lin W, Westerfield M. Combinatorial expression of three zebrafish genes related to distal-less: part of a homeobox gene code for the head. *J Neurosci.* 1994;14:3475–86.
82. Watanabe S, Kaneko T, Watanabe Y. Immunocytochemical detection of mitochondria-rich cells in the brood pouch epithelium of the pipefish, *Syngnathus schlegeli*: structural comparison with mitochondria-rich cells in the gills and larval epidermis. *Cell Tissue Res.* 1999;295:141–9.
83. Laksanawimol P, Damrongphol P, Kruatrachue M. Alteration of the brood pouch morphology during gestation of male seahorses, *Hippocampus kuda*. *Mar Freshw Res.* 2006;57:497–502.
84. Goncalves IB, Ahnesjo I, Kvarnemo C. Embryo oxygenation in pipefish brood pouches: novel insights. *J Exp Biol.* 2015;218:1639–46.
85. Roth O, Klein V, Beemelmanns A, Scharsack JP, Reusch TBH. Male pregnancy and biparental immune priming. *Am Nat.* 2012;180:802–14.
86. Ohno S. Evolution by gene duplication. Heidelberg, Germany: Springer-Verlag; 1970.
87. Force A, Lynch M, Pickett FB, Amores A, Yan YL, Postlethwait J. Preservation of duplicate genes by complementary, degenerative mutations. *Genetics.* 1999;151:1531–45.
88. Lan X, Pritchard JK. Coregulation of tandem duplicate genes slows evolution of subfunctionalization in mammals. *Science.* 2016;352:1009–13.
89. Kawaguchi M, Yasumasu S, Hiroi J, Naruse K, Inoue M, Iuchi I. Evolution of teleostean hatching enzyme genes and their paralogous genes. *Dev Genes Evol.* 2006;216:769–84.
90. Kijimoto T, Watanabe M, Fujimura K, Nakazawa M, Murakami Y, Kuratani S, et al. cimp1, a novel astacin family metalloproteinase gene from East African cichlids, is differentially expressed between species during growth. *Mol Biol Evol.* 2005;22:1649–60.
91. Catchen JM, Amores A, Hohenlohe P, Cresko W, Postlethwait JH. Stacks: building and genotyping Loci de novo from short-read sequences. *G3 (Bethesda).* 2011;1:171–82.
92. Catchen J, Hohenlohe PA, Bassham S, Amores A, Cresko WA. Stacks: an analysis tool set for population genomics. *Mol Ecol.* 2013;22:3124–40.
93. Simpson JT, Durbin R. Efficient de novo assembly of large genomes using compressed data structures. *Genome Res.* 2012;22:549–56.
94. Zerbino DR, Birney E. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.* 2008;18:821–9.
95. Boetzer M, Henkel CV, Jansen HJ, Butler D, Pirovano W. Scaffolding pre-assembled contigs using SSPAGE. *Bioinformatics.* 2011;27:578–9.
96. Magoc T, Salzberg SL. FLASH: fast length adjustment of short reads to improve genome assemblies. *Bioinformatics.* 2011;27:2957–63.
97. Grabherr MG, Haas BJ, Yassour M, Levin JZ, Thompson DA, Amit I, et al. Full-length transcriptome assembly from RNA-Seq data without a reference genome. *Nat Biotechnol.* 2011;29:644–52.
98. RepeatMasker Open-4.0.5. <http://www.repeatmasker.org>.
99. RepeatModeler Open-1.0.8. <http://www.repeatmasker.org>.
100. Jurka J, Kapitonov VV, Pavlicek A, Klonowski P, Kohany O, Walichiewicz J. Repbase Update, a database of eukaryotic repetitive elements. *Cytogenet Genome Res.* 2005;110:462–7.
101. Korf I. Gene finding in novel genomes. *BMC Bioinformatics.* 2004;5:59.

102. Stanke M, Waack S. Gene prediction with a hidden Markov model and a new intron submodel. *Bioinformatics*. 2003;19 Suppl 2:i215–225.1.
103. Cantarel BL, Korf I, Robb SM, Parra G, Ross E, Moore B, et al. MAKER: an easy-to-use annotation pipeline designed for emerging model organism genomes. *Genome Res.* 2008;18:188–96.
104. Cunningham F, Amode MR, Barrell D, Beal K, Billis K, Brent S, et al. Ensembl 2015. *Nucleic Acids Res.* 2015;43:D662–9.
105. Quevillon E, Silventoinen V, Pillai S, Harte N, Mulder N, Apweiler R, et al. InterProScan protein domains identifier. *Nucleic Acids Res.* 2005;33:W116–20.
106. Baird NA, Etter PD, Atwood TS, Currey MC, Shiver AL, Lewis ZA, et al. Rapid SNP discovery and genetic mapping using sequenced RAD markers. *PLoS One*. 2008;3, e3376.
107. Hohenlohe PA, Bassham S, Etter PD, Stiffler N, Johnson EA, Cresko WA. Population genomics of parallel adaptation in threespine stickleback using sequenced RAD tags. *PLoS Genet.* 2010;6, e1000862.
108. Etter PD, Bassham S, Hohenlohe PA, Johnson EA, Cresko WA. SNP discovery and genotyping for evolutionary genetics using RAD sequencing. *Methods Mol Biol.* 2011;772:157–78.
109. Hohenlohe PA, Bassham S, Currey M, Cresko WA. Extensive linkage disequilibrium and parallel adaptive divergence across threespine stickleback genomes. *Philos Trans R Soc Lond B Biol Sci.* 2012;367:395–408.
110. Van Ooijen JW. JoinMap® 4, Software for the calculation of genetic linkage maps in experimental populations. Kyazma BV, Wageningen. 2006;33:10–1371.
111. Wu TD, Nacu S. Fast and SNP-tolerant detection of complex variants and splicing in short reads. *Bioinformatics*. 2010;26:873–81.
112. Catchen JM, Conery JS, Postlethwait JH. Automated identification of conserved synteny after whole-genome duplication. *Genome Res.* 2009;19: 1497–505.
113. Rice P, Longden I, Bleasby A. EMBOSS: The European Molecular Biology Open Software Suite. *Trends Genet.* 2000;16(6):276–77.
114. Harris R. Improved pairwise alignment of genomic DNA. ProQuest. 2007.
115. Quinlan AR. BEDTools: The Swiss-Army tool for genome feature analysis. *Curr Protoc Bioinformatics*. 2014;11.12:1–34.
116. Katoh K, Standley DM. MAFFT multiple sequence alignment software version 7: improvements in performance and usability. *Mol Biol Evol.* 2013; 30:772–80.
117. Guindon S, Gascuel O. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Syst Biol.* 2003;52:696–704.
118. Darriba D, Taboada GL, Doallo R, Posada D. jModelTest 2: more models, new heuristics and parallel computing. *Nat Methods*. 2012;9:772.
119. Stamatakis A. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*. 2014;30:1312–3.
120. Stamatakis A, Hoover P, Rougemont J. A rapid bootstrap algorithm for the RAxML Web servers. *Syst Biol.* 2008;57:758–71.
121. Nakamura Y, Mori K, Saitoh K, Oshima K, Mekuchi M, Sugaya T, et al. Evolutionary changes of multiple visual pigment genes in the complete genome of Pacific bluefin tuna. *Proc Natl Acad Sci.* 2013;110:11061–6.
122. Bray N, Dubchak I, Pachter L. AVID: A global alignment program. *Genome Res.* 2003;13:97–102.
123. Brudno M, Do CB, Cooper GM, Kim MF, Davydov E, NISC Comparative Sequencing Program, et al. LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA. *Genome Res.* 2003;13:721–31.
124. Brudno M, Malde S, Poliakov A, Do CB, Couronne O, Dubchak I, et al. Glocal alignments: finding rearrangements during alignment. *Bioinformatics*. 2003; 19 Suppl 1:i54–62.
125. Kozomara A, Griffiths-Jones S. miRBase: integrating microRNA annotation and deep-sequencing data. *Nucleic Acids Res.* 2011;39:D152–7.
126. Edgar RC. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res.* 2004;32:1792–7.
127. BBMap version 35. <http://jgi.doe.gov/data-and-tools/bbtools/bb-tools-user-guide/bbmap-guide/>.
128. Karplus K, Barrett C, Hughey R. Hidden Markov models for detecting remote protein homologies. *Bioinformatics*. 1998;14:846–56.
129. Johnson LS, Eddy SR, Portugaly E. Hidden Markov model speed heuristic and iterative HMM search procedure. *BMC Bioinformatics*. 2010;11:431.
130. Anders S, Pyl PT, Huber W. HTSeq—a Python framework to work with high-throughput sequencing data. *Bioinformatics*. 2015;31:166–9.
131. Robinson MD, Smyth GK. Small-sample estimation of negative binomial dispersion, with applications to SAGE data. *Biostatistics*. 2008;9:321–32.
132. Robinson MD, McCarthy DJ, Smyth GK. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*. 2010;26:139–40.
133. Kanehisa M, Sato Y, Kawashima M, Furumichi M, Tanabe M. KEGG as a reference resource for gene and protein annotation. *Nucleic Acids Res.* 2016;44:D457–62.
134. Moriya Y, Itoh M, Okuda S, Yoshizawa AC, Kanehisa M. KAAS: an automatic genome annotation and pathway reconstruction server. *Nucleic Acids Res.* 2007;35:W182–5.
135. Luo W, Friedman MS, Shedden K, Hankenson KD, Woolf PJ. GAGE: generally applicable gene set enrichment for pathway analysis. *BMC Bioinformatics*. 2009;10:161.
136. Luo W, Brouwer C. Pathview: an R/Bioconductor package for pathway-based data integration and visualization. *Bioinformatics*. 2013;29:1830–1.
137. Mi H, Muruganujan A, Casagrande JT, Thomas PD. Large-scale gene function analysis with the PANTHER classification system. *Nat Protoc.* 2013;8: 1551–66.
138. Mi H, Poudel S, Muruganujan A, Casagrande JT, Thomas PD. PANTHER version 10: expanded protein families and functions, and analysis tools. *Nucleic Acids Res.* 2016;44:D336–42.
139. Benjamini Y, Hochberg Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *J R Stat Soc Ser B Methodol.* 1995;57:289–300.
140. Oksanen J, Blanchet FG, Kindt R, Legendre P, Minchin PR, O'Hara RB, et al. vegan: Community Ecology Package. 2015; R package version 2.3–5.
141. Team RC. R: A language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing; 2015. <http://www.R-project.org/>.
142. Mohrlen F, Maniura M, Plickert G, Frohme M, Frank U. Evolution of astacin-like metalloproteases in animals and their function in development. *Evolution Development*. 2006;8:223–31.
143. Guindon S, Dufayard JF, Lefort V, Anisimova M, Hordijk W, Gascuel O. New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of PhyML 3.0. *Syst Biol.* 2010;59:307–21.
144. Leysen H, Jouk P, Brunain M, Christiaens J, Adriaens D. Cranial architecture of tube-snouted gasterosteiformes (*Syngnathus rostellatus* and *Hippocampus capensis*). *J Morphol.* 2010;271:255–70.
145. Leysen H, Christiaens J, De Kegel B, Boone MN, Van Hoorebeke L, Adriaens D. Musculoskeletal structure of the feeding system and implications of snout elongation in *Hippocampus reidi* and *Dunckerocampus dactyliophorus*. *J Fish Biol.* 2011;78:1799–823.

Submit your next manuscript to BioMed Central and we will help you at every step:

- We accept pre-submission inquiries
- Our selector tool helps you to find the most relevant journal
- We provide round the clock customer support
- Convenient online submission
- Thorough peer review
- Inclusion in PubMed and all major indexing services
- Maximum visibility for your research

Submit your manuscript at
www.biomedcentral.com/submit



WEEK 11 READINGS

WEEK 12 READINGS

WEEK 13 READINGS

WEEK 14 READINGS

WEEK 15 READINGS

Microarray Integrated Analysis of a Gene Network for the CD36 Myocardial Phenotype

Imane Sabaouni^{1,*}, Brigitte Vannier², Ahmed Moussa³ & Azeddine Ibrahimi¹

¹Medical Biotechnology Lab (MedBiotech), Rabat Medical & Pharmacy School, Mohammed Vth University in Rabat, Morocco;

²Receptors, Regulation and Tumor Cells (2RTC) Laboratory, University of Poitiers, France; ³LabTIC Laboratory, ENSA, Abdelmalek Essaadi University, Tangier, Morocco; Imane SABAOUNI - E-mail: imanesabaouni@yahoo.com *Corresponding author

Received July 18, 2016; Revised July 29, 2016; Accepted July 30, 2016; Published October 11, 2016

Abstract:

CD36 is a multifunctional membrane-type receptor glycoprotein that reacts with oxidized low-density lipoprotein and long-chain fatty acid (LCFA). However, much remains to be understood about the molecular mechanism of the cardio-myopathy observed in CD36-KO mice. In this study, we identify different genes pathways involved in response to CD36 cardio-myopathy phenotype by identifying the differences among biological processes, molecular pathways and networks of interactions that emerge from knocking CD3 and using different bioinformatics tools such as STRING, GeneMANIA and Cytoscape. We were able list all the CD36-regulated genes, their related function and their specific networks. Data analysis showed that CD36-regulated genes differentially expressed are involved in biological processes such as FA metabolism, angiogenesis/apoptosis and cell structure. These results provide the first look at mechanisms involved in CD36 deficiency and development of cardio-myopathy and the opportunity to identify new therapeutic targets.

Keywords: CD36, cardio-myopathy, genes networks, genes pathways, metabolism, angiogenesis/apoptosis.

Background:

Hypertrophic cardio-myopathy ("HCM") is characterized by a myocardium hypertrophy. To date the molecular mechanisms underlying this pathology remain elusive [1-6]. It is most well known as a leading cause of sudden cardiac death in young athletes [7]. The occurrence of hypertrophic cardio-myopathy is a significant cause of sudden unexpected cardiac death in any age group and is a cause of disabling cardiac symptoms. Younger people are likely to have a more severe form of hypertrophic cardio-myopathy. HCM is frequently asymptomatic until sudden cardiac death. As a consequence, re-current screening has been suggested for certain populations [8].

The CD36 gene encodes a membrane glycoprotein (also known as platelet glycoprotein IV or IIIb), that is expressed in a wide variety cell types, including platelets, monocytes, erythroblasts, capillary endothelial cells and mammary epithelial cells [9-10]. In the heart CD36 is also expressed both in epithelial and muscle cells [11]. In these latter, CD36 has been shown to be a long chain fatty acids (FA) receptor/transporter [12]. In endothelial cells, it has been proposed as one of thrombospondin receptor [13]. CD36 is a multifunctional membrane-type receptor glycoprotein that reacts with thrombospondin, collagen, oxidized low-density lipoprotein and long-chain fatty acid (LCFA).1,2 LCFA is one of

the major cardiac energy substrates; hence, LCFA metabolism may have an important role in cardiac diseases. We present here a patient with hereditary hypertrophic cardiomyopathy (HCM) and type I CD36 deficiency that showed no myocardial LCFA accumulation.

CD36-KO mice exhibit between 60 and 80% reduction in beta-methyl-p-[123I]-iodophenyl-pentadecanoic acid (BMIPP) uptake by heart tissue [14] which is paralleled by an increase in the heart/body index and the left ventricular size [13]. We propose in this study to define the molecular defects that lead to cardiac hypertrophy and consequent of the fatty acid transporter CD36 deficiency. To define this molecular defect, we used two microarrays technologies (Affymetrix, Agilent) in order to identify genetic alterations related to the myocardial phenotype of CD36-Ko mice.

In a previous study [15], heterogeneity of the arrays data was analyzed by splitting them on three levels: genes, genes set, and network/pathway. We were able to identify the CD36-regulated candidate genes linked to Hypertrophic cardio-myopathy. At the second level we were able to cluster group of genes (gene sets) that may have similar functions [15]. In this study, we aim to ascertain whether each gene set from each subtype is significantly

enriched in a list of selected phenotypes. The third level of analysis was to construct gene networks of the proposed CD36 regulated genes from three selected web-based tools: Ingenuity Pathway Analysis (IPA), Search Tool for The Retrieval of Interacting Genes (STRING), and Gene Multiple Association Network Integration Algorithm (GeneMANIA).

Materials and Methods:

Dataset:

To identify a comprehensive set of genes that are differentially regulated by CD36 expression in the heart, we used two microarray technologies (Affymetrix and Agilent) to compare gene expression in heart tissues from CD36 Knock-K-Out (KO-CD36) versus wild type (WT-CD36) mice. The obtained results using the two technologies were similar with around 35 genes differentially expressed using both technologies [15]. Absence of CD36 led to down-regulation of the expression of three groups of genes involved in pathways of FA metabolism, angiogenesis/apoptosis and structure. These data are consistent with the fact that the CD36 protein binds FA and thrombospondin 1 involved respectively in lipid metabolism and anti-angiogenic activities [15]. Summary of dataset analysis methodology used in this study is shown in **Figure 1**.

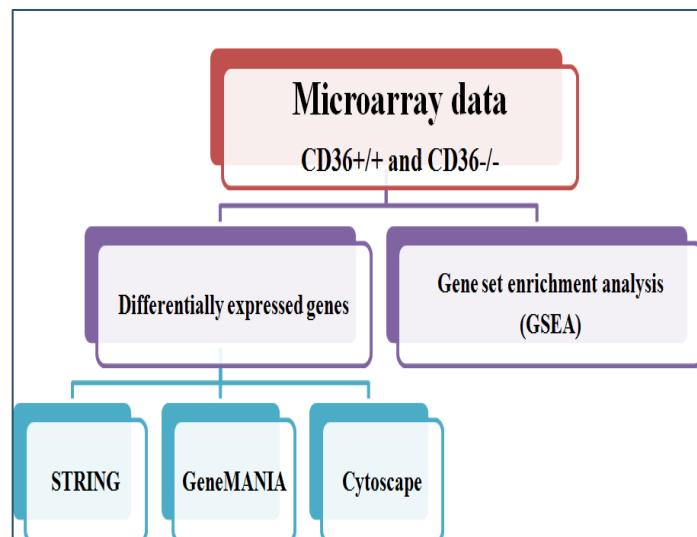


Figure 1: Summary of dataset analysis methodology used in this study

Gene Set Enrichment Analysis and Enrichment Map

Gene set enrichment analyses (GSEA) [16, 17] are commonly used to determine the biological characterization, statistical significance, and concordant differences between an experimental gene set and a selected gene list from annotated gene sets knowledge bases to red on Molecular Signatures Database (MSigDB). GSEA can be downloaded from <http://www.broadinstitute.org/gsea/downloads.jsp>. The Jaccard coefficient is used to compare the similarity between two sample gene sets A and B and defined as the intersection between group A and B divided by their union. The results from GSEA are then visualized through the enrichment map [18], a Cytoscape plugin for network visualization. The ranked experimental gene list

along with the enriched gene sets from GSEA is used to build the network of gene sets (nodes) where edges represent their similarity. The size of a node varies by gene set size and the thickness of the edge represents the degree of correlation between two gene sets.

Networks and Pathway Analysis

Analysis of Network Invoked by CD36-Regulated Genes

The biological knowledge of gene and protein interactions is growing rapidly and there are many tools and curated databases available on a large scale. Insightful knowledge gained from studying gene sets rather than individual genes using network-based approaches can reveal network patterns and relevant molecular pathways from the experiment gene sets. In this study, we utilized two different freely accessible and user-friendly web tools as follows. Gene Multiple Association Network Integration Algorithm (GeneMANIA) [19, 20] (<http://www.genemania.org/>) is a web-based tool for prediction of gene function or implemented as a Cytoscape plugin tool. Based on single gene or gene set query from 7 organisms, it shows results for interactive functional associative network according to their co-expression data from Gene Expression Omnibus (GEO), physical and genetic interaction data derived from BioGRID, predicted protein interaction data based on orthology from I2D, co-localization, shared protein domain, and GO function. Search Tool for the Retrieval of Interacting Genes (STRING) version 9.1 [21, 22] (<http://string-db.org/>) is an online protein-protein interaction database curated from literature and predicted associations from systemic genome comparisons. The user can query using single or multiple name(s) and protein sequence(s). The protein interactions can be displayed according to their confidence, evidence, actions, or interactions.

Results and Discussion:

Identification of CD36-Regulated Genes through a Refined Analysis of Data

Our initial analysis of this time series gene expression data for differentially expressed gene identification followed the same method used in a previous paper [15]. All files were processed and normalized by Robust Multiarray Average (RMA) in R as in the original study. The selected normalization method may have an effect on downstream analysis, for example, reverse engineering analyses [15]; however, investigating this effect is beyond the scope of this study. We found that combining CD36+/+ and CD36-/ data compromises the accuracy of selection of differentially expressed. **Table 1** summarizes the 30 differentially expressed genes from the combined dataset.

Networks and Pathways Analysis

Network analysis can help understand the molecular and cellular interactions [23]. It can be visualized to represent entities (nodes) and their relationships (arcs). The advent of high-throughput technology has led to a large increase in publicly available information. Each data type can capture different aspect of functional roles of interested genes. In this section, we investigated functional interaction among genes and proteins in the cell using available data and knowledge bases.

We selected two different web-based network tools: GeneMANIA and STRING using the differentially expressed genes as a query gene sets. These toolsets integrates computational methods to predict the gene functions based on a collection of interaction networks. GeneMANIA is a large collection of interaction networks from several data sources which identify genes and networks that are functionally associated (protein and genetic interactions, pathways, coexpression, colocalization, and protein domain similarity) with the query gene sets. Another advantage is that the user can run this as a plugin with the Cytoscape tool allowing the user to apply other tools to analyze the networks (**Figure 2**). STRING relies on the phylogeny to infer the functional interaction (protein networks) with direct interaction to score nodes, while GeneMANIA uses functional genomic data with label propagation to score nodes and generate gene networks (**Figure 3**). STRING uses precomputed networks, while those of GeneMANIA are not precomputed, and user can upload their own networks. STRING covers a large number of organisms, while GeneMANIA only covers 7 organisms but allows the user to upload or add more networks through the plugin. In addition, users can run enrichment analysis (GO, KEGG, PFAM, INTERPRO, and protein-protein interaction) on STRING (**Figure 4**).

The results of genes and network that are functionally associated with the gene set from early response of CD36 are shown in **Figures 3 and 4**. We compared the two networks from STRING and GeneMANIA based on the interactions they revealed; here we used CD36 as the centre gene in the comparison. All interactions found in STRING were found in GeneMANIA as described in more detail in **Table 2** (Cf. Annex). Comparison between the results from both tools can be used to confirm the functional associations of the interested gene sets. There is evidence of overlap and uniqueness in the interactions revealed by the two web-based tools.

Discussion

In this report, we were able to show that our data is consistent with the role CD36 protein in the FA metabolism, angiogenesis/apoptosis and structure. In cardiac muscle cells, CD36 is known for its lead role as a receptor / transporter of long-chain fatty acids in heart cells and involvement in metabolism in general. Indeed, the ability of cardiomyocytes to adapt its energy demand is a determinant for myocardial function and the Fatty acids are the main energy source of the heart with oxygen. Thus, the CD36 absence results in a loss of absorption of long chain fatty acids and cause deleterious effects on the heart muscle, while a decrease in the availability of acyl-CoA causes inhibition of B-oxidation. This leads to a decrease in free carnitine and accumulation of intra-mitochondrial acyl-carnitine long chain and an important membrane alteration. These disturbances result in a significant decrease in energy production as ATP, which causes the use of an alternative energy source such as glucose. In 1963, Randle and his colleagues [24], have brought to light, using *in vitro* experiments on rat isolated and perfused hearts, the existence of competition between glucose and fatty acids with long chain fatty acids being preferentially metabolized. Moreover, according to Dyck *et al.* [25], CD36 plays an important role in the choice of substrate in

ISSN 0973-2063 (online) 0973-8894 (print)

the heart [25]. WT-CD36 which normally draws more of its energy from fatty acids is forced to uses more glucose in the absence of CD36 (The heart of the CD36-KO). Our work confirms these results as we see the over-expression of genes that stimulate glucose metabolism such as IRS1, IRS2, IRS3, IDE etc.

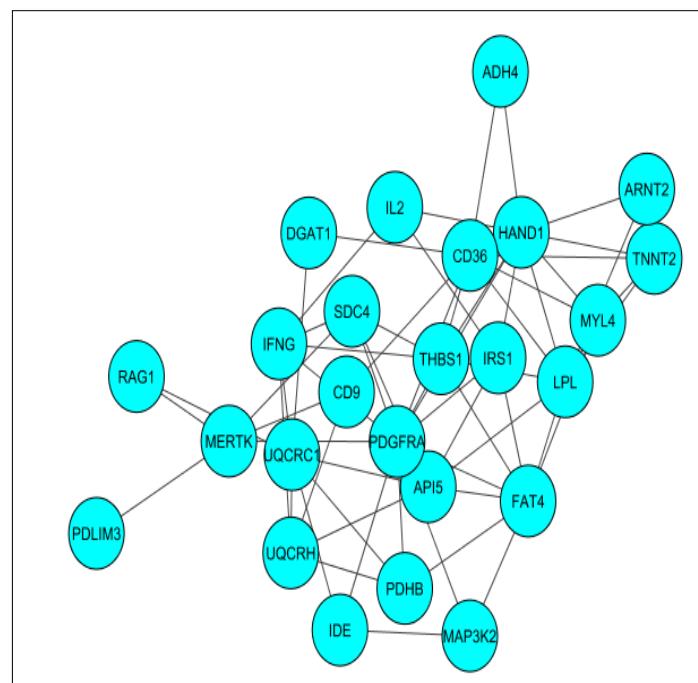


Figure 2: Interaction network Visualization using Cytoscape². Cytoscape and enrichment map were used for visualization of the results; only gene sets from Gene Ontology were used. Nodes represent enriched GO gene sets, whose size reflects the total number of genes in that gene set. Edge thickness represents the number of overlapping genes between gene sets calculated using Jaccard coefficient.

On the other hand, in endothelial cells where angiogenesis/apoptosis are critical mechanisms for cell survival, CD36 has been described as a thrombospondin membrane receptor (TSP-1) [26]. TSP-1 plays a role in the inhibition of endothelial cell migration and apoptosis induction. Observations indicate that CD36 does not contain the motif CxxC shown to be necessary for the LCK-CD4 association, which allows to think that other proteins can be associated to CD36 to facilitate signal transduction induced by the CD36_TSP1 binding [26].

Our results confirmed that showing that CD9 that encodes the CD9 protein and CD36 share the same expression profiles. Studies on platelets (for solubilization of the platelet membrane) identify CD9; a11b3 and a6b11 integrins are CD36 partners [27]. Immunofluorescence studies show that CD9 and CD36 alpha6beta1 are co-located in endothelial cells. Thus it suggests that CD9, CD36, and alpha 6 beta 1 might form a complex in endothelial cells upon the binding of TSP1. Our data also showed a differential expression of MAP3K2 (also known as MEKK2), a member of the serine / threonine kinase family, and

an activator of P38 which in turns activates a cascade that results in increased cell size [28]. In monocytic cells, the same results were observed since it was determined the existence of a physical association between the intra-cytoplasmic carboxyl terminus domain of CD36 and a signaling complex containing Lyn and MEKK2, upstream of the P38 MAP kinase cascade was de [29].

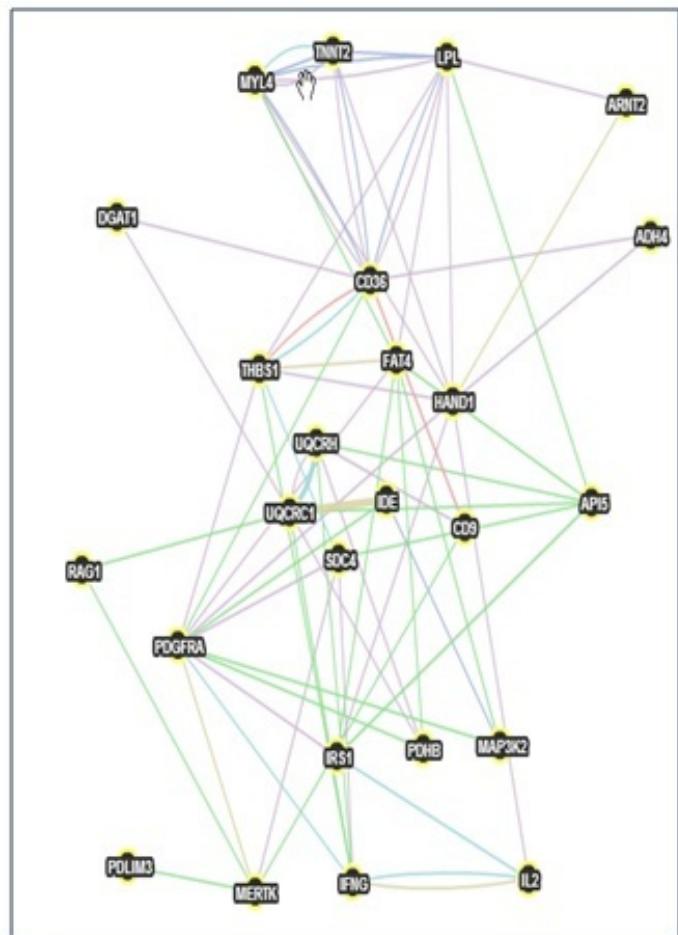


Figure 3: Gene network of CD36 derived from GeneMANIA. A gene network from GeneMANIA shows the relationships for genes from the list (nodes) connected (with edges) according to the functional association networks from the databases.

Data analysis showed that many Structural genes were altered in CD36-KO. Indeed, failing heart differs from the normal heart in function as well as in structure as failing heart is most often remodeled with hypertrophy. Cardiac imaging with the increases of the ventricular wall thickness and smaller ventricular chambers can clinically recognize hypertrophy. Our work confirms cardiac such remodeling as we see the expression of genes that stimulate cardiac cell structure such as: MYL4, TNNT2, HAND1, PDLIM3 AND PDGFRA.

References

- [1] Richardson P, et al. 1996 Circulation 93(5): 841 [PMID 8598070]
- [2] Maron BJ, 2002 JAMA 287(10): 1308 [PMID 11886323]

ISSN 0973-2063 (online) 0973-8894 (print)

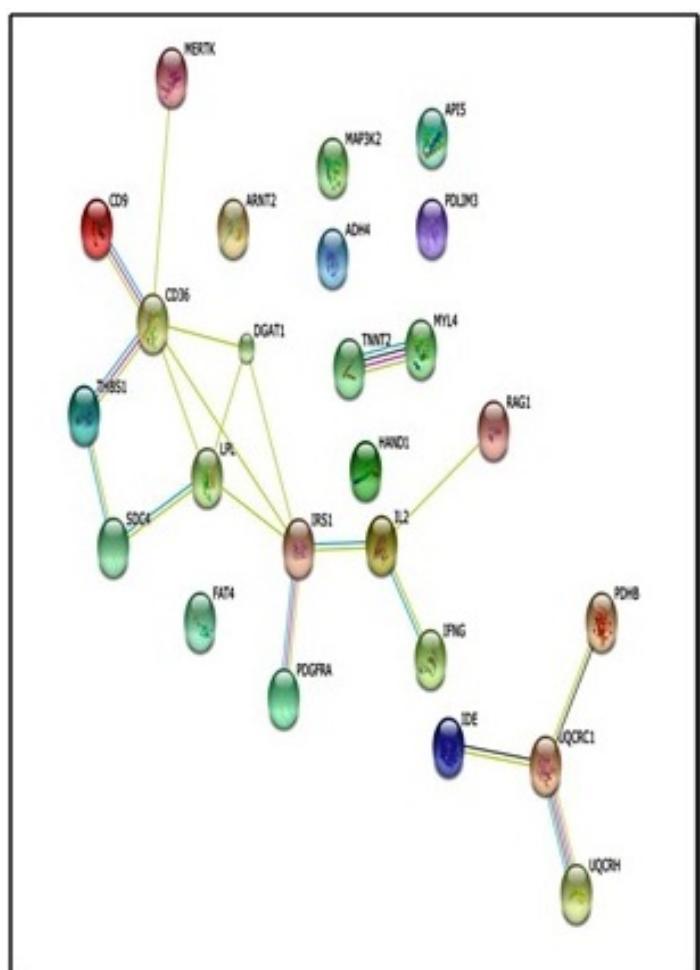


Figure 4: Results from STRING search of Protein interaction for CD36. The figure illustrates the protein interaction upon querying STRING protein network (evidence view) in *Mus Musculus* with 25 proteins. Additional information from other resources can be retrieved for each protein and interaction. Nodes represent proteins and different line colours denote the type of evidence for the interaction.

Conclusion

In summary, we utilized the strengths of existing network/pathway tools and databases to gain insight into processes related to cardio-myopathy have distinct molecular interaction patterns visible from various systems levels, including gene (microarray analysis), gene set, molecular pathway, and gene networks have shown that cardio-myopathy could involve three pathways; FA metabolism, angiogenesis/apoptosis and structure. Discriminating between the three pathways can help to improve the understanding of a drug's mechanism and further improve targeting in therapeutics drug research.

- [3] Sherrid MV et al. 2003 Ann Thorac Surg. 75(2): 620 [PMID 12607696]
- [4] Wigle ED et al. (1985). 28(1): 1 [PMID 3160067]
- [5] Wigle ED et al. 1995 92(7): 1680 [PMID 7671349]

- [6] Maron BJ *et al.* 2003 J Am Coll Cardiol **42**(9): 1687 [PMID 14607462]
- [7] Maron BJ *et al.* 1996 Circulation **94**(4): 850 [PMID 8772711]
- [8] Farzin Halabchi, Tohid Seif-Barghi, and Reza Mazaheri, 2011 -Sudden Cardiac Death in Young Athletes; a Literature Review and Special Considerations in Asia-Asian J Sports Med. 2011 Mar;2(1):1-15. [PMID: 22375212]
- [9] Abumrad N *et al.* 1998 J Lipid Res. **39**(12): 2309 [PMID 9831619]
- [10] Ibrahim A & Abumrad NA, 2002 Proc Nat Acad Sci USA **93**: 2646 [PMID 8610095]
- [11] Ibrahim A., Sfeir Z., maghaais H. 1993 -Expression of the CD36 homolog (FAT) in fibroblast cells: effects on fatty acid transport. Proc Nat Acad Sci USA 93:2646-2651 [PMID 8610095]
- [12] Asch AS *et al.* 1984 J. Clin. Invest **79**: 1054 [PMCID: PMC424283]
- [13] Hajri TR *et al.* (2001) J Biol Chem **276**: 23661 [PMID 11323420]
- [14] Fukuchi K *et al.* 1999 J Nucl Med **40**: 239 [PMID 10025829]
- [15] Sabaouni I *et al.* 2013 Bioinformation **9**(17): 849 [PMID 24250110]
- [16] Subramanian A *et al.* 2005 Proc Nat Acad Sci USA **102**(43): 15545 [PMID 16199517]
- [17] Subramanian A *et al.* 2007 Bioinformatics **23**(23): 3251 [PMID 17644558]
- [18] Merico D *et al.* 2010 PLoS ONE **5**(11): e13984 [PMID 21085593]
- [19] Zuberi K *et al.* 2013 Nucleic Acids Research **41**(1): W115 [PMID 23794635]
- [20] Warde-Farley D *et al.* 2010 Nucleic Acids Research **38**(supplement 2): W214 [PMID 20576703]
- [21] Szklarczyk D *et al.* 2011 Nucleic Acids Research **39**(supplement 1) D561 [PMID 21045058]
- [22] Franceschini A *et al.* 2013 Nucleic Acids Research **41**(1) D808 [PMID 23203871]
- [23] Ma'ayan A. Sci Signal. 2011 **4**:190 [PMID 21917719]
- [24] Randle PJ *et al.* 1993 **(1)**: 785 [PMID 13990765].
- [25] Kuang M *et al.* 2004 Circulation **109**: 1550 [PMID 15023869].
- [26] Asch AS *et al.* 1991 J. Biol. Chem. **266**: 1740 [PMID 1703153].
- [27] Miao WM *et al.* 2001 Blood **97**(6): 1689. [PMID: 11238109]
- [28] Khatri P & Draghici S, 2005 Bioinformatics **21**: 3587 [PMID 15994189]
- [29] Ge H *et al.* 2001 Nat. Genet. **29**: 482 [PMID 11694880]

Table 1: Gene description with corresponding reference

Gene Name	Synonyms/Identifier	Description	R
CD36	ENSG0000135218, ENSP00000308165, ENSP00000378268, ENSP00000392298, ENSP00000396258, ENSP00000398760, ENSP00000399421, ENSP00000401863, ENSP00000407690, ENSP00000409762, ENSP00000410371, ENSP00000411411, ENSP00000415743, ENSP00000416388, ENSP00000431296, ENSP00000433659, ENSP00000435698, ENSP00000439543, ENSP00000441956, 948, CD36, NP_000063, NP_001001547, NP_001001548, NP_001120915, NP_001120916, NM_000072, NM_001001547, NM_001001548, NM_001127443, NM_001127444, GP3B, GP4, GPIV, SCARB3, CD36, HUMAN, P16671,	CD36 molecule (thrombospondin receptor) (472 aa) updated on 21-Jun-2015	[a]
Dgat1	ENSG00000185000, ENSG00000261698, ENSP0000032258, ENSP00000432795, ENSP00000454624, ENSP00000457814, 8694, DGAT1, NP_036211, NM_012079, ARGP1, DGAT, DGAT1_HUMAN, O75907,	diacylglycerol O-acyltransferase 1; Catalyzes the terminal and only committed step in triacylglycerol synthesis by using diacylglycerol and fatty acyl CoA as substrates. In contrast to DGAT2 it is not essential for survival. May be involved in VLDL (very low density lipoprotein) assembly. In liver, plays a role in esterifying exogenous fatty acids to glycerol. Functions as the major acyl-CoA retinol acyltransferase (ARAT) in the skin, where it acts to maintain retinoid homeostasis and prevent retinoid toxicity leading to skin and hair disorders (488 aa)	[b]
Adh4	ENSG00000198099, ENSP00000265512, ENSP00000397939, ENSP00000423571, ENSP00000424583, ENSP00000424630, ENSP00000425416, ENSP00000426667, ENSP00000427261, ENSP00000427525, 127, ADH4, NP_000661, NM_000670, ADH-2, ADH4_HUMAN, P08319,	alcohol dehydrogenase 4 (class II), pi polypeptide (380 aa)	[c]
Irs3	ENSMUSP00000060844	insulin receptor substrate 3	[d]
Irs1	ENSG00000169047, ENSP00000304895, 3667, IRS1, NP_005535, NM_005544, HIRS-1, IRS1_HUMAN, P35568,	insulin receptor substrate 1, May mediate the control of various cellular processes by insulin. When phosphorylated by the insulin receptor binds specifically to various cellular proteins containing SH2 domains such as phosphatidylinositol 3-kinase p85 subunit or GRB2. Activates phosphatidylinositol 3-kinase when bound to the regulatory p85 subunit (By similarity)	[e]
IL2	ENSG00000109471, ENSP00000226730, 3558, IL2, NP_000577, NM_000586, IL-2, TCGF, IL2_HUMAN, P60568,	interleukin 2, Produced by T-cells in response to antigenic or mitogenic stimulation, this protein is required for T-cell proliferation and other activities crucial to regulation of the immune response. Can stimulate B-cells, monocytes, lymphokine- activated killer cells, natural killer cells, and glioma cells	[f]
Fat4	ENSG00000196159, ENSP00000335169, ENSP00000377862, 79633, FAT4, NP_078858, NM_024582, CDH14, CDHR11, FAT-J, FAT4_HUMAN, Q6V017,	FAT tumor suppressor homolog 4 (Drosophila), May function in the regulation of planar cell polarity. Cadherins are cell-cell interaction molecules (By similarity)	[g]
Pdhb	ENSG00000168291, ENSP00000307241, ENSP00000373220, ENSP00000417267, ENSP00000418448,	pyruvate dehydrogenase (lipoyamide) beta, The pyruvate dehydrogenase complex catalyzes the overall conversion of pyruvate to acetyl-CoA and CO(2), and thereby links the glycolytic pathway to the tricarboxylic cycle	[h]
Lpl	ENSG00000175445, ENSP00000309757, ENSP00000428237, ENSP00000428496, ENSP00000428557,	lipoprotein lipase, The primary function of this lipase is the hydrolysis of triglycerides of circulating chylomicrons and very low density lipoproteins (VLDL). Binding to heparin sulfate proteoglycans at the cell surface is vital to the function. The apolipoprotein, APOC2, acts as a coactivator of LPL activity in the presence of lipids on the luminal surface of vascular endothelium (By similarity)	[i]
Ide	ENSG00000175445, ENSP00000309757, ENSP00000428237, ENSP00000428496, ENSP00000428557,	lipoprotein lipase, The primary function of this lipase is the hydrolysis of triglycerides of circulating chylomicrons and very low density lipoproteins (VLDL). Binding to heparin sulfate proteoglycans at the cell surface is vital to the function. The apolipoprotein, APOC2, acts as a coactivator of LPL activity in the presence of lipids on the luminal surface of vascular endothelium (By similarity)	[j]
Uqcrh	ENSG00000173660, ENSP00000309565, 440567, 7388, UQCRH, NP_001083060, NP_00595, NM_001089591, NM_006004, QCR6, UQCR8, P07919, QCR6_HUMAN,	ubiquinol-cytochrome c reductase hinge protein, This is a component of the ubiquinol-cytochrome c reductase complex (complex III or cytochrome b-c1 complex), which is	[k]

Uqcrc1	ENSG0000010256, ENSP0000203407, ENSP0000388660, ENSP0000393696, 7384, UQCRC1, NP_003356, NM_003365, D3S3191, QCR1, UQCRC1, P31930, QCR1_HUMAN,	part of the mitochondrial respiratory chain. This protein may mediate formation of the complex between cytochromes c and c1 (91 aa) ubiquinol-cytochrome c reductase core protein I. This is a component of the ubiquinol-cytochrome c reductase complex (complex III or cytochrome b-c1 complex), which is part of the mitochondrial respiratory chain. This protein may mediate formation of the complex between cytochromes c and c1 (480 aa)	[l]
Gyk	ENSMUSP00000119564	glycerol kinase, Key enzyme in the regulation of glycerol uptake and metabolism (By similarity) (553 aa)	[m]
Thbs1	ENSG0000137801, ENSP0000260356, ENSP0000380720, 7057, THBS1, NP_003237, NM_003246, THBS, THBS-1, TSP, TSP-1, TSP1, P07996, TSP1_HUMAN,	thrombospondin 1, Adhesive glycoprotein that mediates cell-to-cell and cell-to-matrix interactions. Binds heparin. May play a role in dentinogenesis and/or maintenance of dentin and dental pulp (By similarity). Ligand for CD36 mediating antiangiogenic properties (1170 aa)	[n]
Sdc4 CD9	ENSP0000361818 ENSG0000010278, ENSP0000009180, ENSP0000371955, ENSP0000371958, ENSP00000371959, ENSP0000440985, 928, CD9, NP_001760, NM_001769, BA2, M1C3, MRP-1, TSFAN29, CD9_HUMAN, P21926,	Cell surface proteoglycan that bears heparan sulfate molecule. Involved in platelet activation and aggregation. Regulates paranodal junction formation. Involved in cell adhesion, cell motility and tumor metastasis. Required for sperm-egg fusion	[o] [p]
Pdgfra	ENSG0000134853, ENSP0000257290, ENSP0000424218, ENSP0000425232, ENSP0000425626, ENSP0000425648, ENSP0000425902, ENSP0000426472, 5156, PDGFRA, NP_006197, NM_006206, CD140a, PDGFR2, P16234, PGFRA_HUMAN,	platelet-derived growth factor receptor, alpha polypeptide, Tyrosine-protein kinase that acts as a cell-surface receptor for PDGFA, PDGFB and PDGFC and plays an essential role in the regulation of embryonic development, cell proliferation, survival and chemotaxis. Depending on the context, promotes or inhibits cell proliferation and cell migration. Plays an important role in the differentiation of bone marrow-derived mesenchymal stem cells. Required for normal skeleton development and cephalic closure during embryonic development. Required for normal development of the mucosa lining th [...] (1089 aa)	[q]
Hand1	ENSG0000113196, ENSP0000231121, 9421, HAND1, NP_004812, NM_004821, bHLHa27, eHand, Hxt, Thing1, HAND1_HUMAN, O96004,	heart and neural crest derivatives expressed 1, Transcription factor that plays an essential role in both trophoblast-giant cells differentiation and in cardiac morphogenesis. In the adult, could be required for ongoing expression of cardiac-specific genes. Binds the DNA sequence 5'- NRTCTG-3' (non-canonical E-box) (By similarity) (215 aa)	[r]
Arnt2	ENSG0000172379, ENSP0000307479, ENSP0000452961, ENSP0000453651, ENSP0000453792, 9915, ARNT2, NP_055677, NM_014862, bHLHe1, KIAA0307, ARNT2_HUMAN, Q9HBZ2,	aryl-hydrocarbon receptor nuclear translocator 2, Specifically recognizes the xenobiotic response element (XRE)	[s]
Rag1	ENSG0000166349, ENSP0000299440, ENSP0000434610, 5896, RAG1, NP_000439, NM_000448, MGC43321, RNF74, P15918, RAG1_HUMAN,	recombination activating gene 1, Catalytic component of the RAG complex, a multiprotein complex that mediates the DNA cleavage phase during V(D)J recombination. V(D)J recombination assembles a diverse repertoire of immunoglobulin and T-cell receptor genes in developing B and T-lymphocytes through rearrangement of different V (variable), in some cases D (diversity), and J (joining) gene segments. In the RAG complex, RAG1 mediates the DNA-binding to the conserved recombination signal sequences (RSS) and catalyzes the DNA cleavage activities by introducing a double-strand break between t [...]	[t]
Api5	ENSG0000166181, ENSP0000368129, ENSP0000399341, ENSP0000402540, ENSP0000431391, ENSP0000434462, ENSP0000436189, ENSP0000436436, 8539, API5, NP_001136402, NP_001136403, NP_006586, NM_001142930, NM_001142931, NM_006595, AAC-11, AAC11, API5L1, API5_HUMAN, Q9BZZ5,	apoptosis inhibitor 5, Antiapoptotic factor that may have a role in protein assembly. Negatively regulates ACIN1. By binding to ACIN1, it suppresses ACIN1 cleavage from CASP3 and ACIN1-mediated DNA fragmentation. Also known to efficiently suppress E2F1-induced apoptosis. Its depletion enhances the cytotoxic action of the chemotherapeutic drugs	[u]
Map3K2	ENSG0000166181, ENSP0000368129, ENSP0000399341, ENSP0000402540, ENSP0000431391, ENSP0000434462, ENSP0000436189, ENSP0000436436, 8539, API5, NP_001136402, NP_001136403, NP_006586, NM_001142930, NM_001142931, NM_006595, AAC-11, AAC11, API5L1, API5_HUMAN, Q9BZZ5,	apoptosis inhibitor 5, Antiapoptotic factor that may have a role in protein assembly. Negatively regulates ACIN1. By binding to ACIN1, it suppresses ACIN1 cleavage from CASP3 and ACIN1-mediated DNA fragmentation. Also known to efficiently suppress E2F1-induced apoptosis. Its depletion enhances the cytotoxic action of the chemotherapeutic drugs	[v]
Merk	ENSG0000153208, ENSP0000295408, ENSP0000387277, ENSP0000389152, ENSP0000402129, ENSP0000412660, 10461, MERTK, NP_006334, NM_006343, mer, RP38, MERTK_HUMAN, Q12866,	c-mer proto-oncogene tyrosine kinase, Receptor tyrosine kinase that transduces signals from the extracellular matrix into the cytoplasm by binding to several ligands including LGALS3, TUB, TULP1 or GAS6. Regulates many physiological processes including cell survival, migration, differentiation, and phagocytosis of apoptotic cells (efferocytosis). Ligand binding at the cell surface induces autophasphorylation of MERTK on its intracellular domain that provides docking sites for downstream signaling molecules. Following activation by ligand, interacts with GRB2 or PLCG2 and induces phosph [...] (999 aa)	[w]
Myl4	ENSG0000198336, ENSP0000347055, ENSP0000377096, ENSP0000442375, ENSP0000458194, ENSP0000458907, ENSP0000459035, ENSP0000460734, ENSP0000461121, ENSP0000461570, ENSP0000461747, 4635, MYL4, NP_001002841, NP_002467, NM_001002841, NM_002467, AMLC, GT1, PRO1957, MYL4_HUMAN, P12829,	myosin, light chain 4, alkali, atrial, embryonic, Regulatory light chain of myosin. Does not bind calcium (197 aa)	[x]
Tnnt2	ENSG0000118194, ENSP0000236918, ENSP0000353535, ENSP0000356284, ENSP0000356286, ENSP0000356287, ENSP0000356289, ENSP0000356291, ENSP0000387874, ENSP0000395163, ENSP0000402238, ENSP0000404134, ENSP0000408731, ENSP0000414036, ENSP0000422031, 7139, TNNT2, NP_000355, NP_001001430, NP_001001431, NP_001001432, NP_001263274, NP_001263275, NP_001263276, NM_00364, NM_001001430, NM_001001431, NM_001001432, NM_001276345, NM_001276346, NM_001276347, CMH2, P45379, TNNT2_HUMAN, ENSP0000284770	troponin T type 2 (cardiac), Troponin T is the tropomyosin-binding subunit of troponin, the thin filament regulatory complex which confers calcium-sensitivity to striated muscle actomyosin ATPase activity (288 aa)	[y]
Pdlim3	ENSP00000284770	PDZ and LIM domain 3,May play a role in the organization of actin filament arrays within muscle	[z]
Ifng	ENSG0000111537, ENSP0000229135, 3458, IFNG, NP_000610, NM_000619, IFNG_HUMAN, P01579,	respiratory chain. This protein may mediate formation of the complex between cytochromes c and c1 (480 aa)	[aa]
Pdlim3	ENSP00000284770	interferon, gamma, Produced by lymphocytes activated by specific antigens or mitogens. IFN-gamma, in addition to having antiviral activity, has important immunoregulatory functions. It is a potent activator of macrophages, it has antiproliferative effects on transformed cells and it can potentiate the antiviral and antitumor effects of the type I interferons (By similarity)	[bb]
		PDZ and LIM domain 3,May play a role in the organization of actin filament arrays within muscle cells (By similarity) (364 aa)	

R = Reference

Table 3: Reference list to Table 1

Reference [R]	URL
[a]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=948

ISSN 0973-2063 (online) 0973-8894 (print)

[b]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=8694
[c]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=127
[d]	http://www.uniprot.org/uniprot/O5
[e]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=3667
[f]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=3558
[g]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=79633
[h]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=5162
[i]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=4023
[j]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=4023
[k]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=440567
[l]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=7384
[m]	http://string-db.org/newstring_cgi/show_network_section.pl
[n]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=7057
[o]	http://www.uniprot.org/uniprot/B4E1S6
[p]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=928
[q]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=5156
[r]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=9421
[s]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=9915
[t]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=5896
[u]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=8539
[v]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=8539
[w]	http://www.uniprot.org/uniprot/B2RE75
[x]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=4635
[y]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=7139
[z]	http://www.uniprot.org/uniprot/D6RAF1
[aa]	http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene&cmd=search&term=3458
[bb]	http://www.uniprot.org/uniprot/D6RAF1

Table 2: Interactions for CD36 (early response) network using GeneMania and STRING

Interactions	GENMANIA	STRING
CD36 -> Dgat1	Co-localization	Co-Mentioned in PubMed Abstracts
CD36 -> Lpl	Co-expression	Co-Expression and Co-Mentioned in PubMed Abstracts
CD36 -> Uqcrc2	Co-expression	-
CD36 -> Scab1	Predicted shared protein domain	-
CD36 -> Scap2	Predicted, shared protein domain co-expression	-
CD36 -> Mertk	Predicted	Experimental/Biochemical Data : putative homologs were found interacting in other species ; Co-Mentioned in PubMed Abstracts
CD36 -> CD9	Predicted	Experimental/Biochemical Data : putative homologs were found interacting in other species Co-Mentioned in PubMed Abstracts Association in Curated Databases
CD36 -> Hand1	Predicted	-
CD36 -> Rag1	Predicted	-
CD36 -> Map3K2	Predicted	-
CD36 -> Il2	Predicted	-
CD36 -> Ifg	Predicted	-
CD36 -> Thbs1	Predicted	Experimental/Biochemical Data : putative homologs were found interacting in other species Co-Mentioned in PubMed Abstracts Association in Curated Databases
CD36 -> Sdc4	Predicted	Experimental/Biochemical Data : putative homologs were found interacting in other species Co-Mentioned in PubMed Abstracts Association in Curated Databases
CD36 -> Irs1	Predicted	-
CD36 -> Pdagfra	Predicted and Co-expression	-
CD36 -> TnnT3	Co-expression	-
CD36 -> TnnT1	Co-expression	-
CD36 -> Sdc1	Predicted	Experimental/Biochemical Data : putative homologs were found interacting in other species Co-Mentioned in PubMed Abstracts Association in Curated Databases
CD36 -> Sdc3	Predicted	Experimental/Biochemical Data : putative homologs were found interacting in other species Co-Mentioned in PubMed Abstracts Association in Curated Databases
TNNT2 -> MYL4	Co-expression	-
UQCRC1 ->UQCRRH	Co-expression	-
CD9->UQCRRH	Co-expression	-
PDHB->UQCRRH	Co-expression	-
UQCRC1->PDHB	Co-expression	-
HAND1->TNNT2	Co-expression	-
PDGFRA->THBS1	Co-expression	-
LPL->THBS1	Co-expression	-
IR51->PDGFRA	Co-expression	-
LPL->MYL4	Co-expression	-
SDC4->MERTK	Co-expression	-
SDC4->PDGFRA	Co-expression	-
SDC4->IFNG	Co-expression	-
HAND1->MYL4	Co-expression	-
HAND1->PDGFRA	Co-expression	-
HAND1->THBS1	Co-expression	-
HAND1->IRS1	Co-expression	-
HAND1->LPL	Genetic interactions	-
UQCRC1 ->DGAT1	Genetic interactions	-
HAND1->ADH4	Genetic interactions	-
HAND1 ->IL2	Genetic interactions	-
LPL ->FAT4	Genetic interactions	-
PDGFRA ->FAT4	Genetic interactions	-
LPL ->TNNT2	Genetic interactions	-
MAP3K2 ->IDE	Genetic interactions	-
LPL->MYL4	Genetic interactions	-
TNNT2->MYL4	Genetic interactions	-
UQCRC1 ->RAG1	Genetic interactions	-
IFNG->THBS1	Genetic interactions	-
SDC4 ->API5	Genetic interactions	-
UQCRC1 ->API5	Genetic interactions	-

LPL->API5	Genetic interactions	-
MERTK->RAG1	Genetic interactions	-
IFNG->UQCRC1	Genetic interactions	-
MYL4->FAT4	Genetic interactions	-
UQCRRH->API5	Genetic interactions	-
IFNG->UQCRRH	Genetic interactions	-
IRS1->API5	Genetic interactions	-
PDGFRA->MAP3K2	Genetic interactions	-
MAP3K2->FAT4	Genetic interactions	-
MERTK->PDLIM3	Genetic interactions	-
CD9->MERTK	Genetic interactions	-
PDGFRA ->IDE	Genetic interactions	-
FAT4 ->API5	Genetic interactions	-
IRS1->FAT4	Genetic interactions	-
PDGFRA->PDHB	Genetic interactions	-
PDHB ->FAT4	Genetic interactions	-
SDC4->THBS1	Pathway	-
TNNT2->MYL4	Pathway	-
IFNG->PDGFRA	Pathway	-
IFNG ->IL2	Pathway	-
IRS1->IL2	Pathway	-
UQCRC1->IDE	Shared protein domains	-
PDGFRA->MERTK	Shared protein domains	-
IFNG->IL2	Shared protein domains	-
THBS1->FAT4	Shared protein domains	-

Edited by P Kanguane

Citation: Sabaouni *et al.* Bioinformation 12(6): 332-339 (2016)

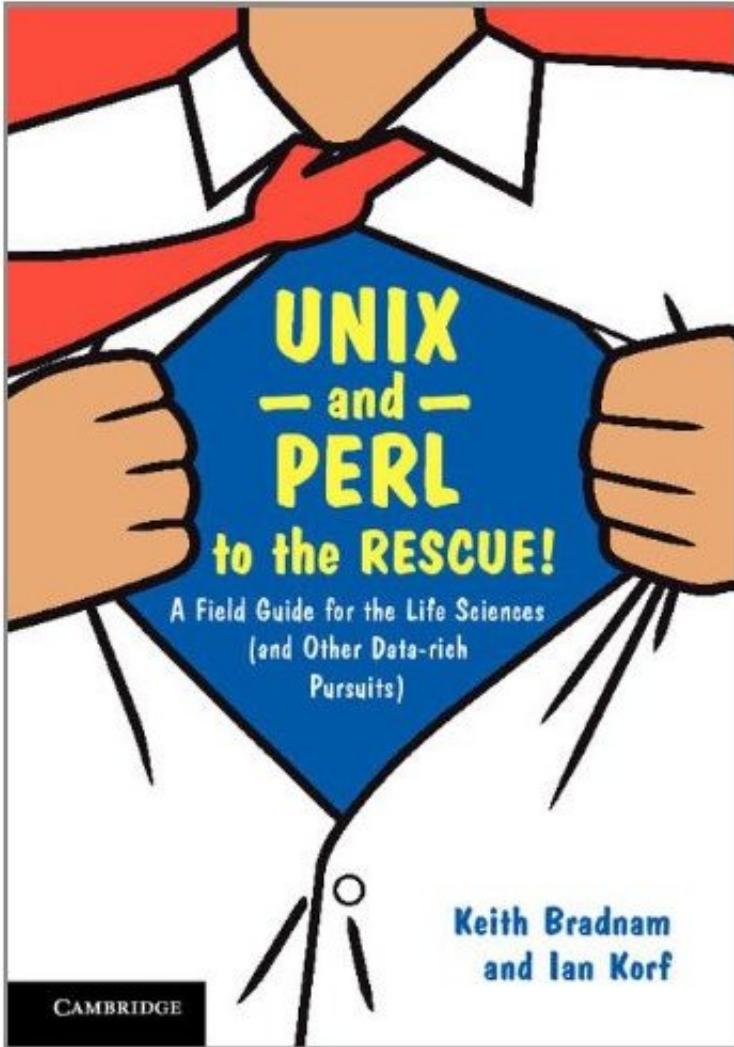
License statement: This is an Open Access article which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly credited. This is distributed under the terms of the Creative Commons Attribution License.

APPENDICES

Shameless Plug

This course has been greatly extended and reworked into a book that has been published by Cambridge University Press. It is available to order on [Amazon.com](#) and at many other [online stores](#). It is also available in various [ebook formats](#).

Unix and Perl to the Rescue! A field guide for the life sciences (and other data-rich pursuits)



Unix and Perl to the Rescue!

This primer will remain freely available, though we of course hope that if you find the primer useful, you will consider taking a look at our book. In the book we greatly expand on every subject that is in the primer, as well as covering many more topics. Some of these extra topics include more coverage of Unix and Perl, but we also devote sections to areas such as ‘Data Management’, ‘Revision Control’, and ‘Code Beautification’. There are also many more jokes and geeky cultural references.

We have also created a website at <http://unixandperl.com/> to support both the primer and the book, and should there ever be a movie adaptation of the book (starring Tom Cruise as ‘grep’?) I expect that you’ll be able to find out about that on the website as well.

Enjoy!

Keith Bradnam & Ian Korf May 2012

Introduction

Advances in high-throughput biology have transformed modern biology into an incredibly data-rich science. Biologists who never thought they needed computer programming skills are now finding that using an Excel spreadsheet is simply not enough. Learning to program a computer can be a daunting task, but it is also incredibly worthwhile. You will not only improve your research, you will also open your mind to new ways of thinking and have a lot of fun.

This course is designed for Biologists who want to learn how to program but never got around to it. Programming, like language or math, comes more naturally to some than others. But we all learn to read, write, add, subtract, etc., and we can all learn to program. Programming, more than just about any other skill, comes in waves of understanding. You will get stuck for a while and a little frustrated, but then suddenly you will see how a new concept aggregates a lot of seemingly disconnected information. And then you will embrace the new way, and never imagine going back to the old way.

As you are learning, if you are getting confused and discouraged, slow down and ask questions. You can contact us either in person, by email, or (preferably) on the associated [Unix and Perl for Biologists Google Group](#). The lessons build on each other, so do not skip ahead thinking you will return to the confusing concept at a later date.

Why Unix?

The [Unix operating system](#) has been around since 1969. Back then there was no such thing as a graphical user interface. You typed everything. It may seem archaic to use a keyboard to issue commands today, but it's much easier to automate keyboard tasks than mouse tasks. There are several variants of Unix (including [Linux](#)), though the differences do not matter much. Though you may not have noticed it, Apple has been using Unix as the underlying operating system on all of their computers since 2001.

Increasingly, the raw output of biological research exists as *in silico* data, usually in the form of large text files. Unix is particularly suited to working with such files and has several powerful (and flexible) commands that can process your data for you. The real strength of learning Unix is that most of these commands can be combined in an almost unlimited fashion. So if you can learn just five Unix commands, you will be able to do a lot more than just five things.

Why Perl?

Perl is one of the most popular Unix programming languages. It doesn't matter much which language you learn first because once you know how one works, it is much easier to learn others. Among languages, there is often a distinction between interpreted (e.g. Perl, Python, Ruby) and compiled (e.g. C, C++, Java) languages. People often call interpreted programs scripts. It is generally easier to learn programming in a scripting language because you don't have to worry as much about variable types and memory allocation. The downside is the interpreted programs often run much slower than compiled ones (100-fold is common). But let's not get lost in petty details. Scripts are programs, scripting is programming, and computers can solve problems quickly regardless of the language.

Typeset Conventions

All of the Unix and Perl code in these guides is written in constant-width font with line numbering. Here is an example with 3 lines:

```
1. for ($i = 0; $i < 10; $i++) {  
2.     print $i, "\n";  
3. }
```

Text you are meant to type into a terminal is indented in constant-width font without line numbering. Here is an example:

```
ls -lrh
```

Sometimes a paragraph will include a reference to a Unix command, Perl function, or a file that you should be working with, Any such text will be in a constant-width, boxed font. E.g.

Type the `pwd` command again.

From time to time this documentation will contain [web links](#) to pages that will help you find out more about certain Unix commands and Perl functions. Usually, the *first* mention of a command or function will be a hyperlink to Wikipedia (for Unix commands) or to <http://perldoc.perl.org> (for Perl functions). Important or critical points will be styled like so:

This is an important point!

About the authors

Keith Bradnam started out his academic career studying ecology. This involved lots of field trips and throwing [quadrats](#) around on windy hillsides. He was then lucky to be in the right place at the right time to do a Masters degree in Bioinformatics (at a time when nobody was very sure what bioinformatics was). From that point onwards he has spent most of his waking life sat a keyboard (often staring into a Unix terminal). A PhD studying eukaryotic genome evolution followed; this was made easier by the fact that only one genome had been completed at the time he started (this soon changed). After a brief stint working on an Arabidopsis genome database, he moved to working on the excellent model organism database [WormBase](#) at the Wellcome Trust Sanger Institute. It was here that he first met Ian Korf and they bonded over a shared love of Macs, neatly written code, and English puddings. Ian then tried to run away and hide in California at the UC Davis [Genome Center](#) but Keith tracked him down and joined his lab. Apart from doing research, he also gets to look after all the computers in the lab and teach the occasional class or two. However, he would give it all up for the chance to be able to consistently beat Ian at foosball, but that seems unlikely to happen anytime soon. Keith still likes Macs and neatly written code, but now has a much harder job finding English puddings.

Ian Korf believes that you can tell what a person will do with their life by examining their passions as a teen. Although he had no idea what a ‘sequence analysis algorithm’ was at 16, a deep curiosity about biological mechanisms and an obsession with writing/playing computer games is only a few bits away. Ian’s first experience with bioinformatics came as a post-doc at Washington University (St. Louis) where he was a member of the Human Genome Project. He then went across the pond to the Sanger Centre for another post-doc. There he met Keith Bradnam, and found someone who truly understood the role of communication and presentation in science. Ian was somehow able to persuade Keith to join his new lab in Davis California, and this primer on Unix and Perl is but one of their hopefully useful contributions.

Preamble

What computers can run Perl?

One of the main goals of this course is to learn Perl. As a programming language, Perl is platform agnostic. You can write (and run) Perl scripts on just about any computer. We will assume that >99% of the people who are reading this use either a Microsoft Windows PC, an Apple Mac, or one of the many Linux distributions that are available (Linux can be considered as a type of Unix, though this claim might offend the Linux purists reading this). A small proportion of you may be using some other type of dedicated Unix platform, such as Sun or SGI. For the Perl examples, none of this matters. All of the Perl scripts in this course should work on any machine that you can install Perl on (if an example doesn't work then please let us know!).

What computers can run Unix?

Unlike our Perl documentation, the Unix part of this course is not quite so portable to other types of computer. We decided that this course should include an introduction to Unix because most bioinformatics happens on Unix/Linux platforms; so it makes sense to learn how to run your Perl scripts in the context of a Unix operating system. If you read the Introduction, then you will know that all modern Mac computers are in fact Unix machines. This makes teaching Perl & Unix on a Mac a relatively straightforward proposition, though we are aware that this does not help those of you who use Windows. This is something that we will try to specifically address in later updates to this course. For now, we would like to point out that you can achieve a Unix-like environment on your Windows PC in one of two ways:

1. Install [Cygwin](#) — this provides a Linux-like environment on your PC, it is also free to download. There are some differences between Cygwin and other types of Unix which may mean that not every Unix example in this course works exactly as described, but overall it should be sufficient for you to learn the basics of Unix.
2. Install Linux by using [virtualization](#) software — there are many pieces of software that will now allow you effectively install one operating system within another operating system. Microsoft has its own (free) [Virtual PC](#) software, and here are some [instructions for installing Linux](#) using Virtual PC.

You should also be aware that there is a lot of variation within the world of Unix/Linux. Most commands will be the same, but the layout of the file system may look a little different. Hopefully our documentation should work for most types of Unix, but bear in mind it was written (and tested) with Apple's version of Unix.

Do I need to run this course from a USB drive?

We originally developed this course to be taught in a computer classroom environment. Because of this we decided to put the entire course (documentation & data) on to a USB flash drive. One reason for doing this was so that people could take the flash drive home with them and continue working on their own computers.

If you have your own computer which is capable of running a Unix/Linux environment then you might prefer to use that, rather than using a flash drive. If you have downloaded the course material, then after unpacking it you should have a directory called 'Unix_and_Perl_course'. You can either copy this directory (about 100 MB in size at the time of writing) to a flash drive or to any other directory within your Unix environment. Instructions in this document will assume that you are working on a flash drive on a Mac computer, so many of the Unix examples will not work exactly as written on other systems. In most cases you will just need to change the name of any directories they are used in the examples.

In our examples, we assume that the course material is located on a flash drive that is named 'USB'. If you run the course from your own flash-drive, you might find it easier to rename it to 'USB' as well, though you don't have to do this.

Part 1: Unix - Learning the essentials

Introduction to Unix

These exercises will (hopefully) teach you to become comfortable when working in the environment of the Unix terminal. Unix contains many hundred of commands but you will probably use just 10 or so to achieve most of what you want to do.

You are probably used to working with programs like the Apple Finder or the Windows File Explorer to navigate around the hard drive of your computer. Some people are so used to using the mouse to move files, drag files to trash etc. that it can seem strange switching from this behavior to typing commands instead. Be patient, and try — as much as possible — to stay within world of the Unix terminal. Please make sure you complete and understand each task before moving on to the next one.

First steps

The lessons from this point onwards will assume the following:

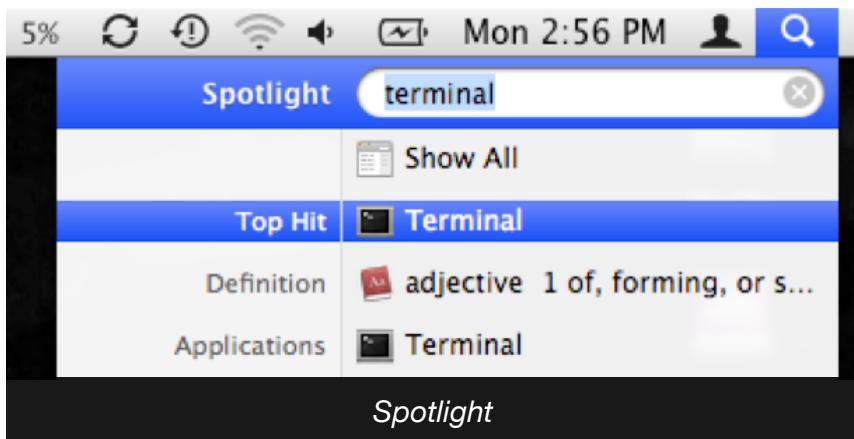
1. You have downloaded the [Unix and Perl course material](#) and copied it to a USB flash drive .
2. The flash drive has been renamed to ‘USB’.
3. You have removed the downloaded files from your Desktop/Downloads folder (this is often the source of confusion when you have one copy on your USB drive and a separate copy on your Desktop) .

U1. The Terminal

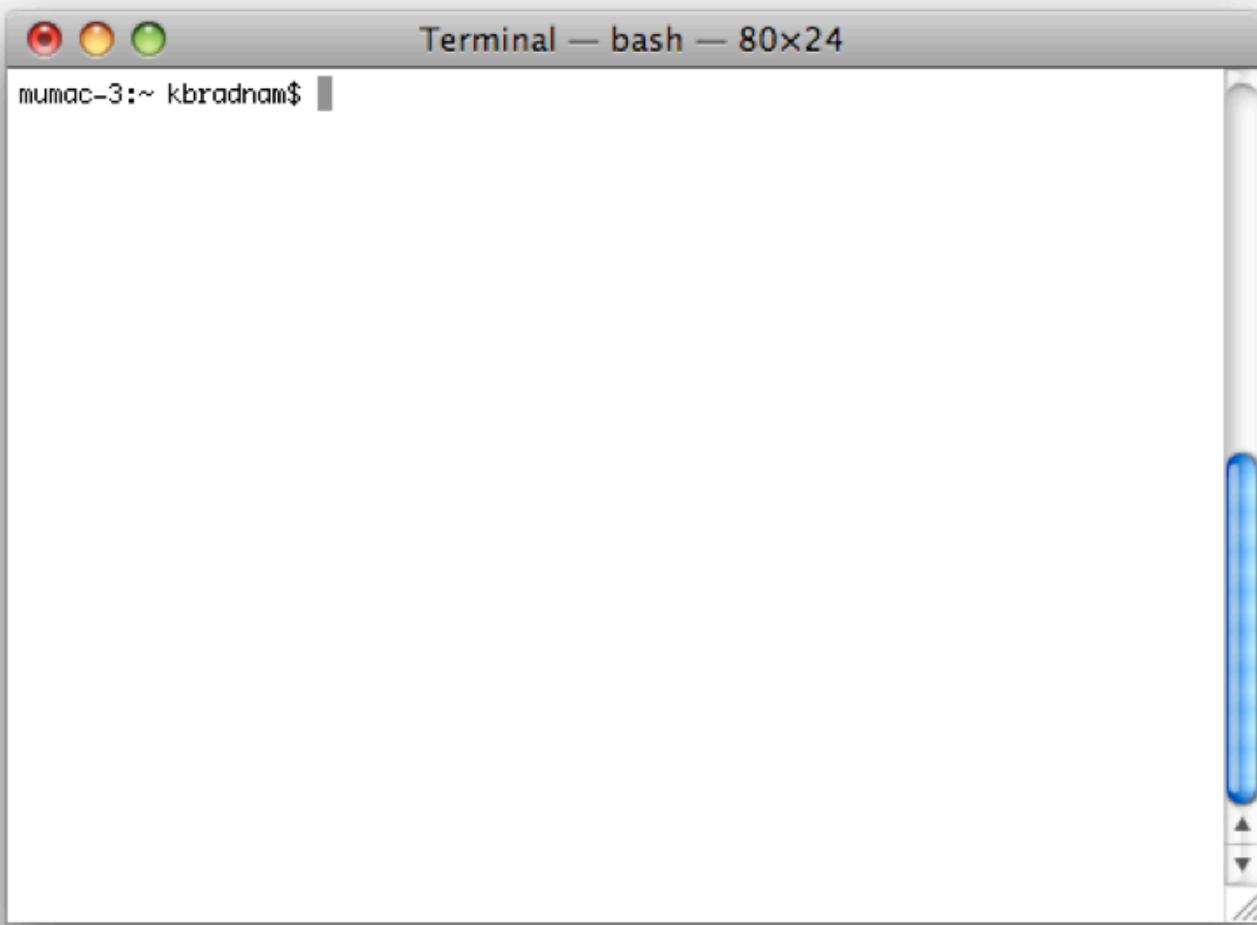
A ‘terminal’ is the common name for the program that does two main things. It allows you to type input to the computer (i.e. run programs, move/view files etc.) and it allows you to see output from those programs. All Unix machines will have a terminal program and on Apple computers, the terminal application is unsurprisingly named ‘Terminal’.

Task U1.1

Use the ‘Spotlight’ search tool (the little magnifying glass in the top right of the menu bar) to find, and then launch, Apple’s Terminal application:



You should now see something that looks like the following (any text that appears inside your terminal window will look different):



(http://korflab.ucdavis.edu/Unix_and_Perl/terminal.png)

Before we go any further, you should note that you can:

- make the text larger/smaller (hold down ‘command’ and either ‘+’ or ‘-’)
- resize the window (this will often be necessary)
- have multiple terminal windows on screen (see the ‘Shell’ menu)
- have multiple tabs open within each window (again see the ‘Shell’ menu)

There will be many situations where it will be useful to have multiple terminals open and it will be a matter of preference as to whether you want to have multiple windows, or one window with multiple tabs (there are keyboard shortcuts for switching between windows, or moving between tabs).

U2. Your first Unix command

Unix keeps files arranged in a hierarchical structure. From the ‘top-level’ of the computer, there will be a number of directories, each of which can contain files and subdirectories, and each of those in turn can of course contain more files and directories and so on, ad infinitum. It’s important to note that you will always be “in” a directory when using the terminal. The default behavior is that when you open a new terminal you start in your own ‘home’ directory (containing files and directories that only you can modify).

To see what files are in our home directory, we need to use the `ls` command. This command ‘lists’ the contents of a directory. So why don’t they call the command ‘list’ instead? Well, this is a good thing because typing long commands over and over again is tiring and time-consuming. There are many (frequently used) Unix commands that are just two or three letters. If we run the `ls` command we should see something like:

```
olson27-1:~ kbradnam$ ls
Application Shortcuts Documents Library
Desktop Downloads
olson27-1:~ kbradnam$
```

There are four things that you should note here:

1. You will probably see different output to what is shown here, it depends on your computer. Don’t worry about that for now.
2. The `olson27-1:~ kbradnam$` text that you see is the Unix **command prompt**. It contains a user name (`kbradnam`), the name of the machine that this user is working on (`olson27-1`) and the name of the current directory (`~` more on that later). Note that the command prompt might not look the same on different Unix systems. In this case, the `$` sign marks the end of the prompt.
3. The output of the `ls` command lists five things. In this case, they are all directories, but they could also be files. We’ll learn how to tell them apart later on.
4. After the `ls` command finishes it produces a new command prompt, ready for you to type your next command.

The `ls` command is used to list the contents of *any* directory, not necessarily the one that you are currently in. Plug in your USB drive, and type the following:

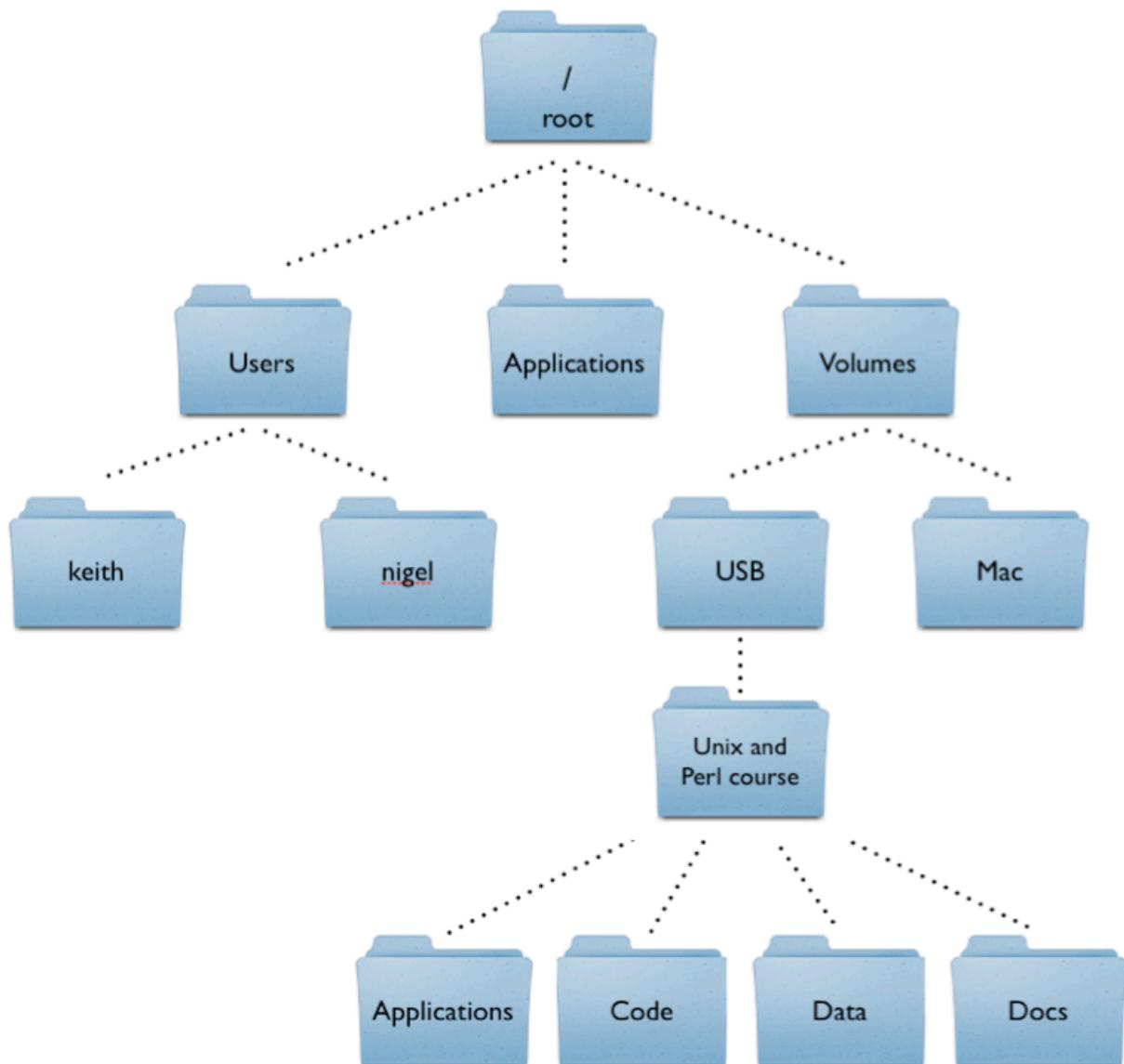
```
olson27-1:~ kbradnam$ ls /Volumes/USB/Unix_and_Perl_course
Applications      Code      Data      Documentation
```

On a Mac, plugged in drives appear as subdirectories in the special ‘Volumes’ directory. The name of the USB flash drive is ‘USB’. The above output shows a set of four directories that are all “inside” the ‘Unix_and_Perl_course’ directory). Note how the underscore character ‘_’ is used to space out words in the directory name.

U3: The Unix tree

Looking at directories from within a Unix terminal can often seem confusing. But bear in mind that these directories are exactly the same type of folders that you can see if you use Apple's graphical file-management program (known as 'The Finder'). A tree analogy is often used when describing computer filesystems. From the root level (/) there can be one or more top level directories, though most Macs will have about a dozen. In the example below, we show just three. When you log in to a computer you are working with your files in your home directory, and this will nearly always be inside a 'Users' directory. On many computers there will be multiple users.

All Macs have an applications directory where all the GUI (graphical user interface) programs are kept (e.g. iTunes, Microsoft Word, Terminal). Another directory that will be on all Macs is the Volumes directory. In addition to any attached *external* drives, the Volumes directory should also contain directories for every *internal* hard drive (of which there should be at least one, in this case it's simply called 'Mac'). It will help to think of this tree when we come to copying and moving files. E.g. if we had a file in the 'Code' directory and wanted to copy it to the 'keith' directory, we would have to go *up* four levels to the root level, and then *down* two levels.



Example directory structure

U4: Finding out where you are

There may be many hundreds of directories on any Unix machine, so how do you know which one you are in? The command `pwd` will Print the [Working Directory](#) and that's pretty much all this command does:

```
olson27-1:~ kbradnam$ pwd  
/users/clmuser
```

When you log in to a Unix computer, you are typically placed into your *home* directory. In this example, after we log in, we are placed in a directory called ‘clmuser’ which itself is a subdirectory of another directory called ‘users’. Conversely, ‘users’ is the parent directory of ‘clmuser’. The first forward slash that appears in a list of directory names always refers to the top level directory of the file system (known as the [root directory](#)). The remaining forward slash (between ‘users’ and ‘clmuser’) delimits the various parts of the directory hierarchy. If you ever get ‘lost’ in Unix, remember the `pwd` command.

As you learn Unix you will frequently type commands that don’t seem to work. Most of the time this will be because you are in the wrong directory, so it’s a really good habit to get used to running the `pwd` command a lot.

U5: Getting from ‘A’ to ‘B’

We are in the home directory on the computer but we want to work on the USB drive. To change directories in Unix, we use the `cd` command:

```
olson27-1:~ kbradnam$ cd /Volumes/USB/Unix_and_Perl_course
olson27-1:USB kbradnam$ ls
Applications      Code          Data          Documentation
olson27-1:USB kbradnam$ pwd
/Volumes/USB/Unix_and_Perl_course
```

The first command reads as “change directory to the `Unix_and_Perl_course` directory that is inside a directory called ‘`USB`’, which itself is inside the `Volumes` directory that is at the root level of the computer”. Did you notice that the command prompt changed after you ran the `cd` command? The ‘`~`’ sign should have changed to ‘`Unix_and_Perl_course`’. This is a useful feature of the command prompt. By default it reminds you where you are as you move through different directories on the computer.

NB. For the sake of clarity, we will now simplify the command prompt in all of the following examples

U6: Root is the root of all evil

In the previous example, we could have achieved the same result in three separate steps:

```
$ cd /Volumes  
$ cd USB  
$ cd Unix_and_Perl_course
```

Note that the second and third commands do not include a forward slash. When you specify a directory that starts with a forward slash, you are referring to a directory that should exist one level below the root level of the computer. What happens if you try the following two commands? The first command should produce an error message.

```
$ cd Volumes  
$ cd /Volumes
```

The error is because without including a leading slash, Unix is trying to change to a ‘Volumes’ directory below your current level in the file hierarchy (/Volumes/USB/Unix_and_Perl_course), and there is no directory called Volumes at this location.

U7: Up, up, and away

Frequently, you will find that you want to go ‘upwards’ one level in the directory hierarchy. Two dots .. are used in Unix to refer to the *parent* directory of wherever you are. Every directory has a parent except the root level of the computer:

```
$ cd /Volumes/USB/Unix_and_Perl_course
$ pwd
/Volumes/USB/Unix_and_Perl_course
$ cd ..
$ pwd
/Volumes/USB
```

What if you wanted to navigate up *two* levels in the file system in one go? It’s very simple, just use two sets of the .. operator, separated by a forward slash:

```
$ cd /Volumes/USB/Unix_and_Perl_course
$ pwd
/Volumes/USB/Unix_and_Perl_course
$ cd ../../..
$ pwd
/Volumes
```

U8: I'm absolutely sure that this is all relative

Using `cd ..` allows us to change directory *relative* to where we are now. You can also always change to a directory based on its *absolute* location. E.g. if you are working in the `/Volumes/USB/Unix_and_Perl_course/Code` directory and you then want to change to the `/Volumes/USB/Unix_and_Perl_course/Data` directory, then you could do either of the following:

```
$ cd ../Data
```

or...

```
$ cd /Volumes/USB/Unix_and_Perl_course/Data
```

They both achieve the same thing, but the 2nd example requires that you know about the full *path* from the root level of the computer to your directory of interest (the ‘path’ is an important concept in Unix). Sometimes it is quicker to change directories using the relative path, and other times it will be quicker to use the absolute path.

U9: Time to go home

Remember that the command prompt shows you the name of the directory that you are currently in, and that when you are in your home directory it shows you a tilde character (~) instead? This is because Unix uses the tilde character as a short-hand way of [specifying a home directory](#).

Task U9.1

See what happens when you try the following commands (use the `pwd` command after each one to confirm the results):

```
$ cd /
$ cd ~
$ cd /
$ cd
```

Hopefully, you should find that `cd` and `cd ~` do the same thing, i.e. they take you back to your home directory (from wherever you were). Also notice how you can specify the single forward slash to refer to the root directory of the computer. When working with Unix you will frequently want to jump straight back to your home directory, and typing `cd` is a very quick way to get there.

U10: Making the ls command more useful

The .. operator that we saw earlier can also be used with the ls command. Can you see how the following command is listing the contents of the root directory? If you want to test this, try running ls / and see if the output is any different.

```
$ cd /Volumes/USB/Unix_and_Perl_course
$ ls ../../..
Applications      Volumes      net
CRC              bin          oldlogins
Developer        cores        private
Library          dev          sbin
Network          etc          tmp
Server           home         usr
System            mach_kernel var
Users             mach_kernel.ctfsys
```

The ls command (like most Unix commands) has a set of options that can be added to the command to change the results. Command-line options in Unix are specified by using a dash ('-') after the command name followed by various letters, numbers, or words. If you add the letter 'l' to the ls command it will give you a 'longer' output compared to the default:

```
$ ls -l /Volumes/USB/Unix_and_Perl_course
total 192
drwxrwxrwx  1 keith  staff  16384 Oct  3 09:03 Applications
drwxrwxrwx  1 keith  staff  16384 Oct  3 11:11 Code
drwxrwxrwx  1 keith  staff  16384 Oct  3 11:12 Data
drwxrwxrwx  1 keith  staff  16384 Oct  3 11:34 Documentation
```

For each file or directory we now see more information (including file ownership and modification times). The 'd' at the start of each line indicates that these are directories

Task U10.1

There are many, many different options for the ls command. Try out the following (against any directory of your choice) to see how the output changes.

```
ls -l  
ls -R  
ls -l -t -r  
ls -lh
```

Note that the last example combine multiple options but only use one dash. This is a very common way of specifying multiple command-line options. You may be wondering what some of these options are doing. It's time to learn about Unix documentation....

U11: Man your battle stations!

If every Unix command has so many options, you might be wondering how you find out what they are and what they do. Well, thankfully every Unix command has an associated ‘manual’ that you can access by using the `man` command. E.g.

```
$ man ls  
$ man cd  
$ man man # yes even the man command has a manual page
```

When you are using the `man` command, press space to scroll down a page, `b` to go back a page, or `q` to quit. You can also use the up and down arrows to scroll a line at a time. The `man` command is actually using another Unix program, a text viewer called `less`, which we’ll come to later on.

Some Unix commands have very long manual pages, which might seem very confusing. It is typical though to always list the command line options early on in the documentation, so you shouldn’t have to read too much in order to find out what a command-line option is doing.

U12: Make directories, not war

If we want to make a new directory (e.g. to store some work related data), we can use the `mkdir` command:

```
$ cd /Volumes/USB/Unix_and_Perl_course
$ mkdir Work
$ ls
Applications      Code          Data          Documentation    Work
$ mkdir Temp1
$ cd Temp1
$ mkdir Temp2
$ cd Temp2
$ pwd
/Volumes/USB/Unix_and_Perl_course/Temp1/Temp2
```

In the last example we created the two temp directories in two separate steps. If we had used the `-p` option of the `mkdir` command we could have done this in one step. E.g.

```
$ mkdir -p Temp1/Temp2
```

Task U12.1

Practice creating some directories and navigating between them using the `cd` command. Try changing directories using both the *absolute* as well as the *relative* path (see section [U8](#)).

U13: Time to tidy up

We now have a few (empty) directories that we should remove. To do this use the `rmdir` command, this will only remove empty directories so it is quite safe to use. If you want to know more about this command (or any Unix command), then remember that you can just look at its man page.

```
$ cd /Volumes/USB/Unix_and_Perl_course  
$ rmdir Work
```

Task U13.1

Remove the remaining empty Temp directories that you have created

U14: The art of typing less to do more

Saving keystrokes may not seem important, but the longer that you spend typing in a terminal window, the happier you will be if you can reduce the time you spend at the keyboard.

Especially, as prolonged typing is not good for your body. So the best Unix tip to learn early on is that you can [tab complete](#) the names of files and programs on most Unix systems. Type enough letters that uniquely identify the name of a file, directory or program and press tab... Unix will do the rest. E.g. if you type 'tou' and then press tab, Unix will autocomplete the word to touch (which we will learn more about in a minute). In this case, tab completion will occur because there are no other Unix commands that start with 'tou'. If pressing tab doesn't do anything, then you have not have typed enough unique characters. In this case pressing tab twice will show you all possible completions. This trick can save you a LOT of typing...if you don't use tab-completion then you must be a masochist.

Task U14.1

Navigate to your home directory, and then use the cd command to change to the /Volumes/USB/Unix_and_Perl_course/Code/ directory. Use tab completion for each directory name. This should only take 13 key strokes compared to 41 if you type the whole thing yourself.

Another great time-saver is that Unix stores a list of all the commands that you have typed in each login session. You can access this list by using the [history](#) command or more simply by using the up and down arrows to access anything from your history. So if you type a long command but make a mistake, press the up arrow and then you can use the left and right arrows to move the cursor in order to make a change.

U15: U can touch this

The following sections will deal with Unix commands that help us to work with files, i.e. copy files to/from places, move files, rename files, remove files, and most importantly, look at files. Remember, we want to be able to do all of these things without leaving the terminal. First, we need to have some files to play with. The Unix command `touch` will let us create a new, empty file. The `touch` command does other things too, but for now we just want a couple of files to work with.

```
$ cd /Volumes/USB/Unix_and_Perl_course
$ touch heaven.txt
$ touch earth.txt
$ ls
Applications  Code      Data      Documentation  earth.txt      heaven.txt
```

U16: Moving heaven and earth

Now, let's assume that we want to move these files to a new directory ('Temp'). We will do this using the Unix `mv` (move) command:

```
$ mkdir Temp  
$ mv heaven.txt Temp/  
$ mv earth.txt Temp/  
$ ls  
Applications   Code       Data       Documentation   Temp  
$ ls Temp/  
earth.txt heaven.txt
```

For the `mv` command, we always have to specify a source file (or directory) that we want to move, and then specify a target location. If we had wanted to we could have moved both files in one go by typing any of the following commands:

```
$ mv *.txt Temp/  
$ mv *t Temp/  
$ mv *ea* Temp/
```

The asterisk * acts as a [wild-card character](#), essentially meaning 'match anything'. The second example works because there are no other files or directories in the directory that end with the letters 't' (if there was, then they would be copied too). Likewise, the third example works because only those two files contain the letters 'ea' in their names. Using wild-card characters can save you a lot of typing.

Task U16.1

Use `touch` to create three files called 'fat', 'fit', and 'feet' inside the Temp directory. I.e.

```
$ cd Temp  
$ touch fat fit feet
```

Then type either `ls f?t` or `ls f*t` and see what happens. The ? character is also a wild-card but with a slightly different meaning. Try typing `ls f??t` as well.

U17: Renaming files

In the earlier example, the destination for the `mv` command was a directory name (Temp). So we moved a file from its source location to a target location ('source' and 'target' are important concepts for many Unix commands). But note that the target could have also been a (different) file name, rather than a directory. E.g. let's make a new file and move it whilst renaming it at the same time:

```
$ touch rags
$ ls
Applications      Code          Data       Documentation   Temp    rags
$ mv rags Temp/riches
$ ls Temp/
earth.txt        heaven.txt     riches
```

In this example we create a new file ('rags') and move it to a new location and in the process change the name (to 'riches'). So `mv` can rename a file as well as move it. The logical extension of this is using `mv` to rename a file without moving it (you have to use `mv` to do this as Unix does not have a separate 'rename' command):

```
$ mv Temp/riches Temp/rags
$ ls Temp/
earth.txt        heaven.txt     rags
```

U18: Stay on target

It is important to understand that as long as you have specified a ‘source’ and a ‘target’ location when you are moving a file, then it doesn’t matter what your current directory is. You can move or copy things within the same directory or between different directories regardless of whether you are “in” any of those directories. Moving directories is just like moving files:

```
$ mkdir Temp2
$ ls
Applications      Code          Data       Documentation   Temp   Temp2
$ mv Temp2 Temp/
$ ls Temp/
Temp2           earth.txt      heaven.txt  rags
```

This step moves the Temp2 directory inside the Temp directory.

Task U18.1

Create another Temp directory (Temp3) and then change directory to your home directory (/users/clmuser). **Without** changing directory, move the Temp3 directory to inside the /Volumes/USB/Unix_and_Perl_course/Temp directory.

U19: Here, there, and everywhere

The philosophy of ‘not having to be in a directory to do something in that directory’, extends to just about any operation that you might want to do in Unix. Just because we need to do something with file X, it doesn’t necessarily mean that we have to change directory to wherever file X is located. Let’s assume that we just want to quickly check what is in the Data directory before continuing work with whatever we were previously doing in /Volumes/USB/Unix_and_Perl_course. Which of the following looks more convenient:

```
$ cd Data  
$ ls  
Arabidopsis      C_elegans    GenBank      Misc        Unix_test_files  
$ cd ..
```

or...

```
$ ls Data/  
Arabidopsis      C_elegans    GenBank      Misc        Unix_test_files
```

In the first example, we change directories just to run the ls command, and then we change directories back to where we were again. The second example shows how we could have just stayed where we were.

U20: To slash or not to slash?

Task U20.1

Run the following two commands and compare the output

```
$ ls Documentation
```

```
$ ls Documentation/
```

The two examples are not quite identical, but they produce identical output. So does the trailing slash character in the second example matter? Well not really. In both cases we have a directory named ‘Documentation’ and it is optional as to whether you include the trailing slash. When you tab complete any Unix directory name, you will find that a trailing slash character is automatically added for you. This becomes useful when that directory contains subdirectories which you also want to tab complete.

I.e. imagine if you had to type the following (to access a buried directory ‘ggg’) and tab-completion *didn’t* add the trailing slash characters. You’d have to type the seven slashes yourself.

```
$ cd aaa/bbb/ccc/ddd/eee/fff/ggg/
```

U21: The most dangerous Unix command you will ever learn!

You've seen how to remove a directory with the `rmdir` command, but `rmdir` won't remove directories if they contain any files. So how can we remove the files we have created (in `/Volumes/USB/Unix_and_Perl_course/Temp`)? In order to do this, we will have to use the `rm` (remove) command.

Please read the next section VERY carefully. Misuse of the rm command can lead to needless death & destruction

Potentially, `rm` is a very dangerous command; if you delete something with `rm`, you will not get it back! It does not go into the trash or recycle can, it is permanently removed. It is possible to delete everything in your home directory (all directories and subdirectories) with `rm`, that is why it is such a dangerous command.

Let me repeat that last part again. It is possible to delete EVERY file you have ever created with the `rm` command. Are you scared yet? You should be. Luckily there is a way of making `rm` a little bit safer. We can use it with the `-i` command-line option which will ask for confirmation before deleting anything:

```
$ pwd  
/Volumes/USB/Unix_and_Perl_course/Temp  
$ ls  
Temp2      Temp3      earth.txt  heaven.txt  rags  
$ rm -i earth.txt  
remove earth.txt? y  
$ rm -i heaven.txt  
remove heaven.txt? y
```

We could have simplified this step by using a wild-card (e.g. `rm -i *.txt`).

Task U21.1

Remove the last file in the Temp directory ('rags') and then remove the two empty directories (Temp 2 & Temp3).

U22: Go forth and multiply

Copying files with the `cp` (copy) command is very similar to moving them. Remember to always specify a source and a target location. Let's create a new file and make a copy of it.

```
$ touch file1  
$ cp file1 file2  
$ ls  
file1    file2
```

What if we wanted to copy files from a different directory to our current directory? Let's put a file in our home directory (specified by ‘~’ remember) and copy it to the USB drive:

```
$ touch ~/file3  
$ ls  
file1    file2  
$ cp ~/file3 .  
$ ls file1 file2 file3
```

This last step introduces another new concept. In Unix, the current directory can be represented by a ‘.’ (dot) character. You will mostly use this only for copying files to the current directory that you are in. But just to make a quick point, compare the following:

```
$ ls  
$ ls .  
$ ls ./
```

In this case, using the dot is somewhat pointless because `ls` will already list the contents of the current directory by default. Also note again how the trailing slash is optional.

Let's try the opposite situation and copy these files back to the home directory (even though one of them is already there). The default behavior of `copy` is to overwrite (without warning) files that have the same name, so be careful.

```
$ cp file* ~/
```

Based on what we have already covered, do you think the trailing slash in ‘~/’ is necessary?

U23: Going deeper and deeper

The `cp` command also allows us (with the use of a command-line option) to copy entire directories (also note how the `ls` command in this example is used to specify multiple directories):

```
$ mkdir Storage
$ mv file* Storage/
$ ls
Storage
$ cp -R Storage Storage2
$ ls Storage Storage2
Storage:
file1    file2    file3

Storage2:
file1    file2    file3
```

Task U23.1

The `-R` option means ‘copy recursively’, many other Unix commands also have a similar option. See what happens if you don’t include the `-R` option. We’ve finished with all of these temporary files now. Make sure you remove the `Temp` directory and its contents (remember to always use `rm -i`).

U24: When things go wrong

At this point in the course, you may have tried typing some of these commands and have found that things did not work as expected. Some people will then assume that the computer doesn't like them and that it is being deliberately mischievous. The more likely explanation is that you made a typing error. Maybe you have seen one the following error messages:

```
$ ls Codee
ls: Codee: No such file or directory

$ cp Data/Unix_test_files/* Docmentation
usage: cp [-R [-H | -L | -P]] [-fi | -n] [-pvX] source_file target_file
        cp [-R [-H | -L | -P]] [-fi | -n] [-pvX] source_file ... target_directory
```

In both cases, we included a deliberate typo when specifying the name of the directories. With the `ls` command, we get a fairly useful error message. With the `cp` command we get a more cryptic message that reveals the correct usage statement for this command. In general, if a command fails, check your current directory (`pwd`) and check that all the files or directories that you mention actually exist (and are in the right place). Many errors occur because people are not in the right directory!

U25: Less is more

So far we have covered listing the contents of directories and moving/copying/deleting either files and/or directories. Now we will quickly cover how you can look at files; in Unix the [less](#) command lets you view (but not edit) text files. Let's take a look at a file of *Arabidopsis thaliana* protein sequences:

```
$ less Data/Arabidopsis/At_proteins.fasta
```

When you are using less, you can bring up a page of help commands by pressing h, scroll forward a page by pressing space, or go forward or backwards one line at a time by pressing j or k. To exit less, press q (for quit). The less program also does about a million other useful things (including text searching).

U26: Directory enquiries

When you have a directory containing a mixture of files and directories, it is not often clear which is which. One solution is to use `ls -l` which will put a ‘d’ at the start of each line of output for items which are directories. A better solution is to use `ls -p`. This command simply adds a trailing slash character to those items which are directories. Compare the following:

```
$ ls
Applications      Data      file1      Code      Documentation      file2

$ ls -p
Applications/    Data/     file1      Code/     Documentation/   file2
```

Hopefully, you’ll agree that the second example makes things a little clearer. You can also do things like always capitalizing directory names (like I have done) but ideally we would suggest that you always use `ls -p`. If this sounds a bit of a pain, then it is. Ideally you want to be able to make `ls -p` the default behavior for `ls`. Luckily, there is a way of doing this by using Unix [aliases](#). It’s very easy to create an alias:

```
$ alias ls='ls -p'
$ ls
Applications/    Data/     file1      Code/     Documentation/   file2
```

If you have trouble remembering what some of these very short Unix commands do, then aliases allow you to use human-readable alternatives. I.e. you could make a ‘copy’ alias for the `cp` command or even make ‘list_files_sorted_by_date’ perform the `ls -lt` command. Note that aliases do not replace the original command. It can be dangerous to use the name of an existing command as an alias for a different command. I.e. you could make an `rm` alias that put files to a ‘trash’ directory by using the `mv` command. This might work for you, but what if you start working on someone else’s machine who doesn’t have that alias? Or what if someone else starts working on your machine?

Task U26.1

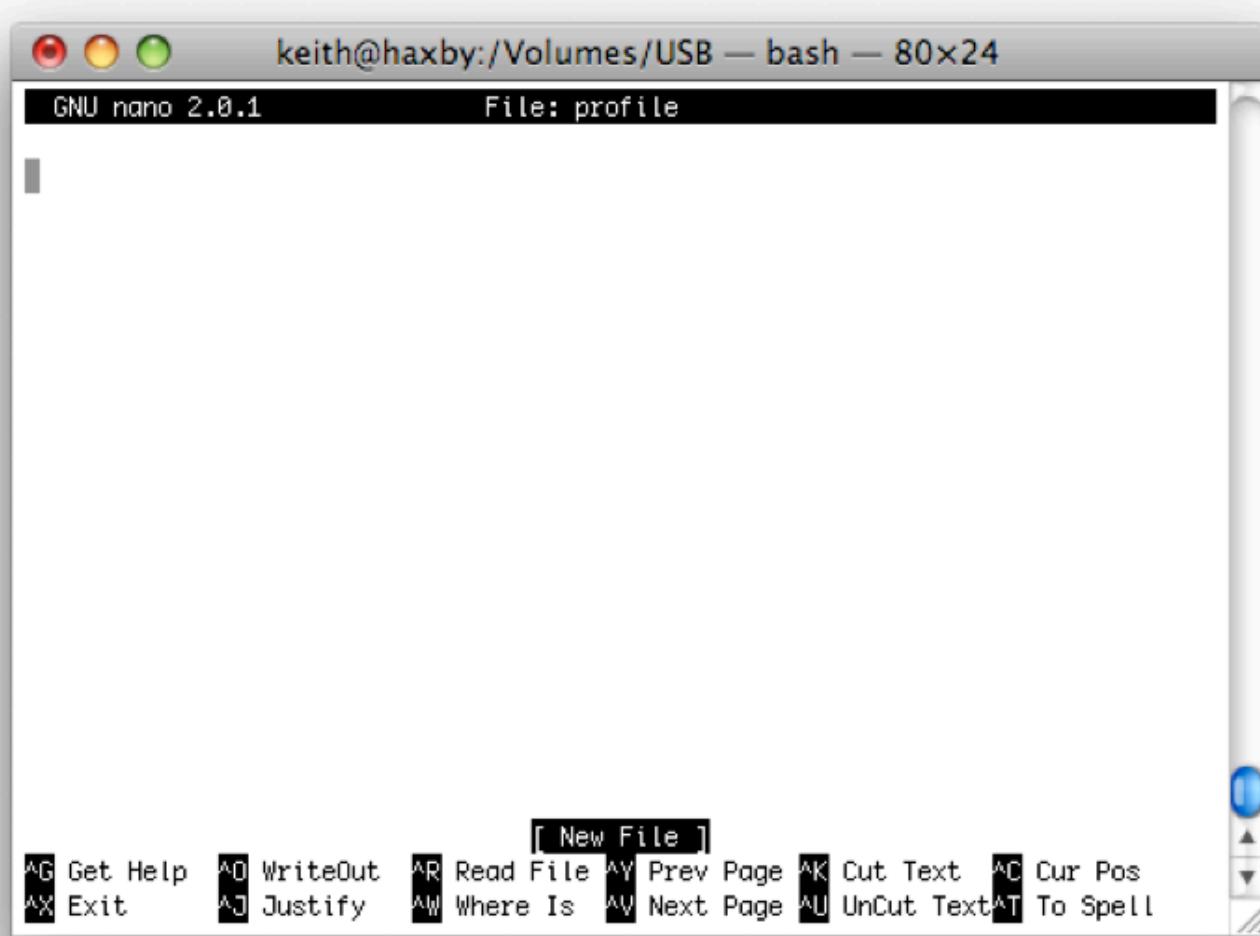
Create an alias such that typing `rm` will always invoke `rm -i`. Try running the alias command on its own to see what happens. Now open a new terminal window (or a new tab) and try running your `ls` alias. What happens?

U27: Fire the editor

The problem with aliases is that they only exist in the current terminal session. Once you log out, or use a new terminal window, then you'll have to retype the alias. Fortunately though, there is a way of storing settings like these. To do this, we need to be able to create a configuration file and this requires using a text editor. We could use a program likeTextEdit to do this (or even Microsoft Word), but as this is a Unix course, we will use a simple Unix editor called ` . Let's create a file called profile:

```
$ cd /Volumes/USB/Unix_and_Perl_course  
$ nano profile
```

You should see the following appear in your terminal:



the nano editor

The bottom of the nano window shows you a list of simple commands which are all accessible by typing ‘Control’ plus a letter. E.g. Control + X exits the program.

Task U27.1

Type the following text in the editor and then save it (Control + O). Nano will ask if you want to ‘save the modified buffer’ and then ask if you want to keep the same name. Then exit nano (Control + X) and use less to confirm that the profile file contains the text you added.

```
# some useful command line short-cuts
alias ls='ls -p'
alias rm='rm -i'
```

Now you have successfully created a configuration file (called ‘profile’) which contains two aliases. The first line that starts with a hash (#) is a comment, these are just notes that you can add to explain what the other lines are doing. But how do you get Unix to recognize the contents of this file? The [source](#) command tells Unix to read the contents of a file and treat it as a series of Unix commands (but it will ignore any comments).

Task U27.2

Open a new terminal window or tab (to ensure that any aliases will not work) and then type the following (make sure you first change to the correct directory):

```
$ source profile
```

Now try the `ls` command to see if the output looks different. Next, use `touch` to make a new file and then try deleting it with the `rm` command. Are the aliases working?

U28: Hidden treasure

In addition to adding aliases, profile files in Unix are very useful for many other reasons. We have actually already created a profile for you. It's in /Volumes/USB/Unix_and_Perl_course but you probably won't have seen it yet. That's because it is a hidden file named '.profile' (dot profile). If a filename starts with a dot, Unix will treat it as a hidden file. To see it, you can use `ls -a` which lists all hidden files (there may be several more files that appear).

Task U28.1

Use `less` to look at the profile file that we have created. See if you can understand what all the lines mean (any lines that start with a # are just comments). Use `source` to read this file. See how this changes the behavior of typing `cd` on its own. You can now delete the profile file that you made earlier, from now on we will use the `.profile` file.

If you have a `.profile` file in your *home* directory then it will be automatically read every time you open a new terminal. A problem for this class is your home directories are wiped each day, so we can't store files on the computer (which is why we are using the USB drive). So for this course we have to do a bit of extra work.

Remember to type:

source /Volumes/USB/Unix_and_Perl_course/.profile
every time you use a new terminal window

U29: Sticking to the script

Unix can also be used as a programming language just like Perl. Depending on what you want to do, a Unix script might solve all your problems and mean that you don't really need to learn Perl at all.

So how do you make a Unix script (which are commonly called 'shell scripts')? At the simplest level, we just write one or more Unix commands to a file and then treat that file as if it was any other Unix command or program.

Task U29.1

Copy the following two lines to a file (using `nano`). Name that file `hello.sh` (shell scripts are typically given a `.sh` extension) and **make sure that you save this file in `/Volumes/USB/Unix_and_Perl_course/Code`.**

```
# my first Unix shell script
echo "Hello World"
```

When you have done that, simply type '`hello.sh`' and see what happens. If you have previously run `source .profile` then you should be able to run '`hello.sh`' from any directory that you navigate to. If it worked, then it should have printed 'Hello world'. This very simple script uses the Unix command `echo` which just prints output to the screen. Also note the comment that precedes the `echo` command, it is a good habit to add explanatory comments.

Task U29.2

Try moving the script outside of the `Code` directory (maybe move it 'up' one level) and then `cd` to that directory. Now try running the script again. You should find that it doesn't work anymore. Now try running `./hello.sh` (that's a dot + slash at the beginning). It should work again.

U30: Keep to the \$PATH

The reason why the script worked when it was in the Code directory and then stopped working when you moved it is because we did something to make the Code directory a bit special. Remember this line that is in your .profile file?

```
PATH=$PATH":$HOME/Code"
```

When you try running *any* program in Unix, your computer will look in a set of predetermined places to see if a program by that name lives there. All Unix commands are just files that live in directories somewhere on your computer. Unix uses something called \$PATH (which is an *environment variable*) to store a list of places to look for programs to run. In our .profile file we have just told Unix to also look in your Code directory. If we didn't add the Code directory to the \$PATH, then we have to run the program by first typing ./ (dot slash). Remember that the dot means the current directory. Think of it as a way of forcing Unix to run a program (including Perl scripts).

U31: Ask for permission

Programs in Unix need permission to be run. We will normally always have to type the following for any script that we create:

```
$ chmod u+x hello.sh
```

This would use the **chmod** to add *executable* permissions (+x) to the file called ‘hello.sh’ (the ‘u’ means add this permission to just you, the user). Without it, your script won’t run. Except that it did. One of the oddities of using the USB drive for this course, is that files copied to a USB drive have all permissions turned on by default. Just remember that you will normally need to run **chmod** on any script that you create. It’s probably a good habit to get into now.

The **chmod** command can also modify read and write permissions for files, and change any of the three sets of permissions (read, write, execute) at the level of ‘user’, ‘group’, and ‘other’. You probably won’t need to know any more about the **chmod** command other than you need to use it to make scripts executable.

U32: The power of shell scripts

Time to make some Unix shell scripts that might actually be useful.

Task U32.1

Look in the Data/Unix_test_files directory. You should see several files (all are empty) and four directories. Now put the following information into a shell script (using nano) and save it as cleanup.sh.

```
#!/bin/bash
mv *.txt Text
mv *.jpg Pictures
mv *.mp3 Music
mv *.fa Sequences
```

Make sure that this script is saved in your Unix_and_Perl_course/Code directory. Now return to the Unix_and_Perl_course/Data/Unix_test_files directory and run this script. It should place the relevant files in the correct directories. This is a relatively simple use of shell scripting. As you can see the script just contains regular Unix commands that you might type at the command prompt. But if you had to do this type of file sorting every day, and had many different types of file, then it would save you a lot of time.

Did you notice the #!/bin/bash line in this script? There are several different types of shell script in Unix, and this line makes it clearer that a) that this is actually a file that can be treated as a program and b) that it will be a bash script (bash is a type of Unix). As a general rule, all type of scriptable programming languages should have a similar line as the first line in the program.

Task U32.2

Here is another script. Copy this information into a file called change_file_extension.sh and again place that file in the Code directory.

```
#!/bin/bash

for filename in *.$1
do
    mv $filename ${filename%$1}$2
done
```

Now go to the Data/Unix_test_files/Text directory. If you have run the exercise from Task [U32.1](#) then your text directory should now contain three files. Run the following command:

```
$ change_file_extension.sh txt text
```

Now run the `ls` command to see what has happened to the files in the directory. You should see that all the files that ended with ‘txt’ now end with ‘text’. Try using this script to change the file extensions of other files.

It’s not essential that you understand exactly how this script works at the moment (things will become clearer as you learn Perl), but you should at least see how a relatively simple Unix shell script can be potentially very useful.

End of part 1.

You can now continue to learn a series of much more powerful Unix commands, or you can switch to [Part 3](#) in order to start learning Perl. The choice is yours!

Part 2: Advanced Unix

How to Become a Unix power user

The commands that you have learnt so far are essential for doing any work in Unix but they don't really let you do anything that is very useful. The following sections will introduce a few new commands that will start to show you how powerful Unix is.

U33: Match making

You will often want to search files to find lines that match a certain pattern. The Unix command `grep` does this (and much more). You might already know that FASTA files (used frequently in bioinformatics) have a simple format: one header line which must start with a ‘>’ character, followed by a DNA or protein sequence on subsequent lines. To find only those header lines in a FASTA file, we can use grep, which just requires you specify a pattern to search for, and one or more files to search:

```
$ cd Data/Arabidopsis/  
$ grep ">" intron_IME_data.fasta  
  
>AT1G68260.1_i1_204_CDS  
>AT1G68260.1_i2_457_CDS  
>AT1G68260.1_i3_1286_CDS  
>AT1G68260.1_i4_1464_CDS  
. . .
```

This will produce lots of output which will flood past your screen. If you ever want to stop a program running in Unix, you can type Control+C (this sends an interrupt signal which should stop most Unix programs). The grep command has many different command-line options (type `man grep` to see them all), and one common option is to get grep to show lines that don’t match your input pattern. You can do this with the `-v` option and in this example we are seeing just the sequence part of the FASTA file.

```
$ grep -v ">" intron_IME_data.fasta
```

```
GTATACACATCTCTACTTCAATTGCATCTAACGAAATCGGATTCCGTCGTTG  
TGAAATTGAGTTTCGGATTCAGTGGTCTGAGATTCTATATCTGATTGAGTCTAAT  
GATTCTGATTGAAAATCTCGCTATTGTACAG  
GTTAGTTTCAATGTTGCTGCTTCTGATTGTTGAAAGTGTTCATACATTGTGAATTAG  
TTGATAAAATCTGAACTCTGCATGATCAAAGTTACTCCTTACTTAGTTGACAGGGACT  
TTTTTGTGAATGTGGTTGAGTAGAATTAGGGCTTGGATTAAATGTGACAAGATTG  
.  
.  
.
```

U34: Your first ever Unix pipe

By now, you might be getting a bit fed up of waiting for the grep command to finish, or you might want a cleaner way of controlling things without having to reach for Ctrl-C. Ideally, you might want to look at the output from any command in a controlled manner, i.e. you might want to use a Unix program like less to view the output.

This is very easy to do in Unix, you can send the output from any command to any other Unix program (as long as the second program accepts input of some sort). We do this by using what is known as a [pipe](#). This is implemented using the ‘|’ character (which is a character which always seems to be on different keys depending on the keyboard that you are using). Think of the pipe as simply connecting two Unix programs. In this next example we send the output from grep down a pipe to the less program. Let’s imagine that we just want to see lines in the input file which contain the pattern “ATGTGA” (a potential start and stop codon combined):

```
$ grep "ATGTGA" intron_IME_data.fasta | less
```



```
TTTTTGTGAATGTGGTTGAGTAGAATTAGGGCTTGGATTAAATGTGACAAGATTTG
CTGAATGTGACTGGAAGAACATGAAATGTGTTAAGATCTTGTTCGTTAAGTTAGAGTCTTG
GGTGGAAATGAATTATGTATCATGTGATAGCTGTTGCATTACAAGATGTAATTTGCAAA
GTCTATGTGATGCCATAGCCCATAGTGACTGATAGCTCCTTACCTTGTAAAAAATTTCT
TTACTTGCAAAATCCATGTGATTTTATATTACTTGAAGAATTATAATATATTTT
TTGCATCAAGATATGTGACATCTCAAAAAGATAACTTGTGAGAAGACAATTATAATATG
GTAACCTTTGATTGAATCAGTAACTGTATTGTTATCATGATTGTGAATATGTGA
AATCTTGTGGTGGCTACGATATGAGCTGTCAATATATTTGTTACATGTGATC
GTATGTGAGCAAACGATGTCTCGTTCTCTCTCAATGATCAAGCACCTAAAT\\
.
.
.
```

Notice that you still have control of your output as you are now in the less program. If you press the forward slash (/) key in less, you can then specify a search pattern. Type ATGTGA after the slash and press enter. The less program will highlight the location of these matches on each line. Note that grep matches patterns on a per line basis. So if one line ended ATG and the next line started TGA, then grep would not find it.

Any time you run a Unix program or command that outputs a lot of text to the screen, you can instead pipe that output into the less program.

U35: Heads and tails

Sometimes we do not want to use less to see *all* of the output from a command like grep. We might just want to see a few lines to get a feeling for what the output looks like, or just check that our program (or Unix command) is working properly. There are two useful Unix commands for doing this: head and tail. These commands show (by default) the first or last 10 lines of a file (though it is easy to specify more or fewer lines of output). So now, let's look for another pattern which might be in all the sequence files in the directory. If we didn't know whether the DNA/protein sequence in a FASTA files was in upper-case or lower-case letters, then we could use the -i option of grep which 'ignores' case when searching:

```
$ grep -i ACGTC * | head
At_proteins.fasta:TYRSPRCNSAVCSRAGSIACGTCFSPPRPGCSNNTGAFPDNSITGWATSGEFLDVVSIO
STNGSNPGRFVKIPNLIFS
At_proteins.fasta:FRRYGHYISSDVFRFKGSNGNFKESLTGYAKGMLSLYEAAHLGTTKDYILQEALSFTSSH
LESLAACGTCPPHLSVHIQ
At_proteins.fasta:MAISKALIASLLISLLVLQLVQADVENSQKKNGYAKKIDCGSACVARCRLSRRPRLCHRAC
GTCCYRCNCVPPGTYGNYD
At_proteins.fasta:MAVFRVLLASLLISLLVLDFVHADMVTSNDAPKIDCNSRCQERCSLSSRPNLCHRACGTCC
ARCNCVAPGTSGNYDKCPC
chr1.fasta:TGTCTACTGATTGATTTCTAAACTGTTGATTGTTCAAGGTCAACCAATCACGTCAACGAAAT
TCAGGATCTTA
chr1.fasta:TATGCTGCAAGTACCACTGGAAACTATAAACATGTATAATCAACCAATGAACACG
TCAATAACCTA
chr1.fasta:TTAACAGCTTAGGGTAAAATTATGATCCGTAGAGACAGCATTAAAAGTTCTTACGTCCACGTAA
AATAATATATC
chr1.fasta:GGGATCACGAGTCTGTTGAGTTCCGACGTCGCTTGGTGTACCACTTGTCAACATGTGTTCTT
CTCCGGAGGTG
chr1.fasta:CTGCAAAGGCCTACCTGTTGCCCTGTTACTGACAATACGTCTATGGAACCCATAAAAGGGATCAAC
TGGGAATTGGT
chr1.fasta:ACGTCGAAGGGGGTAAGATTGCAGCTAATCATTGATGAAATGGATTGGATTCACGTGGAGGATGAT
CCTGATGAAGT
```

The * character acts as a wildcard meaning ‘search all files in the current directory’ and the head command restricts the total amount of output to 10 lines. Notice that the output also includes the name of the file containing the matching pattern. In this case, the grep command finds the ACGTC pattern in four protein sequences and several lines of the the chromosome 1 DNA sequence (we don’t know how many exactly because the head command is only giving us ten lines of output).

U36: Getting fancy with regular expressions

A concept that is supported by many Unix programs and also by most programming languages (including Perl) is that of using [regular expressions](#). These allow you to specify search patterns which are quite complex and really help restrict the huge amount of data that you might be searching for to some very specific lines of output. E.g. you might want to find lines that start with an ‘ATG’ and finish with ‘TGA’ but which have at least three AC dinucleotides in the middle:

```
$ grep "^\u00d7ATG.*ACACAC.*TGA$" chr1.fasta
```

```
ATGAACCTTGTACTTCACCGGGTGCCTCAAAGACGTTCTGCTCGGAAGGTTGTCTTACACACTTGATGTCAAATGA  
ATGATAGCTCAACCACGAAATGTCATTACCTGAAACCCCTAAACACACTCTACCTCAAACACTTACTGGTAAAACATTGA  
ATGCATACCTCAGTTGCATCCGGCGCAGGGCAAGCATAACCGCTCAACACACACTGCTTGAGTTGAGCTCCATTGA
```

You’ll learn more about regular expressions when you learn Perl. The ^ character is a special character that tells grep to only match a pattern if it occurs at the start of a line. Similarly, the \$ tells grep to match patterns that occur at the end of the line.

Task U36.1

The . and * characters are also special characters that form part of the regular expression. Try to understand how the following patterns all differ. Try using each of these these patterns with grep against any one of the sequence files. Can you predict which of the five patterns will generate the most matches?

```
ACGT  
AC.GT  
AC*GT  
AC.*GT
```

The asterisk in a regular expression is similar to, but NOT the same, as the other asterisks that we have seen so far. An asterisk in a regular expression means: ‘match zero or more of the preceding character or pattern’.

Try searching for the following patterns to ensure you understand what . and * are doing:

A...T

AG*T

A*C*G*T*

U37: Counting with grep

Rather than showing you the lines that match a certain pattern, grep can also just give you a count of how many lines match. This is one of the frequently used grep options. Running grep -c simply counts how many lines match the specified pattern. It doesn't show you the lines themselves, just a number:

```
$ grep -c i2 intron_IME_data.fasta  
9785
```

Task U37.1

Count how many times each pattern from Task [U36.1](#) occurs in all of the sequence files (specifying * .fasta will allow you to specify all sequence files).

U38: Regular expressions in less

You have seen already how you can use `less` to view files, and also to search for patterns. If you are viewing a file with `less`, you can type a forward-slash / character, and this allows you to then specify a pattern and it will then search for (and highlight) all matches to that pattern. Technically it is searching forward from whatever point you are at in the file. You can also type a question-mark ? and `less` will allow you to search backwards. The real bonus is that the patterns you specify can be regular expressions.

Task U38.1

Try viewing a sequence file with `less` and then searching for a pattern such as `ATCG.*TAG$`. This should make it easier to see exactly where your regular expression pattern matches. After typing a forward-slash (or a question-mark), you can press the up and down arrows to select previous searches.

U39: Let me transl(ite)r(ate) that for you

We have seen that these sequence files contain upper-case characters. What if we wanted to turn them into lower-case characters (because maybe another bioinformatics program will only work if they are lower-case)? The Unix command `tr` (short for transliterate) does just this, it takes one range of characters that you specify and changes them into another range of characters:

```
$ head -n 2 chr1.fasta

>Chr1 dumped from ADB: Mar/14/08 12:28; last updated: 2007-12-20
CCCTAAACCTAAACCCTAAACCTAAACCTCTGAATCCTTAATCCCTAAATCCCTAAATCTTAAATCCTACATCCAT

$ head -n 2 chr1.fasta | tr 'A-Z' 'a-z'

>chr1 dumped from adb: mar/14/08 12:28; last updated: 2007-12-20
ccctaaacccctaaaccctaaaccctaaacctctgaatccttaatccctaaatccctaaatcttaaatcctacatccat
```

U40: That's what she sed

The `tr` command let's you change a range of characters into another range. But what if you wanted to change a particular pattern into something completely different? Unix has a very powerful command called `sed` that is capable of performing a variety of text manipulations. Let's assume that you want to change the way the FASTA header looks:

```
$ head -n 1 chr1.fasta >Chr1 dumped from ADB: Mar/14/08 12:28; last updated: 2007-12-20

$ head -n 1 chr1.fasta | sed 's/Chr1/Chromosome 1/' >Chromosome 1 dumped from ADB: Mar/14/08 12:28; last updated: 2007-12-20
```

The 's' part of the `sed` command puts `sed` in 'substitute' mode, where you specify one pattern (between the first two forward slashes) to be replaced by another pattern (specified between the second set of forward slashes). Note that this doesn't actually change the contents of the file, it just changes the screen output from the previous command in the pipe. We will learn later on how to send the output from a command into a new file.

U41: Word up

For this section we want to work with a different type of file. It is sometimes good to get a feeling for how large a file is before you start running lots of commands against it. The `ls -l` command will tell you how big a file is, but for many purposes it is often more desirable to know how many ‘lines’ it has. That is because many Unix commands like `grep` and `sed` work on a line by line basis. Fortunately, there is a simple Unix command called `wc` (word count) that does this:

```
$ cd Data/Arabidopsis/ $ wc At_genes.gff 531497 4783473 39322356 At_genes.gff
```

The three numbers in the output above count the number of lines, words and bytes in the specified file(s). If we had run `wc -l`, the `-l` option would have shown us just the line count.

U42: GFF and the art of redirection

The Arabidopsis directory also contains a [GFF file](#). This is a common file format in bioinformatics and GFF files are used to describe the location of various features on a DNA sequence. Features can be exons, genes, binding sites etc, and the sequence can be a single gene or (more commonly) an entire chromosome.

This GFF file describes of all of the gene-related features from chromosome I of *A. thaliana*. We want to play around with some of this data, but don't need all of the file...just 10,000 lines will do (rather than the ~500,000 lines in the original). We will create a new (smaller) file that contains a subset of the original:

```
$ head -n 10000 At_genes.gff > At_genes_subset.gff
$ ls -l
total 195360
-rwxrwxrwx 1 keith staff 39322356 Jul  9 15:02 At_genes.gff
-rwxrwxrwx 1 keith staff 705370 Jul 10 13:33 At_genes_subset.gff
-rwxrwxrwx 1 keith staf f 17836225 Oct  9 2008 At_proteins.fasta
-rwxrwxrwx 1 keith staff 30817851 May  7 2008 chr1.fasta
-rwxrwxrwx 1 keith staff 11330285 Jul 10 11:11 intron_IME_data.fasta
```

This step introduces a new concept. Up till now we have sent the output of any command to the screen (this is the default behavior of Unix commands), or through a pipe to another program. Sometimes you just want to redirect the output into an actual file, and that is what the `>` symbol is doing, it acts as one of three [redirection operators](#) in Unix.

As already mentioned, the GFF file that we are working with is a standard file format in bioinformatics. For now, all you really need to know is that every GFF file has 9 fields, each separated with a tab character. There should always be some text at every position (even if it is just a '.' character). The last field often is used to store a lot of text.

U43: Not just a pipe dream

The 2nd and/or 3rd fields of a GFF file are usually used to describe some sort of biological feature. We might be interested in seeing how many different features are in our file:

```
$ cut -f 3 At_genes_subset.gff | sort | uniq
```

CDS
chromosome
exon
five_prime_UTR
gene
mRNA
miRNA
ncRNA
protein
pseudogene
pseudogenic_exon
pseudogenic_transcript
snoRNA
tRNA
three_prime_UTR
transposable_element_gene

In this example, we combine three separate Unix commands together in one go. Let's break it down (it can be useful to just run each command one at a time to see how each additional command is modifying the preceding output):

1. The `cut` command first takes the `At_genes_subset.gff` file and ‘cuts’ out just the 3rd column (as specified by the `-f` option). Luckily, the default behavior for the `cut` command is to split text files into columns based on tab characters (if the columns were separated by another character such as a comma then we would need to use another command line option to specify the comma).
2. The `sort` command takes the output of the `cut` command and sorts it alphanumerically.
3. The `uniq` command (in its default format) only keeps lines which are unique to the output (otherwise you would see thousands of fields which said ‘curated’, ‘Coding_transcript’)

etc.)

Now let's imagine that you might want to find which features start earliest in the chromosome sequence. The start coordinate of features is always specified by column 4 of the GFF file, so:

```
$ cut -f 3,4 At_genes_subset.gff | sort -n -k 2 | head  
  
chromosome 1  
exon      3631  
five_prime_UTR 3631  
gene      3631  
mRNA      3631  
CDS 3760  
protein   3760  
CDS 3996  
exon      3996  
CDS 4486
```

Here we first cut out just two columns of interest (3 & 4) from the GFF file. The `-f` option of the `cut` command lets us specify which columns we want to remove. The output is then sorted with the `sort` command. By default, `sort` will sort alphanumerically, rather than numerically, so we use the `-n` option to specify that we want to sort numerically. We have two columns of output at this point and we could sort based on either column. The `-k 2` specifies that we use the second column. Finally, we use the `head` command to get just the 10 rows of output. These should be lines from the GFF file that have the lowest starting coordinate.

U44: The end of the line

When you press the return/enter key on your keyboard you may think that this causes the same effect no matter what computer you are using. The visible effects of hitting this key are indeed the same...if you are in a word processor or text editor, then your cursor will move down one line. However, behind the scenes pressing enter will generate one of two different events (depending on what computer you are using). Technically speaking, pressing enter generates a newline character which is represented internally by either a *line feed* or *carriage return* character (actually, Windows uses a combination of both to represent a newline). If this is all sounding confusing, well it is, and it is [even more complex](#) than we are revealing here.

The relevance of this to Unix is that you will sometimes receive a text file from someone else which looks fine on their computer, but looks unreadable in the Unix text viewer that you are using. In Unix (and in Perl and other programming languages) the patterns `\n` and `\r` can both be used to denote newlines. A common fix for this requires substituting `\r` for `\n`.

Use `less` to look at the `Data/Misc/excel_data.csv` file. This is a simple 4-line file that was exported from a Mac version of Microsoft Excel. You should see that if you use `less`, then this appears as one line with the newlines replaced with `^M` characters. You can convert these carriage returns into Unix-friendly line-feed characters by using the `tr` command like so:

```
$ cd Data/Misc  
$ tr '\r' '\n' < excel_data.csv  
sequence 1,acacagagag  
sequence 2,acacaggggaaa  
sequence 3,ttcacagaga  
sequence 4,cacacccaaacac
```

This will convert the characters but not save the resulting output, if you wanted to send this output to a new file you will have to use a second redirect operator:

```
$ tr '\r' '\n' < excel_data.csv > excel_data_formatted.csv
```

U45: This one goes to 11

Finally, let's parse the Arabidopsis `intron_IME_data.fasta` file to see if we can extract a subset of sequences that match criteria based on something in the FASTA header line. Every intron sequence in this file has a header line that contains the following pieces of information:

- gene name
- intron position in gene
- distance of intron from transcription start site (TSS)
- type of sequence that intron is located in (either CDS or UTR)

Let's say that we want to extract five sequences from this file that are: a) from first introns, b) in the 5' UTR, and c) closest to the TSS. Therefore we will need to look for FASTA headers that contain the text 'i1' (first intron) and also the text '5UTR'.

We can use grep to find header lines that match these terms, but this will not let us extract the associated sequences. The distance to the TSS is the number in the FASTA header which comes after the intron position. So we want to find the five introns which have the lowest values.

Before I show you one way of doing this in Unix, think for a moment how you would go about this if you didn't know any Unix or Perl...would it even be something you could do without manually going through a text file and selecting each sequence by eye? Note that this Unix command is so long that — depending on how you are viewing this document — it may appear to wrap across two lines. When you type this, it should all be on a single line:

```

$ tr '\n' '@' < intron_IME_data.fasta | sed 's/>/#/g' | tr '#' '\n' | grep "i1
_.*5UTR" | sort -nk 3 -t "_" | head -n 5 | tr '@' '\n'

>AT4G39070.1_i1_7_5UTR
GTGTGAAACCAAAACCAAAACAAGTCAATTGGGGCATTGAAAGCAAAGGAGAGAGTAG
CTATCAAATCAAGAAAATGAGAGGAAGGAGTTAAAAAGACAAAGGAAACCTAAGCTGCT
TATCTATAAGCCAACACATTATTCTTACCCCTTGCCACACTATACCCCATCACCT
CTACATACACTCACCCACATGAGTGTCTACATAAACACTACTATATAGTACTGGTCCA
AAGGTACAAGTTGAGGGAG

>AT5G38430.1_i1_7_5UTR
GCTTTTG CCTCTACGGTCTCACTATATAAGATGACAAAACCAATAGAAAAACAATT
AAG

>AT1G31820.1_i1_14_5UTR
GTTTGTACTTCTTACCTCTCGTAAATGTTAGACTTCGTATAAGGATCCAAGAATT
TCTGATTGTTTTTTCTTGTGTTGATTAG

>AT3G12670.1_i1_18_5UTR
GTAGAATTCGTAAATTCTCTGCTCACTTATTGTTGACTCATACCGATAATCTCT
TCTATGTTGGTAGAGATATCTTCTCAAAGTCTTACCGTGTCTGTGTT
TTTGATGATTTAG

>AT1G26930.1_i1_19_5UTR
GTATAATATGAGAGATAGACAAATGTAAGAAAAACACAGAGAGAAAATTAGTTAATT
ATCTCTCAAATATACAAATATTAAAACCTCTTCTTCAATTACAATTCTCATT
TTTTCTTGTCTTATATTGTAGTTGCAAGAAAGTTAAAAGATTTGACTTTCTTGT
CAG

```

That's a long command, but it does a lot. Try to break down each step and work out what it is doing (you will need to consult the man page for some commands maybe). Notice that I use one of the other redirect operators < to read from a file. It took seven Unix commands to do this, but these are all relatively simple Unix commands; it is the combination of them together which makes them so powerful. One might argue that when things get this complex with Unix that it might be easier to do it in Perl!

Summary

Congratulations are due if you have reached this far. If you have learnt (and understood) all of the Unix commands so far then you probably will never need to learn anything more in order to do a lot of productive Unix work. But keep on dipping into the man page for all of these commands to explore them in even further detail.

The following table provides a reminder of most of the commands that we have covered so far. If you include the three, as-yet-unmentioned, commands in the last column, then you will probably be able to achieve >95% of everything that you will ever want to do in Unix (remember, you can use the `man` command to find out more about `top`, `ps`, and `kill`). The power comes from how you can use combinations of these commands.

The absolute basics		File control	Viewing, creating, or editing files	Misc. useful commands	Power commands	Process-related commands
ls	mv		less	man	uniq	top
cd	cp		head	chmod	sort	ps
pwd	mkdir		tail	source	cut	kill
	rmdir		touch	wc	tr	
	rm		nano		grep	
	(pipe)				sed	
	> (write to file)					
	< (read from file)					

Part 3: Perl

Your programming environment

For this course, you will be using two applications, a code editor and a terminal. You should already be familiar with the Terminal application from the Unix lessons. If you are using a Mac then we recommend using a free code editor such as [Fraise](#), [Tincta](#), or [Text Wrangler](#). A copy of Fraise is provided in `/Volumes/USB/Unix_and_Perl_course/Applications`.

Windows users might want to consider using [Notepad++](#). But please remember that there are many more editors out there and Wikipedia has a useful page [comparing many of them](#). All of these editors will share several useful features such as syntax highlighting, automatic indentation, line numbering, and advanced search & replace.

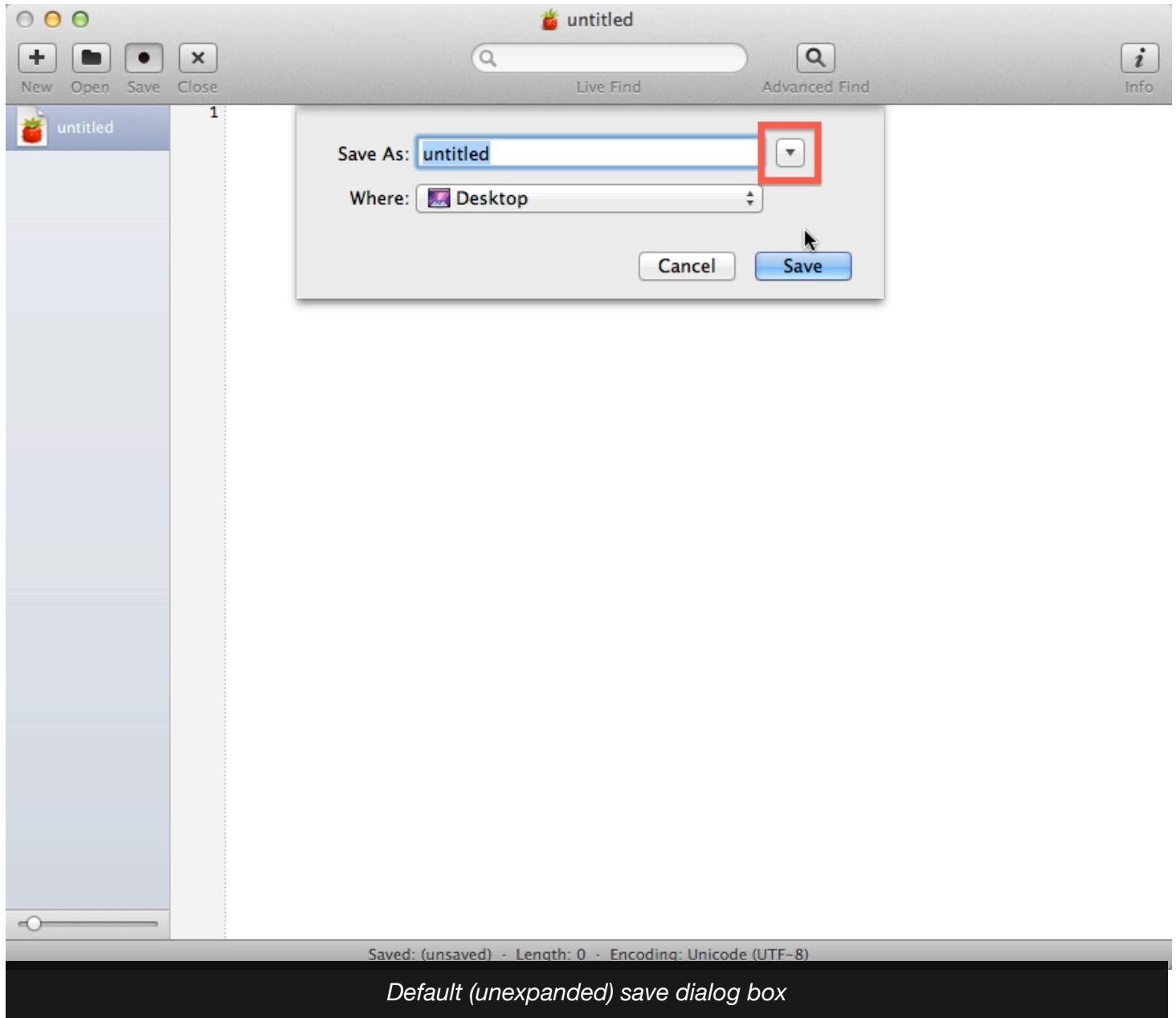
Remember to type: source

/Volumes/USB/Unix_and_Perl_course/.profile at the beginning of every session

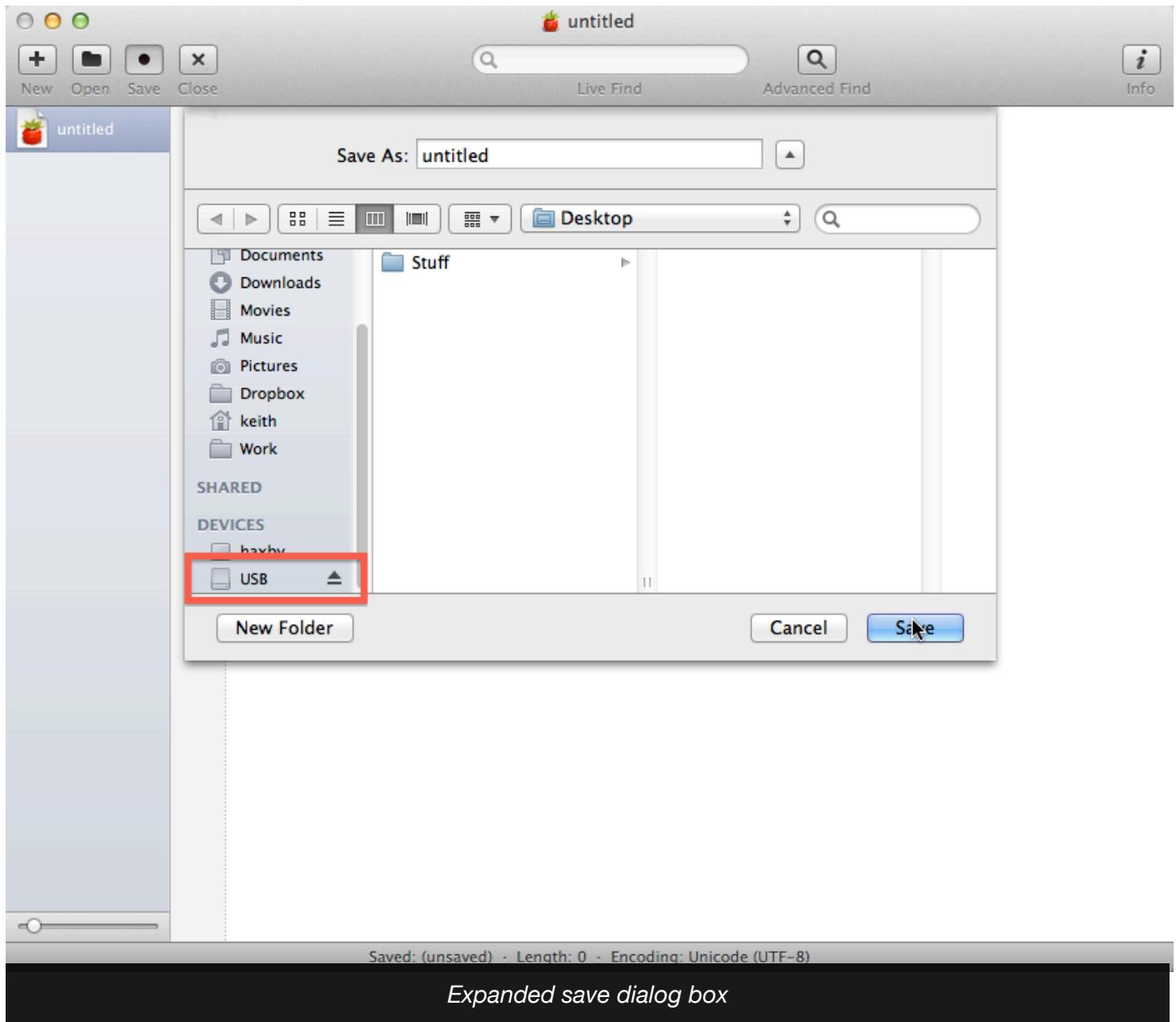
Where to save your Perl scripts

Every time you write a script you should save it in the `Unix_and_Perl_course/Code` directory. This is because we have specified this directory to be part of your Unix PATH (see section [U30](#)). If you keep your Perl scripts here then you can call them from any directory.

If you are new to Macs then it can be confusing to find out how to save a file to specific directory. When you click on the Save button in your code editor the default is to offer to save the file on the Desktop. Click on the *disclosure triangle* and this will expand the save dialog sheet and let you other folders and drives as the save destination:

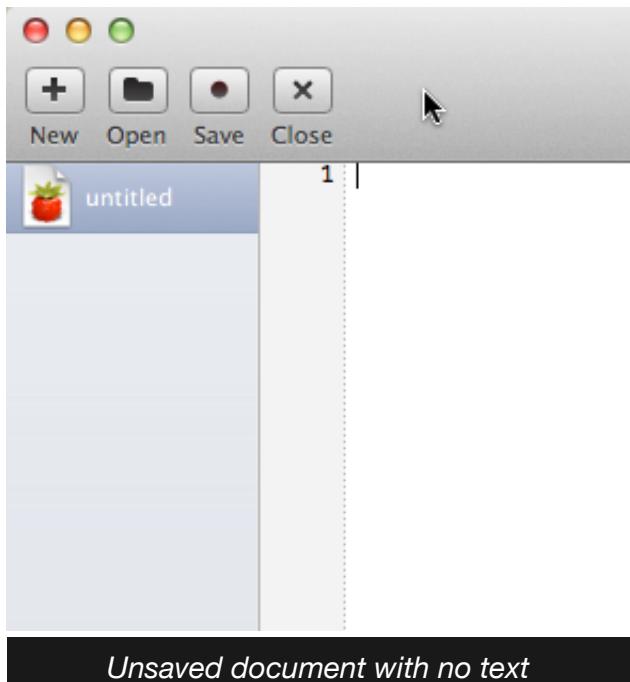


You should now be able to select your USB drive from the list of devices on the left hand side of the save sheet (you might need to scroll to make this available):

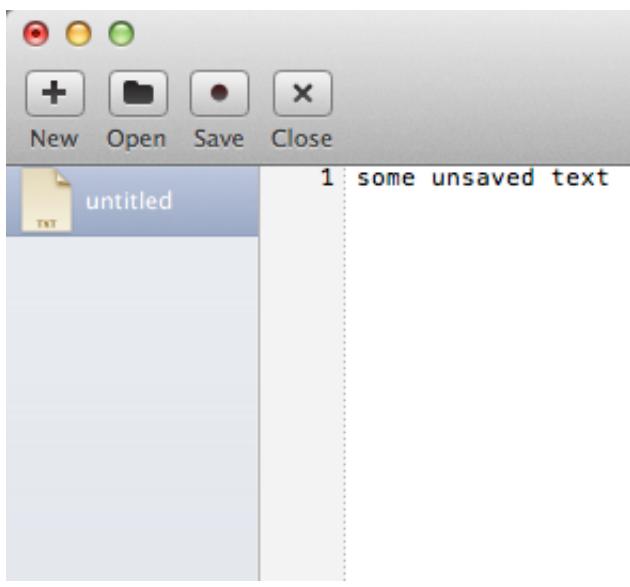


When to save your Perl scripts

Here is a handy Mac tip that will apply to Fraise and also to any other Mac graphical application that allows you to edit and save text. When you first open a new empty document, the program is — as yet — unsaved. If you haven't written anything then this is not a problem, and the top left corner of your application should look like this:



Now notice what happens when you start entering text into the main Fraise window. The window's 'close' button (the red circle in the top left of the window), now has a small black dot inside it:



This is meant to serve as a visual reminder that your file is still unsaved. As soon as you click the 'Save' button, this black dot will disappear. From time to time you will have problems with your Perl scripts, and this might simply be because you have not saved any changes that you have made.

P1. Hello World

The first program you write in any language is always called **Hello World**. The purpose of this program is to demonstrate that the programming environment is working, so the program is as simple as possible.

Task P1.1

Enter the text below into your text editor, but do not include the numbers. The numbers are there only so that we can reference specific lines.

```
1. # helloworld.pl by _insert_your_name_here_
2. print("Hello World!\n");
```

Line 1 has a # sign on it. When Perl sees a # sign, everything that follows on that line is considered a comment. Programmers use comments to describe what a program does, who wrote the program, what needs to be fixed, etc. It's a good idea to put comments in your code, especially as they grow larger.

Line 2 is the only line of this program that does anything. The `print()` function outputs its arguments to your terminal. In this case, there is only one argument, the text "Hello World\n". The funny \n at the end is a newline character, which is like a carriage return. Most of the time, Perl statements end with a semicolon. This is like a period at the end of a sentence. The last statement in a block does not require a semicolon. We will revisit this in a later lesson.

Save the program as `helloworld.pl` (in your Code directory). To run the program, type the following in the terminal and hit return (making sure you have first changed directory to your 'Code' directory).

```
$ perl helloworld.pl
```

This will run the perl program and tell it to execute the instructions of the `helloworld.pl` file. If it worked, great. If it doesn't work, then you may see an error message like the one below:

```
Can't open perl script "helloworld.pl": No such file or directory
```

If you see this, you may have forgotten to save the file, misspelled the file name, or saved the file to someplace unintended. Always use tab-completion to prevent spelling mistakes. Always save your programs to the `Unix_and_Perl_course/Code` directory (for now anyway).

Task P1.2

Modify the program to output some other text, for example the date. Add a few more print statements and experiment with what happens if you omit or add extra newlines.

Task P1.3

Make a few deleterious mutations to your program. For example, leave off the semicolon or one of the parentheses. Observe the error messages. One of the most important aspects of programming is debugging. Probably more time is spent debugging than programming, so it's a good idea to start recognizing errors now.

P2. Scalar variables

Variables hold data. In Perl, the main variable type is called a scalar variable. A scalar holds one thing. This thing could be a number, some text, or an entire genome. We will see other data types later. You can always tell a scalar variable because it has a \$ on the front (the dollar sign is a mnemonic for scalar). For example, a variable might be named \$x. When speaking aloud, we do not say “dollar x”. We just call it “x”.

Task P2.1

Create a new blank text document. Enter the text below and save this program as `scalar.pl` in your Code directory.

```
1.  #!/usr/bin/perl
2.  # scalar.pl by _insert_your_name_here_
3.  use warnings;
4.
5.  $x = 3;
6.  print($x, "\n");
```

Line 1 will appear at the top of every Perl script that we write from now on. This line of code is very similar to the line that appeared at the top of our Unix shell script. It lets Unix know that the Perl program (located at `/usr/bin/perl`) can read this file and run the remaining code inside it.

Line 2 is simply a comment. You should always include a few comments in your programs.

Line 3 is another line that we will add to every script from now on. This line effectively tells Perl that we would like to be warned if we start writing certain types of ‘bad’ code. This is a good thing! We will return to this later on.

Line 4 is deliberately blank. You should use spaces and blank lines to improve the readability of your code. In this case we are separating the first three lines of the script (which don’t actually calculate anything) from the rest.

Line 5 is a variable assignment. The variable `$x` gets the value of 3

Line 6 prints the value of `$x` and then print a newline. As you can see, the `print()` function can take multiple arguments separated by commas.

Run the program by typing the line below in your terminal. Observe the output and go back through the code and line descriptions to make sure you understand everything.

```
$ scalar.pl
```

The addition of `#!/usr/bin/perl` to the script means that we no longer have to type:

```
$ perl scalar.pl
```

What is actually happening here is that we are making it clear that these text files contain instructions written in Perl. The line that we added tells Unix that it should expect to find a program called perl in the `/usr/bin` directory and that program should be capable of making sense of your Perl commands. Now try adding the following lines to your program.

```
7. $s = "something";
8. print($s, "\n");
9. print("$s\n");
```

Line 7 is another variable assignment, but unlike `$x`, our new variable `$s` gets a character string, which is just another term for text.

Lines 8–9 print our new variable `$s` and then print a newline character.

Save the script and run it again. You should see that although lines 8–9 are different they produce exactly the same output. The `print` function can print a list of items (all separated by commas), but it often makes more sense to print just one thing instead. It would have been possible to rewrite our very first Perl script with the following:

```
print("H", "e", "l", "l", "o", " ", "W", "o", "r", "l", "d", "!", "\n");
```

Hopefully you will agree that printing this phrase as one string and not thirteen separate strings is a lot easier on the eye. Now add the following line to your program, and run it again.

```
10. print "$s\n";
```

Line 10 calls the `print` function without parentheses. You do not have to use parentheses for Perl functions, but they are often useful to keep a line organized. In most cases, you will see the `print` function without parentheses. Now add the final two lines to the program:

```
11. print '$x $s\n';
12. print "$x $s\n";
```

Line 11 puts the two variables between single quotes. Any text between single quotes will print exactly as shown. This also means that `\n` loses its special meaning as a newline character. In contrast, strings between double quotes will undergo a process known as *variable interpolation*. This means that variables are always expanded inside double quotes, and `print` will always show what those variables contain.

Task P2.2

Mutate your program. Delete a `$` and see what error message you get.

Task P2.3

Modify the program by changing the contents of the variables. Observe the output. Try experimenting by creating more variables.

Variables summary

You can use (almost) anything for your variable names, though you should try to use names which are descriptive and not too long. You should also use lower case names for your variables. This is not essential though. Which of the following is the best variable name for a variable that will store a DNA sequence?

```
$x = "ATGCAGTGA";          # $x is not a good choice
$dna_sequence_variable;    # also not a good choice, too long
$sequence = "ATGCAGTGA";   # $sequence is better
$dna = "ATGCAGTAGA";      # $dna is even better
```

It is perfectly fine to give a variable the same name as an existing function in Perl though this might be confusing. I.e. a variable named `$print` might look a bit too similar to the `print()` function. Sometimes though the choice of variable name is obvious: `$length` is often a good name for variables that contain the length of something, even though there is also a `length()` function in Perl (which we will learn about later on).

As shown in the example above, variable names can contain underscore characters to separate ‘words’. This is often useful and helps make things easier to understand. E.g.

```
$first_name = "Keith";  
$second_name = "Bradnam";
```

Finally, you should be aware that (with a few exceptions) you can use spaces to make things clearer (or less clear if you so desire). The following lines are all treated by Perl in exactly the same way:

```
$dna = "ATGCAGTGA";           # one space either side of the '=' sign  
$dna="ATGCAGTGA";            # no spaces either side of the '=' sign  
$dna      =      "ATGCAGTGA"; # lots of spaces!
```

P3. Safer programming: use strict

Task 3.1

Create the following program, but don't run it yet. Instead try to predict what it will do. Knowing how a program should work (before you run it) is a good programming skill to develop. If you don't understand what a program should be doing, then you will probably not realize if it is doing something wrong.

```
1. #!/usr/bin/perl
2. # strict.pl by _insert_your_name_here_
3. use strict; use warnings;
4.
5. $pi = 3.14;
6. print "pi = $pi\n";
7.
8. $pi = 3.141593;
9. print "pi = $pi\n";
```

You hopefully noticed that this program introduces another new concept; line 3 includes another usage statement: `use strict;` (in addition to `use warnings;`). Up till now we have ended each line of Perl code with a semi-colon, but there are times when it is simpler to put two lines of Perl code into one line in an editor. Perl will still treat these as two separate lines of code.

Telling Perl to `use strict` means that Perl will insist your script is written in a certain way which is widely considered to be a ‘better’ way of writing code. At this point it is not important to go into the details of what exactly `use strict` is doing. Just accept our word that including a `use strict; use warnings;` line in every script that you write is a good thing to do (we will return to these issues later).

Task 3.2

Now try running the script. You should hopefully see the following errors:

```
Global symbol "$pi" requires explicit package name at strict.pl line 5.  
Global symbol "$pi" requires explicit package name at strict.pl line 6.  
Global symbol "$pi" requires explicit package name at strict.pl line 8.  
Global symbol "$pi" requires explicit package name at strict.pl line 9.  
Execution of strict.pl aborted due to compilation errors.
```

We see one error message for each use of the \$pi variable in the script. Now see what happens if you remove the `use strict;` statement and re-run the script. It should now work. What is happening here? When we tell Perl that we want to `use strict;` Perl will first check the code and one of the things it will do is to look at how variables are declared. In Perl, when we first introduce any variable we can *optionally* describe whether they are available to all parts of a program or not. However, if we turn on `use strict;` it becomes *mandatory* to say whether the variable is a *local* or *global* variable. At this time it is not important to understand the details of this (we will return to it later on), other than that we want our programs to include `use strict` and so we will be making our variables local variables.

Task 3.3

Make sure that `use strict;` is back in your program. Now change line 5 of the program to the following and run your script again (it should now work and should not produce any errors):

```
5. my $pi = 3.14;
```

We are now declaring the \$pi variable using the word `my`. This makes the variable a local variable and we will now be doing this most of the time that we introduce any new variable. It might help to think of the `my` word as reading as 'let'. At this point you are probably thinking that including `use strict;` in your programs is making things more complex. That is true but the benefits of including `use strict;` outweigh the costs associated with it.

Task 3.4

The other point of this programming exercise is to introduce you to the simple fact you can reassign variables to different values or strings. Try declaring a new variable and assign it a value. Add two more lines to change that value and print it out again.

P4. Math

Perl, like most programming languages supports a variety of mathematical operators and functions. Let's experiment with some of these.

Task P4.1

Write the program below, save it as `math.pl`, and then run it. But wait, this time we are going to take a slightly different strategy. The program is getting longer. If you type the whole thing and have a lot of errors, it will become difficult to debug. So instead, write only a few lines, and then save, run, and observe the output. Debug if necessary. Try to check that your program is working every few lines. As you get more experience, you will gain skill and confidence and not need to check as frequently.

```
1. #!/usr/bin/perl
2. # math.pl
3. use strict; use warnings;
4.
5. my $x = 3;
6. my $y = 2;
7. print "$x plus $y is ", $x + $y, "\n";
8. print "$x minus $y is ", $x - $y, "\n";
9. print "$x times $y is ", $x * $y, "\n";
10. print "$x divided by $y is ", $x / $y, "\n";
11. print "$x modulo $y is ", $x % $y, "\n";
12. print "$x to the power of $y is ", $x ** $y, "\n";
```

Task P4.2

In addition to the mathematical operators we've just seen, there are a number of built-in numeric functions: e.g. `abs()`, `int()`, `log()`, `rand()`, `sin()`. Add the following lines to the program, run it, and observe the output.

```
13. print "the absolute value of -$x is ", abs(-$x), "\n";
14. print "the natural log of $x is ", log($x), "\n";
15. print "the square root of $x is ", sqrt($x), "\n";
16. print "the sin of $x is ", sin($x), "\n";
17. print "a random number up to $y is ", rand($y), "\n";
18. print "a random integer up to $x x $y is ", int(rand($x * $y)), "\n";
```

Line 18 could have been written as `int rand $x * $y`. This is another example where you can omit parentheses if you like. But just because you can doesn't mean you should.

Task P4.3

In the examples above, the `print()` function outputs text as well as the actual mathematical operations. This is fairly uncommon in real programming. Generally, we want to make some computation, store that value, and do more computations. To store values, we need to create a new variable that will hold the contents.

```
19. my $z = ($x + $y) / 2;
20. print "$z\n";
```

Task P4.4

In this next exercise, you will build a simple calculator that calculates X to the power of Y. Instead of assigning the variables inside the code, we will let the user input the values without editing the file. In general, this is how programs should work. Once written, they can be used without editing the source code.

```
1. #!/usr/bin/perl
2. # pow.pl
3. use strict; use warnings;
4.
5. my ($x, $y) = @ARGV;
6. print $x ** $y, "\n";
```

Line 5 has an unfamiliar construct. `@ARGV` is list of values from the command line. We will discuss lists and arrays in greater detail later. For now, just accept that the values from the command line will end up being contained in `$x` and `$y`. For example, if you type the line below in the terminal, when the program runs, `$x` will contain 3.14 and `$y` will contain 2.718.

```
$ pow.pl 3.14 2.718
```

Task 4.5

Let's make one more calculator for fun (yes, coding is fun!). This one will compute the factorial of a number. Factorials are usually computed with some kind of a loop (we will talk a lot about loops later). Here is an alternate method that provides a reasonable approximation. Unlike the true factorial, this method can use non-integers. Note that we have written the 6th line of code such that it is split across five lines in our coding editor (lines 7—10 are also indented with tabs). This is a common practice to make code easier to read, though it would still be valid — albeit a little untidy — to write this as a single long line of code (in your editor).

```
1.  #!/usr/bin/perl
2.  # stirling.pl (Stirling's approximation to the factorial)
3.  use strict; use warnings;
4.
5.  my ($n) = (@ARGV);
6.  my $ln_factorial =
7.      (0.5 * log(2 * 3.14159265358979))
8.      + ($n + 0.5) * log($n)
9.      - $n + 1 / (12 * $n)
10.     - 1 / (360 * ($n ** 3));
11. print 2.71828 ** $ln_factorial, "\n";
```

Try it out:

```
$ stirling.pl 5
$ stirling.pl 7.1
```

Operator Precedence

Let's quickly discuss operator precedence. Some operators have higher precedence than others. We're used to seeing this in math where multiplication and division come before addition and subtraction: $3 + 2 * 5 = 13$. If you want to force addition before multiplication, you can do this as $(3 + 2) * 5 = 25$. Perl has a lot of operators in addition to the mathematical operators and there are a lot of precedence rules. Don't bother memorizing them. The universal precedence rule is this: multiplication comes before addition, use parentheses for everything else.

P5. Conditional statements

One of the most basic foundations of programming is the conditional statement. This is simply: if *condition* then do something, otherwise do something else. The *condition* is some kind of true-false statement.

Task P5.1

In the following program, note that equality is tested with two equals signs! One of the most common errors of novice programmers is using a single equals sign.

```
1.  #!/usr/bin/perl
2.  # conditional.pl
3.  use strict; use warnings;
4.
5.  my ($x, $y) = @ARGV;
6.  if ($x == $y) {
7.      print "equal\n";
8.  }
9.  else {
10.     print "not equal\n";
11. }
```

Did you notice how the print statements on lines 7 and 9 are indented? This is no accident! It shows the logical hierarchy of the code. The spacing is achieved by using a tab character. Many code editors will be smart enough to put tabs in for you automatically.

Numerical comparison operators in Perl

We have just seen the `==` operator which tests for numerical equality, here are all the ways of comparing two numbers:

Operator	Meaning	Example
<code>==</code>	equal to	<code>if (\$x == \$y)</code>
<code>!=</code>	not equal to	<code>if (\$x != \$y)</code>
<code>></code>	greater than	<code>if (\$x > \$y)</code>
<code><</code>	less than	<code>if (\$x < \$y)</code>
<code>>=</code>	greater than or equal to	<code>if (\$x >= \$y)</code>
<code><=</code>	less than or equal to	<code>if (\$x <= \$y)</code>
<code><=></code>	comparison	<code>if (\$x <=> \$y)</code>

Indentation and block structure

In general, all the statements that are conditional on some other statement are indented with a tab character. You can have conditional statements inside other conditional statements, in which case you will have multiple levels of indentation. Is this necessary? Yes and no. It is necessary to aid readability, but it is not necessary to get your program to run. Pay attention to the indentation in the example programs and follow them closely. Wikipedia has a good page [describing different indentation styles](#). Feel free to choose one of those, but do not make up your own style! Your #1 job as a programmer is to write programs that can be easily understood by others, and inventing new programming paradigms defeats that goal.

Task P5.2

Modify the program by changing the variables and relational operators. The numeric relational operators are in the accompanying table. Experiment to see if you can figure out what the `<=>` operator does (it is called the spaceship operator).

Task P5.3

Hierarchy is one of the most important concepts in programming. We are used to seeing hierarchical file systems where files are inside of folders which might be inside other folders. Programming uses the same concept. In Perl, hierarchy is shown with tabs and curly brackets. Statements (files) are inside curly brackets (folders) which might be inside other curly brackets (more folders). The following code contains an `if-else` statement, with the `if` part containing a second `if` statement.

```
1.  #!/usr/bin/perl
2.  # nested_conditional.pl
3.  use strict; use warnings;
4.
5.  my ($x, $y) = @ARGV;
6.  if ($x > $y) {
7.      print "$x is greater than $y\n";
8.      if ($x < 5) {
9.          print "$x is greater than $y and less than 5\n";
10.     }
11. } else {
12.     print "$x is not greater than $y\n";
13. }
```

Whitespace

Indentation and white space improve readability. Consider the following legal but confusing code which omits tabs and spaces (and even some semicolons):

```
if($x>$y){print"1\n";if($x<5){print"2\n"}}else{print"3\n"}
```

A program must be readable above all else. A program that works but is unreadable is difficult to improve or maintain.

P5.4

Sometimes you want to test a series of conditions. This next example shows you how to do this by using the `elsif` statement. Note that this example also ends with an `else` statement. This ensures that something will always be printed no matter what value is held in `$x`.

```

1.  #!/usr/bin/perl
2.  # elsif.pl
3.  use strict; use warnings;
4.
5.  my ($x) = @ARGV;
6.
7.  if ($x >= 3) {
8.      print "x is at least as big as 3\n";
9.  }
10. elsif ($x >= 2) {
11.     print "x is at least as big as 2, but less than 3\n";
12. }
13. elsif ($x >= 1) {
14.     print "x is at least as big as 1, but less than 2\n";
15. }
16. else { print "x is less than 1\n"; }
```

Task P5.5

For simple switches such as the above example, it is sometimes useful to break the usual indentation rule. In the example below, note that the obligatory semicolons have been dropped. It turns out that the last line of a block does not need to be terminated with a semicolon precisely for this kind of code beautification. Also note that spaces are added so that the braces line up in columns.

```

1. if      ($x >= 3) {print "x is at least as big as 3\n"}
2. elsif   ($x >= 2) {print "x is at least as big as 2, but less than 3\n"}
3. elsif   ($x >= 1) {print "x is at least as big as 1, but less than 2\n"}
4. else            {print "x is less than 1\n"}
```

Other Conditional Constructs

An alternative to `if ($x != $y)` is `unless ($x == y)`. There are times when ‘unless’ is more expressive than ‘if not’. You cannot use `elsif` or `else` statements with an `unless` statement however.

Perl also lets you do something called *post-fix* notation. This allows you to put the `if` or `unless` at the end of the statement rather than at the beginning. You can't use `elsif` or `else` in this case, but sometimes the post-fix notation just reads much better. Here are two examples:

```
5. print "x is less than y\n" if $x < $y;
6. print "x is less than y\n" unless $x >= $y;
```

Finally, Perl includes something called the *trinary* operator that lets you do very simple if-then-else statements with just a few symbols. Consider the following statement:

```
7. if ($x == $y) {print "yes"}
8. else           {print "no"}
```

This can be written more succinctly as:

```
9. print $x == $y ? "yes\n" : "no\n";
```

The trinary operator is not that commonly used, but you will see it from time to time.

Numeric Precision and Conditionals

Although Perl hides the details, numbers in a computer are generally stored either as *integer* or *floating point* (decimal) numbers. Both *ints* and *floats* have minimum and maximum values, and floats have limited precision. You have probably run into these concepts with your calculator. If you keep squaring a number greater than 1.0 you will eventually run into an *overflow error*. In Perl, this will happen at approximately $1e+308$. Similarly, if you repeatedly square a number less than 1.0, you will eventually reach an underflow error. In Perl, the lowest non-zero value you can get to zero is approximately $1e-308$. Try some extreme values in `pow.pl` or `stirling.pl` to reach underflow and overflow.

Floating point numbers do not have the exact value you may expect. For example, 0.1 is not exactly one-tenth. Perl sometimes hides these details. Try the following code. When you run this, you expect to see 0.3 0.3 0.0, but that's not what happens because adding the imprecise 0.1 three times is not the same as the imprecise 0.3.

```
1.  #!/usr/bin/perl
2.  # float.pl
3.  use strict; use warnings;
4.
5.  my $x = 0.1 + 0.1 + 0.1;
6.  my $y = 0.3;
7.  print $x, "\t", $y, "\t", $x - $y, "\n"; # \t is a tab character
```

Since floating point numbers are approximations, you should not compare them in conditional statements. Never ask if `($x == $y)` if the values are floats because as we have seen, 0.3 is not necessarily equal to 0.3. Instead, ask if their difference is smaller than some acceptable threshold value:

```
8.  my $threshold = 0.001;
9.  if (abs($x - $y) < $threshold) {print "close enough\n"}
```

Project 0: Poisson

The [Poisson distribution](#) is commonly (mis)used as a null model in biology. For example, suppose you are aligning genomic sequence reads back to a genome. Your average depth of coverage is expected to be 10x. In one particular region you observe 5x. In another region you observe 20x. One interpretation is that the 5x region is haploid (assuming the organism is diploid). Similarly, the 20x region could be a duplication. But there is some chance that these fluctuations are entirely random. In this project, we will calculate how likely 5x or 20x coverage is given an expectation of 10x.

The Poisson distribution has two parameters: lambda and k. Lambda is an integer greater than zero. It is both the mean (expected value) and the variance. In our example above, lambda is 10. The second parameter, k, is the number of observations seen. In our example above, this is 5 or 20. This parameter can be any non-negative integer, and zero is a very useful value because we can ask how likely is it if we did not observe something we expected. For example, some areas of a genome may have no reads aligned to them. There may be biological reasons for this, but under a Poisson model, there are also stochastic reasons to expect this from time to time.

One of the reasons to use a Poisson distribution is because it gives you a feel for how important it is to have a large number of counts. Let's say you have observed some phenomenon 9 times. Under a Poisson model, the standard deviation is 3, as this is the square root of the variance. So 9 counts of something can also be thought of as 9 ± 3 . Zero is only 3 standard deviations away from expected. So the difference between 9 counts and 0 counts is not that large. In contrast, the difference between 400 counts and 300 counts is much larger, since 300 is 5 standard deviations away from 400 (20 being the square root of 400). In a gene expression context, suppose you see 9 counts of something in a cancer cell and 0 counts in a normal cell. Is this significant? Is 400 vs. 300 more significant? Statistically speaking, 400 vs. 300 is much more robust than 9 vs. 0. But biologically speaking, 9 vs. 0 may be more important if it turns out to be an accurate representation of the data. However, it could be completely random. When looking at thousands of data points, things that happen only once in a thousand times are supposed to occur! This is why it is important to perform replicate experiments. Enough science, let's get coding.

Goals of your program

- Give your program a reasonable name
- Include informative comments
- Use descriptive variable names

- Use whitespace to show hierarchy and to separate thoughts
- Use @ARGV (see lesson [P4.4](#)) so that you can specify lambda and k on the command line
- Use conditional statements to make sure that lambda and k are not out of bounds
- Use Stirling's approximation to compute factorials (see lesson [P4.5](#))
- You could check to make sure that lambda and k are integers but it is not strictly necessary when using Stirling's approximation
- Report error messages where applicable
- Compute and report the answer (the probability of k, given lambda)

Note that the final item in the list is to compute the answer. Proper programming is more than just getting the right answer. A beautiful program that is not quite correct is better than a correct indecipherable program. The former can be maintained and improved in the future, the latter cannot.

Play around with the program and try a bunch of different values for lambda and k. As a thought experiment, if you set k = 0, what value of lambda gives you confidence that k is significantly different from lambda?

P6. String operators

From algebra, we are used to the idea of using variables in math. But what about strings? Can you add, subtract, multiply, divide, and compare strings? Not exactly, but there are analogous operations. We'll see more of this later. For now, let's just look at some simple operators.

Task P6.1

Create the following program and run it.

```
1.  #!/usr/bin/perl
2.  # strings.pl
3.  use strict; use warnings;
4.
5.  my $s1 = "Hello";
6.  my $s2 = "World\n";
7.  my $s3 = $s1 . " " . $s2;
8.  print $s3;
```

Line 7 introduces the concatenate operator which in Perl is represented by the dot . character. This operator allows you to join two or more strings together and (optionally) store the result in a new variable. In this case we create a new variable \$s3 which stores the result of joining three things together (\$s1, a space character “ ”, and \$s2). Now add the following lines to the script.

```
9.  if      ($s1 eq $s2) {print "same string\n"}
10. elsif   ($s1 gt $s2) {print "$s1 is greater than $s2\n"}
11. elsif   ($s1 lt $s2) {print "$s1 is less than $s2\n"}
```

How are these strings compared? It might make sense to compare them by length, but that's not what is happening. They are compared by their ASCII values. So 'A' is less than 'B' which is less than 'Z'. Similarly 'AB' is less than 'AC' and 'ABCDE' is also less than 'AC'. Oddly, 'a' is greater than 'A'. See the [wikipedia page on ASCII](#) to see the various values. To get the length of a string, you use the `length()` function.

String comparison operators in Perl

Operator	Meaning	Example
eq	equal to	if (\$x eq \$y)
ne	not equal to	if (\$x ne \$y)
gt	greater than	if (\$x gt \$y)
lt	less than	if (\$x lt \$y)
.	concatenation	\$z = \$x . \$y
cmp	comparison	if (\$x cmp \$y)

Task P6.2

Modify the program in [P6.1](#) to experiment with different string comparison operators. Then try comparing a number and a string using both numeric and string comparison operators. Try using the `length()` function.

Task P6.3

If you are interested in ASCII values, try using the `ord()` and `chr()` functions, which convert letters to numbers and vice-versa.

```
12. print ord("A"), "\n";
13. print chr(66), "\n";
```

Matching Operators

One of the most common tasks you may have as a programmer is to find a string within another string. In a biological context, you might want to find a restriction site in some DNA sequence. These kinds of operations are really easy in Perl. We are only going to touch on a few examples here. In a few lessons we will get much more detailed.

Task P6.4

Enter the program below and observe the output. The binding operator `=~` signifies that we are going to do some string manipulation next. The exact form of that manipulation depends on the next few characters. The most common is the match operator `m//`. This is used so commonly that the `m` can be omitted. There are also substitution and transliteration operators. If your script is working then try changing line 6 to make the matching operator match other patterns.

```

1.  #!/usr/bin/perl
2.  # matching.pl
3.  use strict; use warnings;
4.
5.  my $sequence = "AACTAGCGGAATTCCGACCGT";
6.  if ($sequence =~ m/GAATTC/) {print "EcoRI site found\n"}
7.  else {print "no EcoRI site found\n"}

```

Matching operators in Perl

Operator	Meaning	Example
=~ m//	match	if (\$s =~ m/GAATTC/)
=~ //	match	if (\$s =~ /GAATTC/)
!~ //	not match	if (\$s !~ m/GAATTC/)
=~ s///	substitution	\$s =~ s/thing/other/;
=~ tr///	transliteration	\$count = \$s =~ tr/A/A/;

Task P6.5

Add the following lines and observe what happens when you use the substitution operator. This behaves in a similar way to the sed command in Unix.

```

8. $sequence =~ s/GAATTC/gaattc/;
9. print "$sequence\n";

```

Now add the following lines and find out what happens to \$sequence.

```

10. $sequence =~ s/A/adenine/;
11. print "$sequence\n";
12. $sequence =~ s/C//;
13. print "$sequence\n";

```

Line 12 replaces the occurrence of a C character with nothing (//), i.e. it deletes a C character. You should have noticed though that lines 10 and 12 only replaced the *first* occurrence of the matching pattern. What if you wanted to replace all occurrences? To specify a ‘global’ option (i.e. replace all occurrences), we add a letter ‘g’ to the end of the substitution operator:

```
14. $sequence =~ s/c//g; # adding 'g' on the end of substitution operator
```

This is similar to how we use command-line options in Unix, the ‘global’ option modifies the default behavior of the operator.

Task P6.6

Add the following lines to the script and try to work out what happens when you add an ‘i’ to the to matching operator:

```
15. my $protein = "MVGGKKTKICDKVSHEDRISQLPEPLISEILFHLSTKDLWQSVPGLD";
16. print "Protein contains proline\n" if ($protein =~ m/p/i);
```

Task P6.7

In bioinformatics, you will sometimes be given incorrectly formatted data files which might break your script. Therefore we often want to stop a script early on if we detect that the input data is not what we were expecting. Add the following lines to your script and see if you can work out what the `die()` function is doing.

```
17. my $input = "ACNGTARGCCTCACACQ"; # do you know your IUPAC characters?
18. die "non-DNA character in input\n" if ($input =~ m/[efijlopqxz]/i);
19. print "We never get here\n";
```

It is very common to stop scripts by using the ‘die ... if’ syntax. There is no point letting a script continue processing data if the data contains errors. Perl does not know about rules of biology so you will need to remember to add suitable checks to your scripts.

The transliteration operator

The transliteration operator gets its own section as it is a little bit different to the other matching operators. If you worked through [Part 2](#) of the Unix lessons you may remember that there is a `tr` command in Unix. The transliteration operator behaves in the same way as this command. It takes a list of characters and changes each item in the list to a character in a second list, though we often use it with just one thing in each list. It automatically performs this operation on all characters in a string (so no need for a ‘global’ option).

Task P6.8

Make a new script to test the full range of abilities of the transliteration operator. Notice how there are comments at the end of many of the lines (the hash character ‘#’ denotes the start of a comment). You don’t have to type these comments, but adding comments to your scripts is a good habit to get into. You will need to add suitable print statements to this script in order for it to do anything.

```
1.  #!/usr/bin/perl
2.  # transliterate.pl
3.  use strict; use warnings;
4.
5.  my $text = "these are letters: abcdef, and these are numbers, 123456";
6.
7.  $text =~ tr/a/b/;      # changes any occurrence of 'a' to 'b'
8.  $text =~ tr/bs/at/;    # the letter 'b' becomes 'a', and 's' becomes 't'
9.  $text =~ tr/123/321/; # 1 becomes 3, 2 stays as 2, 3 becomes 1
10. $text =~ tr/abc/ABC/; # capitalize the letters a, b, and c
11. $text =~ tr/ABC/X/;   # any 'A', 'B', or 'C' will become an X
12. $text =~ tr/d/DE/;    # incorrect use, only 'd' will be changed to 'D'
```

On Line 5 in this script we define a string and save that to a variable `$text`. Lines 7—12 then perform a series of transliterations on the text.

Task P6.9

If you have many characters to transliterate, you can use the `tr` operator in a slightly different way, which you may (or may not) find easier to understand:

```
13. $text =~ tr [abcdefg] [hgfedcba]; # semicolon is here and not on line 13
```

In this case we use two pairs of square brackets to denote the two range of characters, rather than just three slashes. There are two lines of your code in the text editor, but Perl sees this as just one line. Perl scripts can contain any amount of *whitespace*, and it often helps to split one line of code into two separate lines in your editor. The following line would be treated by Perl as exactly the same as Lines 13–14:

```
15. $text =~ tr[abcdefgh][hgfedcba]; # whitespace removed
```

Task P6.10

The transliteration operator can also be used to count how many changes are made. This can be extremely useful when working with DNA sequences. Add the following lines to your script.

```
16. my $sequence = "AACTAGCGGAATTCCGACCGT";  
17. my $g_count = ($sequence =~ tr/G/G/);  
18. print "The letter G occurs $g_count times in $sequence\n";
```

Line 17 may appear confusing. The transliteration operator is changing the letter G to itself, and it then assigns the result of this operation to a new variable (`$g_count`). So what is happening? Perl performs the code inside the parentheses first and this performs the transliteration. The result of the transliteration is that lots of G->G substitutions are made which leaves `$sequence` unchanged. The transliteration operator counts how many changes are made. Normally it does nothing with this count, but if you ask Perl to assign the output of the transliteration to a variable (as in this example), then it will store the count in that variable.

Task P6.11

Remove the parentheses from line 17. Does the script still work? This is a case where the parentheses are not needed by Perl, but their inclusion might make your code more understandable. If you have any code where you use the assignment operator `=`, Perl always evaluates the right-hand side of the equals sign first.

Task P6.12

Add the following line to your script and see if you can understand how to specify a ‘range’ of characters with the `tr` operator.

```
19. $sequence =~ tr/A-Z/a-z/;
```

If you have added this line but nothing seems to be happening, then maybe you need to add another line of code in order to see the result of what line 19 is doing.

Project 1: DNA composition

At this point, we know enough Perl to write our first useful program. The program will read a sequence and determine its length and composition. Unlike the various tasks that appear in each chapter, we will not provide you the code for this project. You must write it yourself.

Program Name

A descriptive program name helps people understand what it does. But people often choose the name based on some other criteria. Unix program names are almost always short and lower case to minimize typing. Bioinformatics programs tend towards acronyms and abbreviations. Once you come up with a concept for a great program, choosing an appropriate name can sometimes be the hardest part (only half-joking). Feel free to choose whatever name you want for this project, but a name such as project1.pl or dnastats.pl is better than 1337.pl.

Executable

Programs should have executable permission. This will happen automatically if they are on your USB flash drive but you should still know how to use the Unix command `chmod` to add execute permission. Your script should also have a `#!` statement as the first line.

Usage Statement

Programs should have some kind of documentation that tell other people how to use the program. Users should not have to figure it out from the source code, especially if they are not programmers. A simple, but useful form of documentation is the usage statement. This is generally between 1 and 20 lines of text that informs people what the program does and what the arguments are. Usage statements are often displayed if the program is given no arguments. In that case, you want the program to report a little documentation and quit (using the `die` function again). Here are the first few lines of your program.

```
1. #!/usr/bin/perl
2. # dnastats.pl by __
3. use strict; use warnings;
4.
5. die "usage: dnastats.pl <dna sequence>\n" unless @ARGV == 1;
6. my ($seq) = @ARGV;
```

While it is possible to use the `die()` function without printing any output, you should always try to include a helpful statement as to why the program has stopped. It is common to see several `die` statements near the start of a script as this is the point when you should ideally check that all of the script's parameters make sense and that any input files are present (and valid).

On line 5 the `die()` function will be run *unless* the `@ARGV` array contains exactly one item (remember that the `@ARGV` array contains a list of anything you specify on the command-line after the script name). Note that we don't need to calculate the length of the array and store that in a variable, we can just test the length of the array implicitly. You will often use the `die` function in conjunction with the `if` operator, i.e. *if* something is missing, stop the script. Note that line 5 could be replaced with the following if we wanted to make things even more explicit:

```
5. my $number_of_arguments = @ARGV;
6. if($number_of_arguments != 1){
7.     die "usage: dnastats.pl < dna sequence>\n";
8. }
```

Goals of your program

Your program should read a sequence that is specified on the command line and report the following:

- The length of the sequence
- The total number of A, C, G, and T nucleotides
- The fraction of A, C, G, and T nucleotides (i.e. %A, %C etc.)
- The GC fraction

P7. List context

Up till now we have only worked with single variables or values, but we often want to work with lists of things. In Perl, if you have multiple scalar values in parentheses separated by commas, this is known as *list context* (actually it's still list context even if you have one or even zero scalar values in parentheses).

Task P7.1: Create the following short program and run it.

```
1.  #!/usr/bin/perl
2.  # list.pl
3.  use strict; use warnings;
4.
5.  my ($x, $y, $z) = (1, 2, 3);
6.  print "x=$x y=$y z=$z\n";
```

The code in line 5 takes a list of three values (1, 2, 3) and assigns them to a list of three variables (\$x, \$y, \$z). Without using lists, we would have to have three separate lines of code in order to declare and initialize each variable with a value.

Assignments in lists occur simultaneously. Because of this, line 7 below exchanges the values for \$x and \$y.

```
7.  ($x, $y) = ($y, $x);
8.  print "x=$x y=$y\n";
```

Task P7.2

Exchange the value of \$x and \$y without using list context. This is one of those problems that appears difficult at first, but once you see the solution, it will seem so obvious that you can't imagine how you didn't think of it immediately.

P8. Safer programming: use warnings

We've been telling you to always include a `use warnings;` line in your Perl scripts, but we haven't really explained why. Let's see what can happen when we *don't* include it.

Task P8.1

In the last lesson, we discussed assignments in lists. What if the lists are not the same length? Let's find out. Try this program, but this time make sure that you *don't* include the `use warnings` statement.

```
1. #!/usr/bin/perl
2. # undefined.pl
3. use strict;
4.
5. my ($x, $y, $z) = (1, 2, 3, 4, 5);
6. print "x=$x y=$y z=$z\n";
7.
8. my ($a, $b, $c) = (1, 2);
9. print "c=$c\n";
10. print length($c), "\n";
11. print $a + $c, "\n";
```

Line 5 assigns 3 variables with 5 values. The two extra values on the right are simply thrown away.

Line 8 assigns 3 variables from only 2 values. So what happens to `$c`? The output from line 9 suggests that `$c` is some kind of a blank, and the output from line 10 suggests it has no length. But the output from line 11 suggests that `$c` has a value of zero. What is happening in this script is that `$c` has an *undefined value*. It is simultaneously zero and an empty string. Do you find this confusing? It is.

Undefined values are bad. You should never assume the contents of a variable. Variables should always be assigned before they are used. Similarly, lists should be the same length on each side of an assignment, but Perl has no way of checking this. To find undefined values, always include `use warnings` in your program. This will alert you when undefined variables are being used. If you have undefined values, stop immediately and debug. A program that runs with undefined values can be very dangerous.

Task P8.2

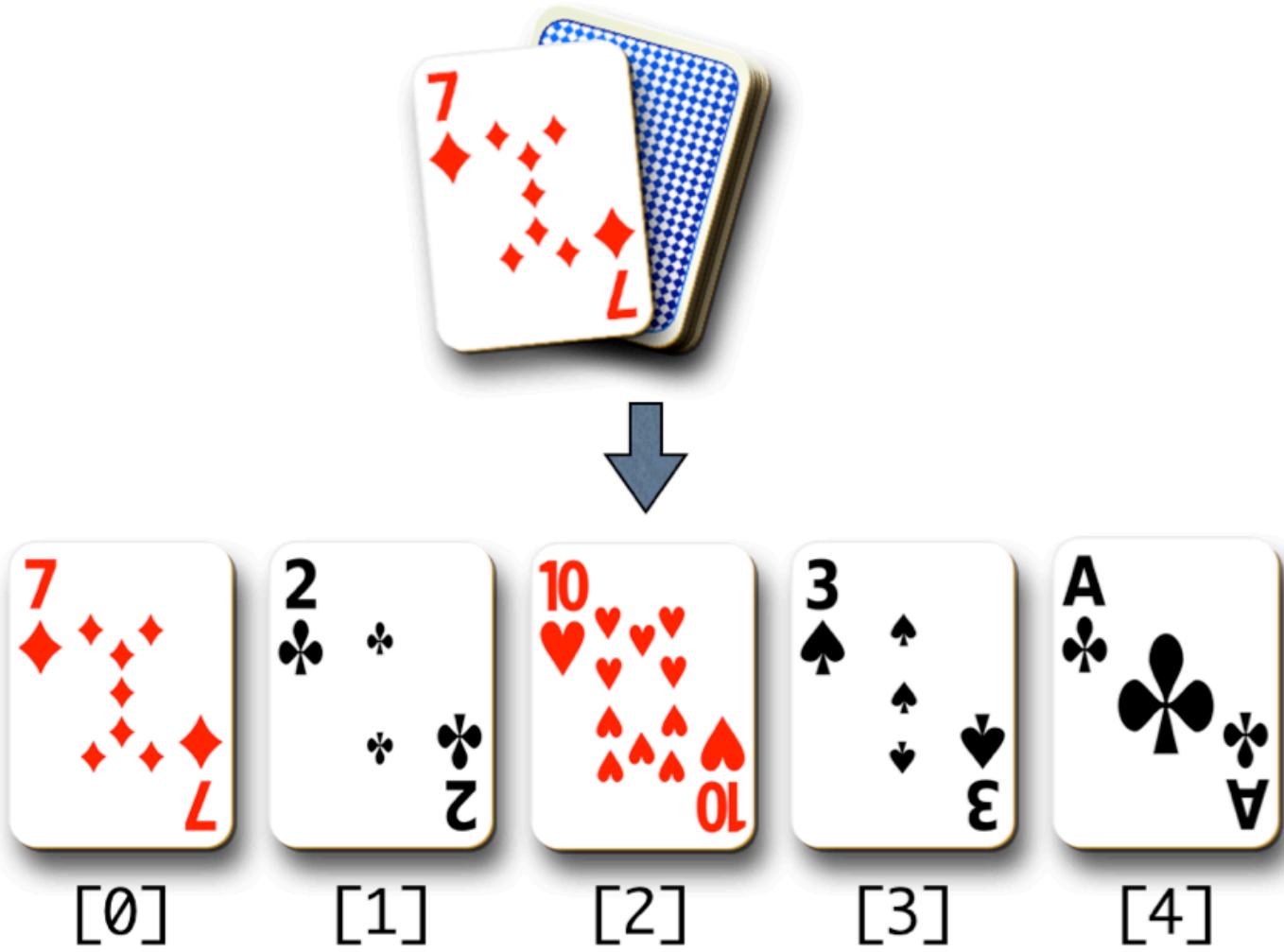
Modify the original program by adding a `use warnings;` line. Run the program and observe what happens. The errors that you should see are a good thing!

P9. Arrays

Lists are useful for declaring and assigning multiple variables at once, but they are transient and if we want to store the details of a list then we have to capture all the values into separate variables. Ideally, there should be a way of referring to all of a list in one go, and there should be a way to access individual items in a list. In Perl (and in most other programming languages) we do this using *arrays*.

An array is a named list. Each array element can be any scalar variable that we have seen so far, e.g. a number, letter, word, sentence etc. In Perl, as in most programming languages, an array is indexed by integers *beginning with zero*. The first element of an array is therefore the zero-th element. This might confuse you but that's just the way it is. Arrays in Perl are named using the @ character. Let's imagine that we have an array called @cards that contains five playing cards (we can imagine that each card in the array would be stored as a text string such as '7D' for 'seven of diamonds').

@cards



Array example

If we wanted to see what the individual elements of the @cards array were, we could access them at array positions 0 through to 4. It's important to note that arrays always have a *start* (the zero-th position), an *end* (in this case, position 4), and a *length* (in this case 5). Arrays can contain just one element in which case the start and the end would be the same. Arrays can also contain no elements whatsoever (more of that later). In biology you might frequently see arrays used to store DNA or protein sequence information. This could either be where each element is a separate DNA/protein sequence, or where each element is one nucleotide/amino acid and the whole array is the sequence.

Task P9.1

Create and run the following program:

```
1.  #!/usr/bin/perl
2.  # array.pl
3.  use strict; use warnings;
4.
5.  my @animals = ('cat', 'dog', 'pig');
6.  print "1st animal in array is: $animals[0]\n";
7.  print "2nd animal in array is: $animals[1]\n";
8.  print "Entire animals array contains: @animals\n";
```

Line 5 assigns the `@animals` array a list of 3 values. Note how we also have to declare arrays with `my` (because we are including the `use strict;` statement).

Lines 6 and 7 show how to access individual elements of an array. You specify a position in the array by putting an integer value between square brackets. The integer value is known as the ‘array index’.

Lines 6 and 7 also shows that you can interpolate individual scalars inside double quotes, i.e. Perl prints out the value stored at the specified array position rather than just printing the text `$animals[0]`.

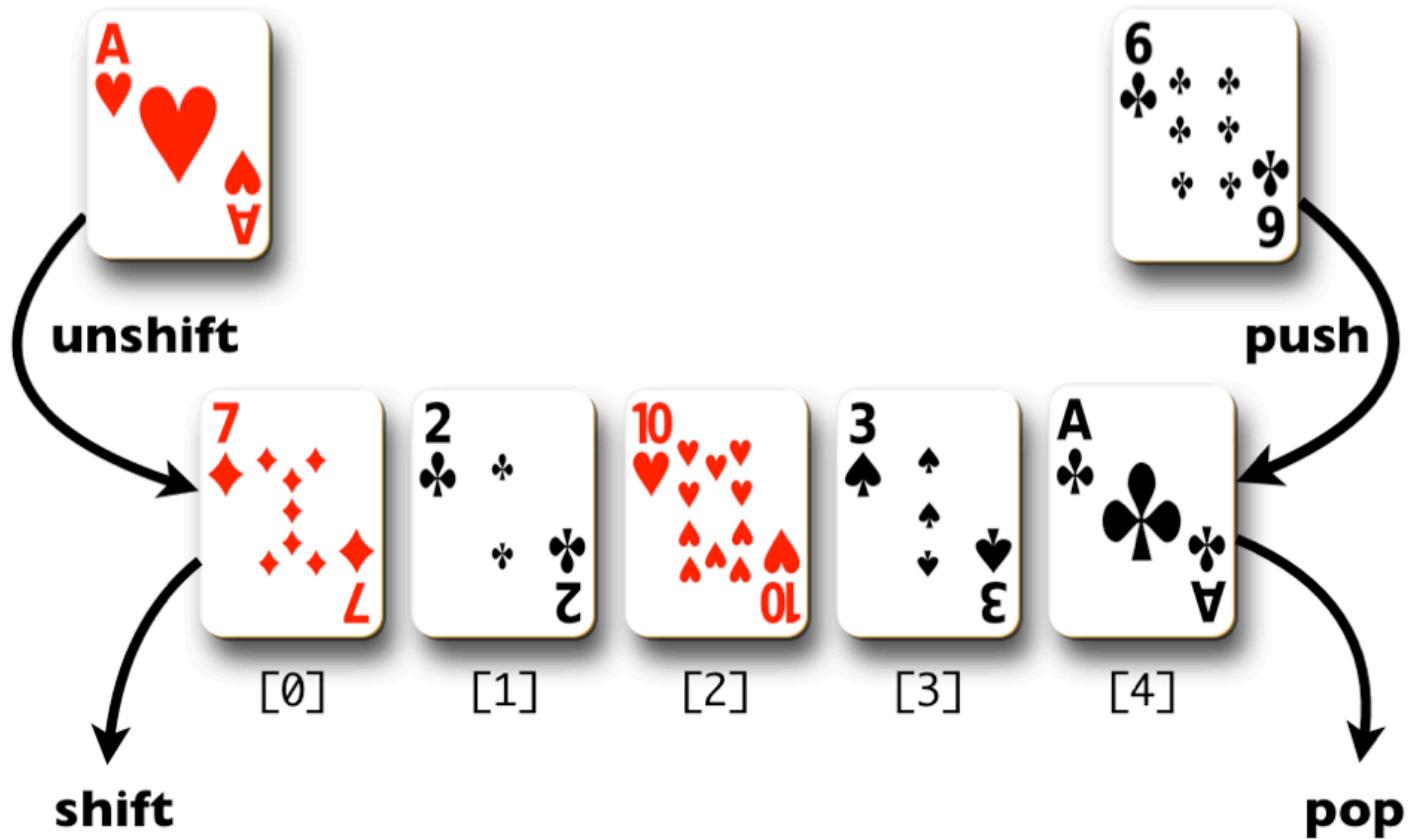
Line 8 shows that if you include an array name between double quotes, then the entire array interpolates and Perl will add spaces between each element in the printed output.

Note that each element of the list is a scalar variable. We write `$animals[0]` never `@animals[0]`. There is no such thing as `@animals[0]` in Perl. The membership of `$animals[0]` in `@animals` is shown by the square brackets. Writing `@animals[0]` is one of the most common errors of new programmers (it’s so common that it will actually be legal in the next version of Perl...). Try modifying the code to include this erroneous syntax and observe the warning message:

```
9.  print "@animals[0]\n"; # bad
```

Making arrays bigger and smaller

Perl arrays are *dynamic*. That is, they grow and shrink automatically as you add/remove data from them. It is very common to modify the contents of arrays, and it is also very common to start off with an array full of things, and then remove one thing at a time. Most of the time we add or remove things to either end of an array and Perl has four dedicated functions to do this:



Functions which modify arrays

Task P9.2

To examine this dynamic behavior, we will first learn to use the `push()` function to add some new data onto the array. The `push` function is used to add one thing to the end of an array. The end of an array is the element with the highest array index position. Add the following lines to your program.

```

10. push @animals, "fox"; # the array is now longer
11. my $length = @animals;
12. print "The array now contains $length elements\n";

```

Line 10 introduces a very useful concept in Perl. If you assign a list to a scalar variable, then the scalar variable becomes the *length of the list*. This is so useful that you will use it a lot in your Perl code. You can think of this in another way. Anywhere in a Perl script where it is possible to specify a numerical value, you can instead specify the name of an array. If that array contains any elements then Perl will calculate the length of the array and use the number of elements.

It is a common mistake to confuse the following two lines of code, can you work out what the difference is?

```
$length = @animals;  
($length) = @animals;
```

The first line of code takes the *length* of the `animals` array and assigns that to the `$length` variable. But when we add parentheses around `$length` we are now making a list, and the second line of code is therefore a *list assignment*. It doesn't look much like a list because there is only one thing in it, but it is still a list. So the second line of code could be read as 'take the `@animals` array and assign all of the elements to a new list called `$length`'. Of course in this case the new list is shorter than the array so it can only receive one item. Have a look again at section [P8.1](#) to see if that helps you understand things.

Task P9.3

Just to make sure you fully understand arrays, let's add a few more lines.

```
13. my ($first, $second) = @animals;  
14. print "First two animals: $first $second\n";  
15. my @animals2 = @animals; # make a copy of @animals  
16. @animals = (); # assign @animals an empty list -> destroys contents  
17. print "Animals array now contains: @animals\n";  
18. print "Animals2 array still contains @animals2\n";
```

Common Array Functions

We already saw `push()` as a way of adding an element to the end (tail) of a list. Naturally, you can add an element to the front (head) of a list, or remove elements instead of adding them. Try modifying your program to use the following set of functions: `pop()`, `shift()`, `unshift()`, and if you're really brave `splice()`. The last function is the hardest one to understand but also the most powerful because it allows you add, remove, or substitute array elements at *any* position in the array, not just at the ends.

Function	Meaning
<code>push(@array, "some value")</code>	add a value to the end of the list
<code>\$popped_value = pop(@array)</code>	remove a value from the end of the list
<code>\$shifted_value = shift (@array)</code>	remove a value from the front of the list
<code>unshift(@array, "some value")</code>	add a value to the front of the list
<code>splice(...)</code>	everything above and more!

Task P9.4

Experiment with the array functions by adding some new lines to array.pl. Rather than just adding a text string to an array, try to see if you can use the `push()` or `unshift()` functions to add *variables* or even other arrays to existing arrays. For the `shift()` and `pop()` functions, try to see what happens if you don't assign the popped or shifted value to a variable. E.g. try to determine the difference between the following two lines of code:

```
my $value = pop(@array);
pop(@array);
```

More About Array Indexes

Let's consider a couple more indexing issues. Add the following lines but before running it, try to guess what will happen.

```
19. @animals = ('cat', 'dog', 'pig'); # needed because @animals was emptied
20. print "Animal at array position 1.2 is $animals[1.2]\n";
21. print "Animal at array position 1.7 is $animals[1.7]\n";
22. print "Animal at array position -1 is $animals[-1]\n";
23. print "array length = ", scalar(@animals), "\n";
```

Floating point values such as 1.2 or 1.7 are rounded down. Using negative numbers for the array index positions have the effect of counting from the tail of the array. The `scalar()` function forces scalar context on its argument. As we know, an array gives its length in scalar context. Recall `$length = @animals`. The `scalar()` function does the same thing without the need to create an extra variable. Something else you can try is to look up an array element using a text string rather than a number. E.g. what happens if you try the following?

```
24. print "Animal at array position 'foobar' is ", $animals["foobar"], "\n";
```

You could substitute “foobar” for any text at all. The first thing that you should notice is that the Perl program should give you a useful warning message:

Argument “foobar” isn’t numeric in array element at...

Strings such as “foobar” have a numeric value of zero and so if you use any text instead of a number when trying to lookup a specific position in an array, you will always get the first (zero-th) element. Hopefully you will never try doing this.

P10. From strings to arrays and back

We saw that if we try printing an array between double quotes, then Perl interpolates the array and prints each element separated by a space. What if we want something other than spaces? In Perl, there's always more than one way to do things, but the best way is with the [join\(\)](#) function, which allows you to create a string from an array and put whatever you want between the elements of the array.

Task P10.1

Let's say we want to create a CSV (comma separated values) format from an array of gene names from the nematode *Caenorhabditis elegans*. Here's how we could do that.

```
1.  #!/usr/bin/perl
2.  # stringarray.pl
3.  use strict; use warnings;
4.
5.  my @gene_names = qw(unc-10 cyc-1 act-1 let-7 dyf-2);
6.  my $joined_string = join(", ", @gene_names);
7.  print "$joined_string\n";
```

Line 5 uses the [qw\(\)](#) function to make an array. `qw()` is short for ‘quote words’. It’s a little shorthand so that we don’t have to keep typing quotation marks.

Line 6 creates a string from an array with `join()`, and specifies that each element of the array should be joined with a comma followed by a space.

The opposite function of `join()` is the [split\(\)](#) function. This divides a string into an array. But we have to tell it where to split. This works sort of like a restriction digest but the restriction site is consumed in the process.

Task P10.2

Add the following lines to your program and run it:

```
8. my $dna = "aaaaGAATTCTttttGAATTCgggggg";  
9. my $EcoRI = "GAATTC";  
10. my @digest = split($EcoRI, $dna);  
11. print "@digest\n";
```

If we want to convert a string into an array and split the string at every possible position, we need to use an empty string ("") in the `split()` function. This is often used to convert DNA/protein sequences stored in variables into arrays:

```
12. my @dna = split("", $dna);  
13. print "@dna\n";
```

P11. Sorting

As in real life, lists are great, but sorted lists are even better. Imagine looking through a telephone book if it wasn't sorted... tedious. Perl has an incredibly flexible sorting function. But it's a little complicated, so you may want to come back and read this part again later.

Task P11.1

Create the following program and run it. How does Perl sort items in a list?

```
1.  #!/usr/bin/perl
2.  # sorting.pl
3.  use strict; use warnings;
4.
5.  my @list = qw( c b a C B A a b c 3 2 1); # an unsorted list
6.  my @sorted_list = sort @list;
7.  print "default: @sorted_list\n";
```

Line 6 calls the `sort()` function. This could have been written with parentheses (e.g. = `sort(@list)`), but this is one of those cases where parentheses are usually omitted. We assign the result of the sort to a new array, but we could have also overwritten the original array, e.g.

```
my @list = sort @list;
```

Looking at the output, it should be clear that Perl sorts by `ASCII` value by default. It is using the `cmp` operator we saw earlier. What if you want to sort numerically? Then you would have to use the numeric comparison operator `<= >`. To specify this, you use an unfamiliar syntax:

```
8.  @sorted_list = sort { $a <= > $b } @list;
9.  print "numeric: @sorted_list\n";
```

In general, sorting routines compare pairs of values. In Perl, these values are held by the magic variables `$a` and `$b`. For this reason, you should not use these variable names in your own programs. Line 8 shows that `$a` and `$b` are compared numerically. The default sort is simply `{$a cmp $b}`.

Because we have the `use warnings;` statement, this code should produce a few warning messages. This is because we are asking to sort values numerically but ‘A’, ‘B’, ‘C’ etc are not numbers. As we saw previously, text has a numeric value of zero. So if you compare text as numbers, it does not sort alphabetically (and Perl warns us of this fact).

If you want to sort in reverse direction, you simply exchange the variables `$a` and `$b`.

```
10. @list = qw (2 34 -1000 1.6 8 121 73.2 0);
11. @sorted_list = sort {$b <=> $a} @list;
12; print "reversed numeric: @sorted_list\n";
```

What if you want to sort both numerically and alphabetically and you want no differentiation between capitals and lowercase? Perl can do this, of course, but the explanation will be left for later.

```
13. @sorted_list = sort { $a <=> $b or uc($a) cmp uc($b)} @list;
14. print "combined: @sorted_list\n";
```

P12. Loops

Loops are one of the most important constructs in programming. Once you have mastered loops, you can do some really useful programming. Loops allow us to do things like count from 1 to 100, or cycle through each element in an array, or even, process every line in an input file. There are three main loops that you will use in programming, the `for` loop, the `foreach` loop, and the `while` loop.

The `for` Loop

The `for` loop generally iterates over integers, usually from zero to some other number. You can think of the integer as a ‘loop counter’ which keeps track of how many times you have been through the loop (just like a lap counter during a car race). The `for` loop has 3 components:

1. initialization — provide some starting value for the loop counter
2. validation — provide a condition for when the loop should end
3. update — how should the loop counter be changed in each loop cycle

If we return to the car race analogy, we can imagine a car having to drive 10 laps around a circular track. At the start of the race the car has not completed any laps so the loop counter would be initialized to zero. The race is clearly over when the counter reaches 10 and each lap of the track updates the counter by 1 lap.

Task P12.1

Create and run the following program.

```
1. #!/usr/bin/perl
2. # loop.pl
3. use strict; use warnings;
4.
5. for (my $i = 0; $i < 10; $i = $i + 1) {
6.     print "$i\n";
7. }
```

The syntax for a `for` loop requires the three loop components to be placed in parentheses and separated with semi-colons. Curly braces are then used to write the code that will be executed during each iteration of the loop. This code is usually indented in the same way that we indent blocks of code following if statements. In this loop we first declare a new variable `my $i` to act as our loop counter. It is a convention in programming to use `$i` as a loop variable name because of the use of `i` as a counter in mathematical notation, e.g.

$$\sum_{i=0}^n x_i$$

mathematical summation

You could name your loop counter anything that you wanted to, but we suggest that for now you just use `$i`. Let's see what the three components of our loop are doing:

- `$i = 0` — performs initialization, i.e. start our loop with `$i` equal to zero
- `$i < 10` — performs validation, i.e. keep the loop going as long as `$i` is less than 10
- `$i = $i + 1` — performs the update, `$i` is incremented by one during each loop iteration

It is very common in Perl that you want to take a number and just add 1 to it. In fact, it is so common that Perl has its own operator to do it, the increment operator. Here's how you could increment the value of `$i` by 1:

```
$i++
```

This is more succinct and is the common way to increment a variable by one. Not surprisingly, you can also decrement a variable by one:

```
$i--
```

Note that the 'update' component of the loop should describe a way of increasing (or decreasing) the value of `$i` otherwise the loop would never end.

Task P12.2

Try looping backwards and skipping:

```
8. for (my $i = 50; $i >= 45; $i--) {print "$i\n"}
9. for (my $i = 0; $i < 100; $i += 10) {print "$i\n"}
```

Since the blocks of code following these `for` loops are only one line long, they do not need a semi-colon at the end. We saw this earlier when making conditional statements tidy.

Line 9 uses the `+=` operator. This is a useful shortcut and in this case the result is exactly the same as if we had typed `$i = $i + 10`. Similar operators exist for subtraction `-=`, multiplication `*=` etc. Note how the loop in line 9 is counting in tens and not incrementing by one at a time.

Task P12.3

Let's do something a little bit useful with a loop. This program computes the sum of integers from 1 to `n`, where `n` is some number on the command line. Of course you could compute this as $(n+1) * n / 2$, but what is the point of having a computer if not to do brute force computations?

```
1. #!/usr/bin/perl
2. # sumint.pl
3. use strict; use warnings;
4.
5. die "usage: sumint.pl <limit>\n" unless @ARGV == 1;
6. my ($limit) = @ARGV;
7. my $sum = 0;
8. for (my $i = 1; $i <= $limit; $i++) {$sum += $i}
9. print "$sum\n";
```

Line 5 contains a usage statement. We saw this earlier in [Project 1](#) and in this script it is just adding a check to ensure that we specify one (and only one) command-line argument when we run the script. Line 6 assigns the specified command-line argument to `$limit`. Line 7 creates a variable to hold the sum. Line 8 uses a loop to add the latest value of `$i` to the `$sum` variable.

Task P12.4

Write a program, `factorial.pl`, that computes the factorial of a number. Structurally, it will be very similar to `sumint.pl`, but of course you will be multiplying values instead of adding.

Task P12.5

One of the most common operations you will do as a programmer is to loop over arrays. Let's do that now. To make it interesting, we will loop over two arrays simultaneously:

```
1. #!/usr/bin/perl
2. # loops.pl
3. use strict; use warnings;
4.
5. my @animals = qw(cat dog cow);
6. my @sounds = qw(Meow Woof Moo);
7. for (my $i = 0; $i < @animals; $i++) {
8.     print "$i) $animals[$i] $sounds[$i]\n";
9. }
```

The `for` loop starts at 0, which is where all arrays start, and continues as long as the loop variable `$i` is less than the length of the array (which is found from the scalar context of an array).

The `foreach` Loop

The `foreach` loop allows you to iterate through the contents of an array without a numeric index. Instead, a temporary variable is set to the contents of each element. Add the following code to your program.

```
10. foreach my $animal (@animals) {
11.     print "$animal\n";
12. }
```

In this loop, `$animal` is the temporary variable. It changes from cat to dog to cow with each iteration of the loop. It is common to name the temporary variable as a singular form of the array name. E.g.

```
foreach my $protein (@proteins) { ... }
foreach my $car (@cars) { ... }
foreach my $knight_who_say_Ni (@knights_who_say_Ni) { ... }
```

You can also use the `foreach` loop in a numeric manner. If you are a lazy typist (potentially an admirable quality if you are concerned about RSI), you can even use `for` rather than `foreach`. Line 13 shows how to create to create a numeric list with the `range operator` ... This can be used to loop over letters or numbers:

```
13. for my $i (0..5) {print "$i\n"}
```

The `while` Loop

The `while` loop continues to iterate as long as some condition is met, where the condition is some notion of True or False. The ‘condition’ part of a `while` loop can be as simple or as complex as you want it to be. Here is an example of a very simple `while` loop which keeps doubling a number until some limit is reached:

```
14. my $x = 1;
15. while($x < 1000){
16.     print "$x\n";
17.     $x += $x;
18. }
```

In this example the code will continue to loop while the value of `$x` is less than 1000, and `$x` is doubled for each iteration of the loop. It is important that the test condition will be testing something that is going to change. But Perl will allow you to write code which contains a pointless test condition.

Task P12.6

Add these lines to your program and run it:

```
19. while (0) {
20.     print "this statement is never executed because 0 is false\n";
21. }
22. while (1) {
23.     print "this statement loops forever\n";
24. }
```

The first while loop will never print anything at all because a zero value is always treated as false by Perl. So the loop will run only while the value of zero evaluates to true which is never going to happen.

The second loop will start but never end because the test condition ('while 1 is true') is always true. In fact, anything which isn't a zero or the null string ("") will always evaluate as true. To stop this program, press Control+c in your terminal. This sends the Unix 'interrupt' signal to the program (you might want to commit that trick to memory).

Let's try looping through an array with a while loop. Replace lines 10–12 with these.

```
10. while (@animals) {  
11.     my $animal = shift @animals;  
12.     print "$animal\n";  
13. }
```

In each iteration through the loop, the array @animals is shortened by removing one item from the front of the list (using the shift function). The loop ends when the length of the array is 0 (empty). There are times when this kind of array-deletion construct is useful, but most of the time you will be looping through arrays with for or foreach.

The do Loop

The do loop is a variation of the while loop. Unlike the while loop, it always executes at least once. do loops are not so common.

```
26. do {  
27.     print "hello\n";  
28 } while (0);
```

Congratulations! You have now learned about variables, numbers, math, strings, conditionals, arrays, and loops. Even though there is still a lot to learn, you have come a long way. You can now write some very useful programs.

Loop Control

There are times when you will want a little more control in your loops. The `next` keyword immediately restarts the loop at the top and advances the loop variable. The `redo` keyword restarts the loop also, but does not advance the loop variable. The `last` keyword terminates the entire loop.

P12.7

Here is a program that illustrates `redo` and `last`. It computes the prime numbers between 100 and 200.

```
1.  #!/usr/bin/perl
2.  # primes.pl
3.  use strict; use warnings;
4.
5.  my $n = 0;
6.  while (1) {
7.      $n++;
8.      redo if $n < 100;
9.      last if $n > 200; # breaks out of while loop
10.
11.     my $prime = 1; # assumed true
12.     for (my $i = 2; $i < $n; $i++) {
13.         if ($n % $i == 0) {
14.             $prime = 0; # now known to be false
15.             last; # breaks out of for loop
16.         }
17.     }
18.
19.     print "$n\n" if $prime;
20. }
```

Line 8 uses the `redo` keyword. This short-circuits the `while` loop as long as `$n` is less than 100. You could have used `next` here also because there is no loop variable.

Line 9 uses the `last` function to terminate the `while` loop, effectively ending the program, if `$n` is greater than 200.

Lines 11–17 determine if a number is prime. This method starts off assuming \$n is prime. It then checks all the numbers between 2 and \$n – 1 to determine if \$i is a factor of \$n. If \$i is a factor of \$n (line 11) then there is no point in calculating any further because \$n is not prime. So \$prime is set to false (line 14) and the for loop is terminated (line 15).

When to use each type of loop?

There will be situations where you can use different types of loop structure to achieve exactly the same goal for a program. Conversely there are times when only one type of loop will do. It might not always be clear to you how to make the correct choice, but with practice it becomes more obvious. Feel free to experiment with different loop structures to see what works and what doesn't.

Project 2: Descriptive statistics

In this project, you will write a program that computes typical descriptive statistics for a set of numbers. Here are the first few lines. Your program should compute the count, sum, minimum, maximum, median, mean, variance, and standard deviation. It should report these in some pleasing format. Of course, it should have a [usage statement](#) like all good programs. Here are the first few lines.

```
1. #!/usr/bin/perl
2. # stats.pl by ___
3. use strict; use warnings;
4.
5. die "usage: stats.pl <number1> <number2> <etc>\n" unless @ARGV > 1;
```

Count, Sum, and Mean

We already know how to do these.

Min, Max, and Median

The median value is at the middle of the sorted list of values. If the list has an even number of elements, then the median is the average of the two at the middle. The minimum and maximum are easily found from the sorted array.

Variance

Variance is the average squared difference from the mean. So compute the mean first and then go back through the values, find the difference from the mean, square it, and add it all up. In the end, you divide by n or by $n - 1$ depending on if you are computing the population or sample variance.

Standard Deviation

Simply the [sqrt\(\)](#) of the variance.

Project 3: Sequence shuffler

In this project, you will create a program that randomly shuffles a DNA sequence (or any text really). Shuffling is a really useful way to provide a null model. For example, suppose you do a sequence alignment and get a score of 30. Is this a good score? How often does a score of 30 happen by chance? To determine this, you could randomly shuffle your sequence and perform the search again (and again, and again...).

There are a variety of ways to shuffle a sequence. One way is to repeatedly exchange randomly selected pairs of letters. Another way is to remove a random letter from one sequence to build up another sequence. Random numbers can be generated with the `rand()` function which we first saw back in lesson [P4.2](#).

As usual, your program should have a [usage statement](#).

Strategy 1

1. Turn the DNA string into an array with `split()`
2. Use a `for` loop to perform the following procedure many times
 1. Select a random position A with `rand()`
 2. Select a random position B with `rand()`
 3. Exchange the letters at indices A and B
3. Print the now shuffled array

Strategy 2

1. Create a new, empty array to hold the shuffled sequence T
2. Turn the DNA string into an array with `split()`
3. Use a `while` loop for as long as the original array exists
 1. Select a random position A in the original array with `rand()`
 2. Remove the letter at index A from the original array with `splice()`
 3. Add the letter to the shuffled array with `push()`
4. Print the new shuffled array

P13. File I/O

Our programs so far have taken arguments on the command line. But that is not a very common way to receive data. People generally hand you a *file* (or more commonly lots of files). Fortunately, reading files in Perl is incredibly simple.

Task P13.1

Create the program below and when you run it, specify the name of a text file on the command line after the program name. E.g.

```
$ cd  
$ linecount.pl Data/Misc/oligos.txt
```

Remember, you can be in *any* directory on your computer when you run this script, but you will always need to specify where the oligos.txt file is *in relation to where you are* (using a relative or absolute path). After you run this script with the oligos.txt file, try running it against several files at once (by putting multiple file names on the command line).

```
1. #!/usr/bin/perl  
2. # linecount.pl  
3. use strict; use warnings;  
4.  
5. my $lines = 0;  
6. my $letters = 0;  
7. while (<>) {  
8.     $lines++;  
9.     $letters += length($_);  
10. }  
11. print "$lines\t$letters\n"; # \t is a tab character
```

Line 7 contains something you haven't seen before. This is the <> file operator. By default, this reads *one line at a time* from the file specified on the command line. If there are multiple files on the command line, it will read them all in succession. It even reads from STDIN (standard input) if you include it in a pipe. True Perl magic!

The default variable `$_`

Line 9 is our first introduction to the default variable `$_`. Perl automatically assigns data to this variable in some settings. In this example, `$_` will contain each line of the file. Although you don't see it, Perl is actually performing the following operation.

```
5. while ($_ = <>) {
```

If you find this confusing, you can use a named variable of your choice instead of using `$_`:

```
5. while (my $line = <>) {
```

But you should get used to using `$_` because it is so common among Perl programs. Perl can also retrieve `$_` by default in many functions. For example, if you try using the `print()` function without any arguments, it will report the contents of `$_`. The following one-line program simply prints out the the contents of any file specified on the command-line:

```
while (<>) {print}
```

You can use `$_` in loops too, but we prefer not to. Here is another one-liner in which `$_` is used in place of a named loop variable:

```
for (0..5) {print}
```

Confusing? Yes, a little. But you do get used to it. For now, feel free to name all your variables. By the way, in addition to `$_`, there are a large number of other *special variables* with equally strange symbols.

The `open()` Function

There are times when you have several files and you don't want to read them all one after the other. For example, one might be a FASTA file and the other GFF. You wouldn't want to process both files with the same code. To open and read a single file, you use the `open()` function. This will open a file for reading or writing, but not both. Let's see how we can use it.

Task P13.2

Create the following program. This will read the contents of a file that you specify on the command line and then create a second file with slightly altered contents.

```

1.  #!/usr/bin/perl
2.  # filemunge.pl
3.  use strict; use warnings;
4.
5.  open(IN, "<$ARGV[0]") or die "error reading $ARGV[0] for reading";
6.  open(OUT, ">$ARGV[0].munge") or die "error creating $ARGV[0].munge";
7.  while (<IN>) {
8.      chomp;
9.      my $rev = reverse $_;
10.     print OUT "$rev\n";
11. }
12. close IN;
13. close OUT;

```

Lines 5 and 6 contain `open()` statements for reading and writing. `IN` and `OUT` are called [filehandles](#). These are special variables used only for file operations. The second argument determines if the `open()` statement is for reading or writing. `<` is for reading and `>` is for writing. This should look familiar from your Unix lessons. If you do not include `<` or `>`, then the file is opened for reading. Both of the `open()` statements include an additional `or` clause in case of failure. We will talk more about this later. Line 7 should look a little familiar. Instead of using `<>` by itself, there is a named filehandle inside the brackets. Only the file associated with `IN` will be read. The file in question does not need to be stored in a variable, but usually is (e.g. you could open a file called ‘sesame.txt’ with `open(IN, "sesame.txt")`).

Line 8 introduces the [chomp\(\)](#) function. This removes a `\n` character from the end of a line if present. It is quite common to chomp your `$_`.

Line 9 uses the [reverse\(\)](#) function to reverse the contents of `$_`. We haven’t seen the `reverse()` function before. It reverses both strings and arrays.

Lines 12 and 13 close the two filehandles. You should always get into the habit of making sure that every `open()` function has a matching `close()` function. It is possible that bad things will happen if you don’t close a filehandle. You should also try to close a filehandle at the first opportunity when it is safe to do so, i.e. as soon as you are finished with reading from, or writing to, a file.

Naming filehandles - part 1

Filehandles are typically given upper-case names. You can use lower-case names and your script will probably still work but Perl will also print out a warning. If you only ever read from one input file and write to one output file then IN and OUT are typical filehandle names, though feel free to name them whatever you feel is most suitable (INPUT, DATA etc.). If you need to read from multiple files then it might be a good idea to use filehandle names that describe the type of data, e.g. GFF or FASTA.

Naming filehandles - part 2

The way in which we have just explained how to name and create filehandles is an older style which has become less common in recent years due to some changes that Perl made (yes, even programming languages have styles that come and go!). Perl now allows you to use a regular scalar variable as a filehandle. This may or may not be more intuitive to you. Here is the code from task [P13.2](#) rewritten to use the newer style of filehandle:

```
1.  #!/usr/bin/perl
2.  # filemunge.pl
3.  use strict; use warnings;
4.
5.  open(my $in, "<$ARGV[0]") or die "error reading $ARGV[0] for reading";
6.  open(my $out, ">$ARGV[0].munge") or die "error creating $ARGV[0].munge";
7.  while (<$in>) {
8.      chomp;
9.      my $rev = reverse @_;
10.     print $out "$rev\n";
11. }
12. close $in;
13. close $out;
```

As you can see, the only differences are that IN and OUT have been replaced by the scalar variables \$in and \$out. If you work with other people's Perl code, you might see examples of the older style filehandle so it is good to know about them both, though we prefer the newer style.

Different ways of creating filehandles

As well as having two different ways of naming filehandles, Perl also allows you to create them using a couple of different methods. So far we have seen the *two-argument* method, where the two arguments in question were:

1. the filehandle name (e.g. IN or my \$in)
2. the file name and the read/write status (using < or >)

If you are open a file to read from it, then the < part (technically known as the *file mode*) is not actually necessary. This means that the following are considered identical by Perl:

```
open(my $in, "<input.txt"); # with file mode (<)
open(my $in, "input.txt"); # without file mode
```

Some people prefer it when things are more explicit (i.e. when you always have to specify the read/write permission) and so Perl also allows you to use a *three-argument* mode for creating filehandles. In this syntax, you *must* specify the read/write permission as the second, of three, arguments. E.g.

```
open(my $in, "<", "input.txt"); # read from file
open(my $out, ">", "output.txt"); # write to file
```

This is the syntax we suggest you use as it more obvious when you are reading from, and when you are writing to, a file. It can be bad to accidentally write to a file when you were expecting to read from it, as you will overwrite the file!

P14. Hashes

A *hash* is also called a dictionary or associative array. It is very similar to the kind of array we saw earlier except that instead of indexing the array with integers, the array is indexed with text. The dictionary analogy is fitting. A word is an index to its definition. A hash can be created in list context, just like an array. But since we need to provide the text index, it is necessary to provide *key, value* pairs.

Task P14.1

Create the following program. We have not seen the % sign in front of a variable before. This is symbol for a hash variable. If we are including the `use strict;` statement (which we *always* should be doing) then we will also need to declare any hashes with `my`.

```
1.  #!/usr/bin/perl
2.  # hash.pl
3.  use strict; use warnings;
4.
5.  my %genetic_code = ('ATG', 'Met', 'AAA', 'Lys', 'CCA', 'Pro');
6.  print "$genetic_code{'ATG'}\n";
```

Notice that when you want to access a value from a hash, you use curly brackets { and } rather than square brackets [and]. Curly brackets lets Perl know you are accessing a hash rather than an array. You could therefore have variables named \$A, @A, and %A, and they would all be different variables. Note that using the same name for different things in this way, would be considered bad programming style. \$A is scalar. \$A[0] is the first element of the @A array. \$A{ 'cat' } is the value for the 'cat' key of the %A hash.

When declaring hashes, there is an alternative syntax that makes the assignments more obvious. Here, we replace the comma between the key and the value with a kind of arrow =>. This reads as 'gets'. Or alternatively 'says'. So 'cat' => 'meow' reads as "cat says meow". Let's change line 5 to use this alternative syntax:

```
5.  %genetic_code = ('ATG' => 'Met', 'AAA' => 'Lys', 'CCA' => 'Pro');
```

This syntax looks even more logical when the hash assignment is split across multiple lines:

```
5. %genetic_code = (
6.     'ATG' => 'Met',
7.     'AAA' => 'Lys',
8.     'CCA' => 'Pro',
9. );
10. print "$genetic_code{'ATG'}\n";
```

The last comma in line 8 is unnecessary, but it does no harm, and we like tidy, consistent code. It turns out that when using the `=>` syntax, Perl knows that you are assigning a hash, so the quotes around the keys are actually unnecessary:

```
5. %genetic_code = (
6.     ATG => 'Met', # single quotes now removed from keys
7.     AAA => 'Lys',
8.     CCA => 'Pro',
9. );
10. print "$genetic_code{'ATG'}\n";
```

The quotes on the values are absolutely required in this example because the values are strings. You would not need them if the values were numbers.

Keys and Values

It's a simple matter to iterate through arrays because they have numeric indices from 0 to one less than the array size. For hashes, we must iterate over the keys, and for that, we need the various strings. Not surprisingly, this is performed with the `keys()` function.

Task P14.2

Add the following code to your program to report the keys and corresponding values from your hash. It is sometimes common to use the variable name `$key` in a `foreach` loop, although in this example `$codon` may also be a suitable choice.

```
11. foreach my $key (keys %genetic_code) {
12.     print "$key $genetic_code{$key}\n";
13. }
```

The `keys()` function returns an array of keys. Each key in turn is then assigned to a temporary variable as part of a `foreach` loop (see section P12 for a refresher on loops). A function that is related to the `keys()` is `values()`. This function returns an array of values. Add the following lines to your program to observe this more explicitly:

```
14. my @keys = keys(%genetic_code);
15. my @vals = values(%genetic_code);
16. print "keys: @keys\n";
17. print "values: @vals\n";
```

Hashes store key-value pairs in a semi-random order (it's not random, but you have no control over it). So you will often want to use the `sort()` function to sort the keys that you extract from the hash. Replace line 11 with the following.

```
11. foreach my $key (sort keys %genetic_code) {
```

Adding, Removing, and Testing

Recall that for arrays, you generally either `push()` or `unshift()` to add new values to an array. You can also assign a value at an arbitrary index such as `$array[999] = 5`. Adding pairs to a hash is similar to assigning an arbitrary index. If you assume the key exists, Perl will create it for you. But watch out, if you use a key that previously existed, the value will be overwritten.

Task P14.3

Modify your program to include the following statements:

```
18. $genetic_code{CCG} = 'Pro';
19. $genetic_code{AAA} = 'Lysine';
```

Line 18 adds a new key CCG to the hash. Note that the value of this key Pro already exists as the value to *another* hash key CCA (this is not a problem). Line 19 reassigns the value that the AAA key points to (Lysine instead of Lys). Sometimes you may want to ask if a particular key already exists in a hash, for example, before overwriting something. To do this, you use the `exists()` function:

```
20. if (exists $genetic_code{AAA}) {print "AAA codon has a value\n"}  
21. else  
                  {print "No value set for AAA codon\n"}
```

To remove both a key, and its associated value, from a hash, you need to use the `delete()` function:

```
22. delete $genetic_code{AAA};
```

Use suitable `print` statements to check that you correctly added and removed new key-value pairs in your hash.

Summary of hash-related functions

Function	Meaning
<code>keys %hash</code>	returns an array of keys
<code>values %hash</code>	returns an array of values
<code>exists \$hash{key}</code>	returns true if the key exists
<code>delete \$hash{key}</code>	removes the key and value from the hash

Hash names

If you work with a lot of hashes, it can sometimes help to make the hash name explain something about the data it contains. Hashes typically link pairs of connected data, e.g. name of sequence, and GC% content of that sequence; name of a politician, and the number of votes that they received. Based on these examples, which of the following hash names do you think best describe the data that they contain?

```
%seq;  
%sequences;  
%sequence_details;  
%sequence2gc;  
%sequence_to_gc;  
  
%vote;  
%names;  
%name2votes;  
%name_to_votes;
```

Bad names for hashes include:

```
%hash;  
%data;  
%stuff;  
%things;  
%Perl_is_awesome; # but bonus points for enthusiasm!
```

P15. Organizing with hashes

Task P15.1

Examine the Data/Misc/oligos.txt file. This is a file containing the names and sequences of some oligos separated by tabs. Suppose you want to calculate the melting temperature (Tm) of each oligo and then print out the oligos ordered by their Tm. We can do this by using two hashes, one will store the sequences and the other will store the Tms. Both hashes will be indexed by the oligo name (i.e we will use the same keys for both hashes).

Note the use of comments and whitespace in this script. As we discussed previously, these help the readability of the program. The “header” is lines 1–3. Line 5 is by itself because it is functionally distinct. Line 7 declares the two hashes that we will use. Lines 9–24 are the main body of the program. Whitespace and comments further refine these sections to their purpose. Lines 26–29 are for output. Try to follow a similar logical structure in your own programs. Line 27 may be incomprehensible at first. If you don’t get it, don’t sweat it.

```

1.  #!/usr/bin/perl
2.  # oligo.pl by ___
3.  use strict; use warnings;
4.
5.  die "usage: oligo.pl <file of oligos>\n" unless @ARGV == 1;
6.
7.  my (%sequences, %tm); # declare two hashes
8.
9.  # process file line-by-line
10. while (<>) {
11.     chomp;
12.
13.     # store sequence
14.     my ($name, $seq) = split("\t", $_);
15.     $sequences{$name} = $seq; # first hash assignment
16.
17.     # calculate and store Tm
18.     my $A = $seq =~ tr/A/A/;
19.     my $C = $seq =~ tr/C/C/;
20.     my $G = $seq =~ tr/G/G/;
21.     my $T = $seq =~ tr/T/T/;
22.     my $tm = 2 * ($A + $T) + 4 * ($C + $G); # simple Tm formula
23.     $tm{$name} = $tm; # second hash assignment
24. }
25.
26. # report oligos sorted by Tm
27. foreach my $name (sort {$tm{$a} <=> $tm{$b}} keys %tm) {
28.     print "$name\t$tm{$name}\t$sequences{$name}\n"; # $name as used as key
in 2 hashes
29. }
```

P16. Counting codons with substr()

Task P16.1

The `substr()` function is useful for extracting a sub-string from a string. In bioinformatics we often want to extract a part of an amino acid or nucleotide sequence. Here is a little program that shows how `substr()` works. Note the 3 arguments that the `substr()` function requires:

1. the string that you want to extract from,
2. an offset (starting from zero)
3. the length for how many characters to extract.

This program should just print MRVLK ... TVLSAPAKIT:

```
1. #!/usr/bin/perl
2. # substr.pl
3. use strict; use warnings;
4.
5. my $seq = "MRVLKFGGTSVANAERFLRVADILESNDARQGVATVLSAPAKIT";
6. my $first5 = substr($seq, 0, 5);
7. my $last10 = substr($seq, length($seq) - 10, 10);
8. print "$first5 ... $last10\n";
```

Task P16.2

Now let's do something useful and determine the codon usage for a sequence that is provided on the command line.

```

1.  #!/usr/bin/perl
2.  # codon_usage.pl by ___
3.  use strict; use warnings;
4.
5.  die "usage: codon_usage.pl <sequence>\n" unless @ARGV == 1;
6.  my ($seq) = @ARGV;
7.
8.  my %count = ();  # individual codons
9.  my $total = 0;   # total codons
10.
11. # extract each codon from the sequence and count it
12. for (my $i = 0; $i < length($seq); $i += 3) {
13.     my $codon = substr($seq, $i, 3);
14.     if (exists $count{$codon}) { $count{$codon}++ }
15.     else                      { $count{$codon} = 1 }
16.     $total++;
17. }
18.
19. # report codon usage of this sequence
20. foreach my $codon (sort keys %count) {
21.     my $frequency = $count{$codon}/$total;
22.     printf "%s\t%d\t%.4f\n", $codon, $count{$codon}, $frequency;
23. }
```

Note that on line 8 we use a slightly different way of introducing a hash. The following lines of code are similar:

```

my %count;
my %count = ();
```

The first example declares a new hash, and the second example additionally initializes the hash which means it will empty the hash of any data (if any existed).

You may have noticed that line 20 adds a second `my $codon` = statement to the script. It is important to realize that the `$codon` within this second `foreach` loop is completely different to the `$codon` that exists in the previous `for` loop (lines 12–17). If this seems confusing, then you will have to wait a little longer before we give the full explanation for this. If it bothers you, then feel free to rename the second `$codon` variable to something else.

Line 22 introduces the `printf()` function to format the output. The `printf()` function has a somewhat arcane syntax handed down from the C programming language, and uses the following special characters:

- `%s` means string
- `%d` means digit (integer)
- `%f` means floating point

When using this function, you first specify how you want to format the list of things that you want to print (this part is between quotation characters). You then specify a list of variables that contain the data (or you could put the actual data here, rather than use variables). So on line 22 we print three things (separated by tab characters) as follows:

- `%s` will print the string contained in `$codon`
- `%d` will print the digit contained in `$count{$codon}`
- `%.4f` will print the floating point number in `$frequency` to 4 decimal places.

Project 4: The name game

Have you ever wondered which protein sequences might contain your name? If you have a short name, and your name does not contain the letters B, J, O, U, X, or Z, then it may occur in many different protein sequences. If not, you may need to alter your name a little or try searching for questions (e.g. ‘WHATISTHEANSWER’).

Goal

Write a program that reports the names of sequences matching your name (or some other word). The [usage statement](#) for your program should look something like this:

```
usage: name_search.pl <protein file> <name>
```

Details

The file of proteins you will use is called `At_proteins.fasta`. You will find this in the `Data/Arabidopsis` directory. This is a typical FASTA formatted file and contains entries with sequences that span multiple lines. Ideally, a good Perl script would be able to look for patterns across multiple lines, but to make things a little easier we will first modify this file in order to make the sequence for each entry only span one line. Later on, you might want to find out about Perl modules like [FAlite.pm](#) that can help you more easily read FASTA files, but for now we will [revisit](#) some of our Unix skills and make a new version of this file:

```
cat At_proteins.fasta | sed 's/^(>.*\n)/!\1!/' | tr '\n' '@' | tr '!' '\n' | sed 's/@//g' > At_proteins_v2.fasta
```

This is a bit of hack and not something that you should assume will work on all FASTA files (particularly if they include characters like ! and @). The result of this operation should be that we create a new file (`At_proteins_v2.fasta`). The first line of this new file is a FASTA sequence definition. The second line now has the entire protein sequence and this pattern repeats for subsequent lines.

If your program works as intended, then you should be able to run it as follows and see the following output:

```
$ name_search.pl At_proteins_v2.fasta KEITH
```

```
AT5G58710.1
```

Project 5: K-mer analysis

In this project, you will write a program to investigate nucleotide patterns in introns. This project is inspired by the [Rose et al. 2008](#) paper which showed that introns near the promoter are compositionally distinct from introns farther downstream. This leads to a gene expression phenomenon known as ‘intron-mediated enhancement’ or IME.

We will make a program that will compare *Arabidopsis* introns that occur at different positions within a gene in order to determine just how different ‘early’ introns are compared to ‘late’ introns, in terms of their nucleotide composition.

K-mers

K-mer is another name for oligo. It’s just a string/word of some fixed length. If we have the sequence ATGCGA there are four possible k-mers of length 3: ATG, TGC, GCG, and CGA. Unlike codons, which skip every 3 nt, k-mers generally step by ones rather than threes. Codons are also strand-specific, while k-mers can be counted on one or both strands.

Coding

This project will use the `intron_IME_data.fasta` file in the `Data/Arabidopsis` directory. However, this is a multi-line FASTA file and you will first need to make a new FASTA file that rearranges each sequence to occupy only one line (see [Project 4](#) for how to do this).

Your program should have the following structure:

1. Provide a typical command-line interface allowing the user to choose the value for k.
2. Create two hashes to store the k-mer counts for 1st introns and other introns. You might name these `%count1` and `%count2`.
3. Read a definition line from your new FASTA file. Extract the intron number from the definition line. First introns will be labeled `i1`, second introns `i2`, and so on.
4. If it is the first intron, count all of the k-mers in the intron and add the counts to the `%count1` hash. If it is a more distant intron, add the counts to `%count2`.
5. After all the counting is done, create two new hashes, call them `%freq1` and `%freq2` to hold the frequencies for every k-mer.
6. Report the log-odds ratio of the frequency of each k-mer occurring in 1st introns vs. other introns.

The last part of the program asks you to report a log-odds ratio. Many programs report a score which is a base-2 log-odds ratio of observed over expected. A positive value indicates more observations than expected, and a negative value is fewer observations than expected. For example, let's say we have made 40 observations when we expected 20. The score is 1.0 because $40/20 = 2$, and the base-2 log of 2 is 1.0. Similarly, if we made 5 observations and expected 20, the score would be -2. Log-odds ratios are just positive and negative powers of 2.

In our program, we don't have an expectation, but rather two observations (first introns, and more distant introns). You can use the code below to calculate and report the log-odds ratio.

```
my $odds_ratio = $freq1{$kmer} / $freq2{$kmer};  
my $lod = log($odds_ratio); # Perl's log() function uses base e, so...  
my $lod2 = $lod / log(2); # ...need to convert base e to base 2  
printf "%s %.3f\n", $kmer, $lod2;
```

If you like brevity, the above four lines can be simplified as:

```
printf "%s %.3f\n", $kmer, log($freq1{$kmer} / $freq2{$kmer}) / log(2);
```

Which k-mers are most differently distributed? To help you observe this, you might want to pipe your output to the Unix `sort` command. Are some of the k-mers statistically or biologically significant? Those can be difficult questions to answer. You could try a Poisson model (see [Project 0](#)) to ensure that the counts are statistically robust. You might also randomly shuffle the intron sequences before counting (see [Project 3](#)) to determine if the k-mers are simply compositional biases or the result of some more interesting biological signal. In the paper, Rose et al. add up all the kmer scores for experimentally validated introns (they call this the IMEter score) and show there is a good correlation between IMEter score and gene expression.

P17. Regular expressions 101

Previously we learned about the matching and substitution operators (see [P6.4](#): the former lets you see whether a variable contains some text, the latter lets you substitute one string for another. These operators are much more powerful than they first appear. This is because you can use them to search for *patterns* rather than strings.

Task P17.1

Create the simple program below. We will be modifying it quite a bit in this section. Let's say we want to see whether a particular DNA sequence contains a codon for proline. There are four codons that encode for proline (CCA, CCC, CCG, or CCT). If we have a coding sequence which is already separated into codons, then one (tedious) way to do this would be with multiple conditional statements:

```
1.  #!/usr/bin/perl
2.  # codonsearch.pl
3.  use strict; use warnings;
4.
5.  my $seq = "ACG TAC GAA GAC CCA ACA GAT AGC GCG TGC CAG AAA TAG ATT";
6.  if      ($seq =~ m/CCA/) {print "Contains proline (CCA)\n"}
7.  elsif   ($seq =~ m/CCC/) {print "Contains proline (CCC)\n"}
8.  elsif   ($seq =~ m/CCG/) {print "Contains proline (CCG)\n"}
9.  elsif   ($seq =~ m/CCT/) {print "Contains proline (CCT)\n"}
10. else
           {print "No proline today. Boo hoo\n"}
```

Imagine doing this for all possible codons... tiresome. Ideally, we want a solution which would search for 'CCN' where N is A, C, G, or T. This is where [regular expressions](#) (also known as regexes) come in. Simply put, a regular expression defines a single pattern which describes a finite range of possibilities. Unix, Perl and other programming languages use a fairly standard way of implementing regular expressions (so anything you learn about them in Perl, will be very useful if you use Unix commands like 'grep' or sed).

Task P17.2

Delete lines 6–9 from the previous program and replace them with the following:

```
6. if ($seq =~ m/CC./){  
7.     print "Contains proline ($&)\n";  
8. }
```

Well that was easy! In the context of regular expressions, the dot . on line 6 represents *any single character*. It should not be confused with the use of a dot as the concatenate operator (see [P6.1](#)).

Line 7 contains a funny variable called \$&. This is another one of Perl's “special” variables (like \$_). Perl sets \$& to be the string matched by the most recent regular expression match.

Task P17.3

Change the regex to now see whether the sequence contains an arginine codon.

```
6. if ($seq =~ m/CG./){  
7.     print "Contains arginine ($&)\n";  
8. }
```

If you copied the sequence exactly as above, your script should be telling you that the \$seq variable contains an arginine codon, even though it doesn't. Can you see why?

The dot character will match *any* single character, including a space. So the last two letters of the ACG codon *plus* the space that follows matches the pattern, i.e. it matches CG. A better solution is to restrict the match to any character that is within a specified set of allowed characters.

Task P17.4

Replace line 6 with this more specific pattern.

```
6. if ($seq =~ m/CG[ACGT]/) {
```

The square brackets allow you denote a number of possible characters, any of which can match (this is known as specifying a *character class*). This is a much better solution when we have a limited range of characters and our regular expression can now only match four different strings (CGA, CGC, CGG, or CGT). Note though, that even when you have many characters inside the square brackets, you are only ever matching *one* character in the target sequence.

Sometimes biological sequences (DNA, RNA, and proteins) are sometimes represented as uppercase characters (ACG), and sometimes as lowercase characters (acg). It's also possible to download sequence files which use a mixture of upper- and lowercase (e.g. representing exons in uppercase, and introns in lowercase). How do you handle situations like this when trying to match patterns?

Task P17.5

Go back to line 5 and substitute some of the capital letters in our DNA string for lowercase characters, as in the example below.

```
5. my $seq = "ACG TAC GAA GAC ccA ACA GAT AGC gcg TGC CAG aaa TAG ATT";
```

There are two solutions to matching both upper- and lowercase. The first option is to make character classes (using the square brackets) that describe out all possible combinations of upper or lowercase letters that specify CCN:

```
6. if ($seq =~ m/[Cc][Cc][ACGTacgt]/){
```

The second option is much simpler. Use the ignore case functionality of the matching operator. This just involves appending an *i* after the second forward slash, and this will now mean that ccc, ccG, cCa, CaT, etc. will all count as a valid match.

```
6. if ($seq =~ m/GG[ACGT]/i){
```

Because there is no uppercase or lowercase standard for sequence files, it is good to always use the ignore-case option when working with sequences. This option also works with the substitution operator (introduced in [P6.5](#)). An alternative is to always convert a sequence to upper- or lowercase *before* you start processing it. The `uc()` and `lc()` functions perform these operations. If the first thing you do with a new sequence is make it all uppercase or all lowercase, then you don't need to use the ignore-case option later on in your script.

Character ranges

Another useful option when specifying a character class is to use a dash to specify a range of characters or numbers.

```
9. if ($seq =~ m/[a-z]/){
10.     print "Contains at least one lower case letter\n";
11. }
```

Perl defines several symbols for common character classes. Two of the most useful ones are \s and \S which are used to match whitespace and non-whitespace respectively. Matching whitespace allows you to match spaces that might be the result of space and/or tab characters (not always obvious when you are processing data from a text file).

anchors

To ensure that a pattern matches the beginning or ending of a string, you can use the ^ and \$ symbols. This is the same as when using regexes in Unix:

```
if ($dna =~ m/^ATG/){ ... } # matches if $seq started with 'ATG'
if ($dna =~ m/TGA$/){ ... } # matches if $seq ended with 'TGA'
if ($pep =~ m/^M[ST]G$/){ ... } # matches if $pep started with M, followed by S
or T, and ended with G
```

Negated character classes

You can also specify character classes that should *not* occur as part of a input string that you are trying to match. Unfortunately, the symbol that is used to specify this is the same symbol as the anchor character we just showed you ^. This can be confusing, but the character is used a little differently when making negated character classes. Here is how you could find whether a protein sequence either contained polar amino acids (D, E, R, K, or H) or whether it *didn't* contain any:

```
if ($pep =~ m/[DERKH]/){ ... } # would only match if $pep has at least one polar residue
if ($pep =~ m/[^DERKH]/){ ... } # would only match if $pep has no polar residues
```

The basic rule here is that if ^ is the first character inside the square brackets of a character class, then it becomes a negated character class.

Repetition

If you want to match several characters or character classes in a row, you can use various repetition symbols+, ?, and * which let you match ‘1 or more’, ‘zero or 1’ and ‘zero or more’ characters respectively:

```
if ($text =~ m/A+/) { ... } # matches 1 or more As.  
if ($text =~ m/A?/) { ... } # matches zero or 1 As.  
if ($text =~ m/A*/) { ... } # matches zero or more As.  
if ($text =~ m/^A+B+$/) { ... } # matches 1 or more As at start of string & 1  
or more Bs at end
```

If you want to match a character between ‘m’ and ‘n’ times or to match exactly ‘m’ times, then you can use the following syntax:

```
if ($text =~ m/A{1,3}/) { ... } # matches between 1 and 3 As  
if ($text =~ m/C{42}/) { ... } # matches exactly 42 Cs  
if ($text =~ m/T{6,}/) { ... } # matches at least 6 Ts
```

Alternation

You can match more than one pattern at once if you separate them with pipe symbols | (in this context, the pipe character is known as the alternation operator). E.g. to match any of the three stop codons you could use:

```
if ($seq =~ m/TAA|TAG|TGA/) { ... } # matches TAA *or* TAG *or* TGA
```

Parentheses can be used to clarify what text is part of the alternation:

```
if ($text =~ m/Super(man|girl|boy|rabbit)/) { ... } # matches various superheroes
```

Combining metacharacters

It can be confusing to learn about regular expressions in Perl because there are so many of these special *metacharacters* that you can use inside patterns. It is even more confusing when some of these characters have completely different meanings outside of their use in regular expressions. All of the metacharacters we have seen can be combined to produce very powerful — and often very confusing — regular expressions. Consider the following regular expression:

```
if ($seq =~ m/^ATGCC[ACGT]GG[ACGT]N{6,9}(TAG|TGA|TAA)$/) { ... }
```

This would produce a match as long as:

1. \$seq started with a start codon ^ATG
2. was followed by a single proline codon CC[ACGT]
3. was followed by a single glycine codon GG[ACGT]
4. was followed by between 6–9 N nucleotides (unknown bases) N{6,9}
5. ended with one of the three valid stop codons (TAG|TGA|TAA)\$

Backslash

We have just seen many special regular expression metacharacters. But what if you wanted to actually match one of these characters? E.g. you wanted to see whether \$species contained the text ‘A. thaliana’. The following would probably work, though not as you might expect:

```
if ($species =~ m/A. thaliana/) { ... }
```

The . is A. thaliana is being used as a regular expression metacharacter, meaning that it will match any single character. Therefore if \$species contained the text AB thaliana, then this would produce an erroneous match. The special meaning of any metacharacter can be “escaped” by prefixing it with a backslash. Therefore, you can match a dot . with \. and it will only match a dot:

```
if ($species =~ m/A\. thaliana/) { ... }
```

The backslash is also used to escape the special meaning of other characters in Perl. What if you wanted to print the value of the variable \$answer but also include the text \$answer in the output string? Or what if you wanted to print \n but not have it print a newline?

```

my $answer = 3;
print "\$answer is $answer\n";
print "This is a newline character: \\n\\n";

```

Perl regular expression metacharacters

Symbol	Meaning
.	any character
\w	alphanumeric and _
\W	any non-word character
\s	any whitespace
\S	any non-whitespace
\d	any digit character
\D	any non-digit character
\t	tab
\n	newline
*	match 0 or more times
+	match 1 or more times
?	match 0 or 1 times
{n}	match exactly n times
{n,m}	match n to m times
^	match from start
\$	match to end

P18. Extracting text

You will often want to extract some strings from a large file. For example, you may be processing a GFF or GenBank file to retrieve specific coordinates or features. Hopefully you will not be parsing HTML for email addresses! The power of regexes not only fuels bioinformatics, but also spam.... There are a number of ways to pull out specific patterns from a file. We have seen a couple of these already; e.g. in P17.2 we saw how \$& contains the string of the last pattern match. But this doesn't let you extract multiple strings at once, so it has limited use. If you happen to be parsing a tab-delimited file, rejoice because you can just use the `split()` function.

```
1. while (<>) {  
2.     my @fields = split("\t", $_); # each line gets placed in $_  
3. }
```

If the file happens to be space-delimited, you can even abbreviate even further.

```
2. my @fields = split; # \s+ and $_ are assumed
```

But you won't always have tab-delimited text. Some files are much more complex.

Task P18.1

Let's retrieve all the gene names and coordinates from a GenBank file. Take a look (using less) at the file `Unix_and_Perl_course//Data/GenBank/E.coli.genbank` and scroll down (or search) until you find the 'gene' keyword in the 'FEATURES' section of the file:

FEATURES	Location/Qualifiers
source	1..4686137 /organism="Escherichia coli str. K12 substr. DH10B" /mol_type="genomic DNA" /strain="K-12" /sub_strain="DH10B" /db_xref="taxon:316385"
gene	190..255 /gene="thrL" /locus_tag="ECDH10B_0001" /db_xref="GeneID:6058969"

The coordinates of the gene are given on the same line 190..225. One line below contains the gene name as /gene="thrL". Page down a bit and you will find another gene on the reverse (complement) strand:

gene	complement(5683..6459) /gene="yaaA" /locus_tag="ECDH10B_0006" /db_xref="GeneID:6061859"
------	--

In order to parse this file, we must deal with genes on the complement strand and also the fact that all the information isn't on the same line. The following program reports the name and coordinates of all genes. Remember to specify the name of the GenBank file when you run the script.

```

1.  #!/usr/bin/perl
2.  # parse_genes.pl
3.  use strict; use warnings;
4.
5.  while (my $line = <>) {
6.      if ($line =~ /^\s{5}gene/) {
7.          my ($beg, $end) = $line =~ /(\d+)\.\.\.(\d+)/;
8.          $line = <>;
9.          my ($name) = $line =~ /="( .+ )"/;
10.         print "$name $beg $end\n";
11.     }
12. }
```

Lines 1–5 should look very familiar by now. Line 5 sets up the `while` loop which will loop over every line of the specified file.

Line 6 asks if `$line` starts with 5 spaces `^\s{5}` followed by the word ‘gene’. GenBank format is very strict about how many spaces begin each line. Had we been lazy, we could have used `\s*` or `\s+` to match the spaces at the start of the line.

Line 7 matches the coordinates from `$line` and assigns them to the `$beg` and `$end` variables. We have seen matches like this before but have not tried assigning the results to anything. Regular expressions matches in list context return values from parenthesized patterns. You might want to repeat the phrase a dozen times or so. It’s that important.

Regular expressions in list context return values from parenthesized patterns

Regular expressions in list context return values from parenthesized patterns

***Regular expressions in list context return values from parenthesized
patterns...***

Line 8 gets another line of input from the GenBank file by using the file operator `<>` once again. We need to read another line because the gene’s name is one line below the gene’s coordinates. The logic of the `while` loop becomes “keep looping over lines and *if I see something that looks like a pair of gene coordinates, then read one more line from the file*”.

Line 9 matches the gene name using the regular expression pattern `= " (.+)"`. The parentheses ensure that any text between the quotation marks is captured (in `$name`). We use this very generalized pattern (match one or more of any single character) because gene names sometimes contain strange characters and spaces, even though they don't in *E. coli*. Nearly all CDSs in the GenBank file have a gene name, but because a few don't we also have to be able to capture lines that match either a `/gene=` or a `/locus_tag=` pattern.

More Info

We've only scratched the surface of regular expressions. For more information, read the Perl man pages.

```
$ man perlrequick  
$ man perlre
```

P19. Boolean logic

Back in [P13.2](#) we saw the following statement:

```
open(my $in, "<$ARGV[0]") or die "error reading $ARGV[0] for reading";
```

The meaning of this is pretty clear: open the file or die trying. We understood the `or` part as something that only happens if the file doesn't open. How exactly does this work? All Perl functions return a True or False value. False values are 0 and the empty string `" "`. All other values are true. This point bears repeating:

False values are 0 and the empty string " ". All other values are true.

So we can understand the `open()` statement above as a more concise version of the following:

```
$return_value = open(my $in, "< $ARGV[0]");
if ($return_value == 0) {
    die "can't open file $ARGV[0]\n";
}
```

But why doesn't the `die()` statement get executed if the return value is True? The answer is because the *whole* statement — from the `open()` function through to the semicolon — that is evaluated with Boolean logic. The Boolean operators are `and`, `or`, `not`. Let's review how `and` and `or` behave:

```
True and True = True
True and False = False
False and True = False
False and False = False
True or True = True
True or False = True
False or True = True
False or False = False
```

If the `open()` function works then that function returns true, which then means that the entire Boolean expression `open() or die` must be True. Perl does not attempt to evaluate more than it needs to, so once `open()` succeeds, it short-circuits the rest of the statement. I.e. if the left-hand side of any `or` statement evaluates as True, Perl doesn't need to evaluate what's on the right-hand side of the `or` statement. Likewise, if the left-hand side of any `and` statement evaluates as False, then there is no need to evaluate the right-hand side.

Back in [P11.1](#) we saw the following statement:

```
@list = sort { $a <=> $b or uc($a) cmp uc($b) } @list
```

What's going on here? The sorting function first compares `$a` and `$b` numerically. If their numeric values are zero (e.g. because they are strings), the expression `$a <=> $b` returns zero. Perl must then evaluate the right side `uc($a) cmp uc($b)` to determine if the whole expression is true or false. So numbers get compared first, and if they are equal, they are further compared by ASCII value.

Project 6: Codon usage of a GenBank file

The goal of this project is to create a codon usage table for any bacteria in GenBank. For example, we could calculate the codon usage of *E. coli*, *B. subtilis*, or *Y. pestis*. You will find GenBank files for several bacteria in the Data/GenBank directory. You will use write a program and use it to analyze their codon usage. Later, we will compare them with Information Theory! But that is for another day.

Use the Unix command `less` to look at the *E. coli* file. GenBank files have a well-defined structure and each file may contain thousands of sequences, but this file contains just one. After the header section which describes details of the entry in GenBank, and various bibliographic details you will see the ‘FEATURES’ section which describes all of the features that have been annotated on this sequence. In this section you will see many ‘CDS’ (coding sequence) features along with their corresponding protein translations. Take a look at several of these and see if you can determine how GenBank distinguishes those genes which are on the reverse DNA strand. At the end of the file you will find the DNA sequence of the genome. Note that GenBank files provide sequence coordinates which appear at the start of each sequence line. Your tasks are:

1. Extract the nucleotide sequences that correspond to these proteins
2. Count the codons in each coding sequence
3. Print a summary of the codon usage from all genes combined

This might sound challenging, and it is. But we have the knowledge to do it. The strategy you will use for this program is outlined below:

1. Create a `$genome` variable that will store the *E. coli* genome sequence
2. Open the GenBank file
3. Skip all lines until you get to the sequence (look for the text ‘ORIGIN’)
4. Process each line of sequence as you capture it:
 1. Remove the digits that start each line
 2. Remove the spaces
 3. Remove the newline at the end of each line
 4. Add each line of processed sequence to `$genome`
5. Close the GenBank file
6. Re-open the GenBank file
7. Process each line
 1. Find lines which describe CDS features

2. Extract the start and end coordinates of each CDS
3. Use these coordinates to extract the corresponding DNA sequence of the CDS from \$genome
4. Reverse-complement CDS sequence if necessary
5. Count codons of current CDS, add to running totals for all CDSs
8. Report frequencies of all codons

Tips

- Use a named filehandle for steps 2 and 6 (e.g. `open(my $genbank, "<", $ARGV[0])`). Don't use the file operator on its own `<>`.
- Make sure that the coding sequences are correct. Most should start with ATG and end with a stop codon. If they do not, you may need to improve your code.
- Remember that sequence coordinates start at '1' but the `substr()` function starts counting positions at '0'.
- Some proteins may contain the peptide 'CDS' (cysteine-aspartate-serine). So be careful when searching for the text 'CDS' in order to find CDS features.
- Don't use this program with eukaryotic GenBank sequences which will have multi-exon CDS features. Describing the joins of the various exons can take several lines, which makes parsing the file a little more difficult.
- Be careful about making assumptions about how biology works (biology has a nasty habit of kicking you in the butt just to surprise you!). I.e. don't assume that all DNA characters in this file will be A, C, G, and T. Rather than assume, check!
- Test each section of code as you write it. If you can't correctly extract the genome sequence, then there is no point going any further with your script!

P20. Functions (a.k.a. subroutines)

We've seen several built-in functions already such as `print()` and `rand()`. Programming would be pretty difficult if we didn't have these. Wouldn't it be great to make your own functions? This is where the real power of programming lies. In Perl, we can make our own functions by using *subroutines*. A subroutine is like a little parcel of code that usually performs one focused task. E.g. calculate the minimum value from a set of numbers, or translate a DNA sequence into a protein sequence. Subroutines are ideally suited to small coding tasks that you might need to perform multiple times within a single script.

Task P20.1

Create the following program. It takes a DNA sequence that you specify on the command-line and runs three simple checks to see whether the sequence might represent a valid CDS. If the sequence fails any check, an error message is printed and we will simply use a subroutine to print the error message:

```

1.  #!/usr/bin/perl
2.  # sequencecheck.pl
3.  use strict; use warnings;
4.
5.  # take sequence from command-line and make upper case
6.  my $seq = uc($ARGV[0]);
7.
8.  if ($seq !~ m/^ATG/){           # test for start codon
9.      print_error();
10. }
11. elsif($seq !~ m/(TGA|TAG|TAA)$/){ # test for stop codon
12.     print_error();
13. }
14. elsif($seq !~ m/^([ACGTN]+$/){   # test for non DNA characters
15.     print_error();
16. }
17. else{
18.     print "$seq looks like a valid CDS\n";
19. }
20.
21. sub print_error {
22.     print "$ARGV[0] is not a valid sequence for a CDS\n";
23.     print "It may not start with an ATG start codon\n";
24.     print "It may not end with a stop codon\n";
25.     print "It may contain non ATCGN DNA characters\n";
26. }
```

Lines 9, 12, and 15 all call the `print_error()` subroutine which is declared on line 21. Subroutines behave just like any other Perl function, but unlike built-in functions like `print()`, you must include parentheses. To declare a function/subroutine, you use the `sub` keyword. This is immediately followed by the name of the function and a block structure delimited by curly braces.

Any time Perl sees the subroutine name `print_error()` it immediately jumps to the block of code that starts `sub print_error`. When Perl finishes processing the code in the subroutine it immediately returns back to where it originally was in the main body of the script. Depending on the sequence that is provided to the script, this code might jump back and forth between the subroutine three times (if all three errors are present). Because Perl will jump straight to the subroutine no matter where it is located, you can define the subroutine anywhere in the script. However, there is a convention to putting subroutines at the end of a Perl script (though in other languages you might find them at the beginning). You can do it either way, but try to be consistent.

This script is not a very good script, it prints the same error message regardless of what error is found in the sequence. However, you should see that by using a subroutine we only need to write the code to produce the error message in one place. If we wanted to add or change the error message, we only need to edit the code in one place.

Task P20.2

When we use subroutines it is far more common to pass the subroutine one or more variables and get the subroutine to do something useful with those variables. Create the following program. It reads a file of sequences and computes the GC% of each one.

```
1.  #!/usr/bin/perl
2.  # gc.pl
3.  use strict; use warnings;
4.
5.  while (my $seq = <>) {
6.      chomp($seq);
7.      gc($seq);
8.  }
9.
10. sub gc {
11.     my ($seq) = @_;
12.     $seq = uc($seq); # convert to upper case to be sure
13.     my $g = $seq =~ tr/G/G/;
14.     my $c = $seq =~ tr/C/C/;
15.     my $gc = ($g + $c) / length($seq);
16.     print "GC% = $gc\n";
17. }
```

Before you run this script you will need to create a text file which contains a few lines of DNA sequence. Use the name of the file when you run the script e.g. `gc.pl dna_file.txt`

Line 5 sets up a `while` loop to loop over any (and every) file specified on the command-line.

Line 7 calls the `gc()` subroutine and passes it the `$seq` variable. To pass a variable to a subroutine, include it between the parentheses that follow the subroutine name. The `gc()` subroutine will be called for every line that is present in your input file.

Lines 10–17 contain the `gc()` function. Because line 7 passes a variable to the subroutine, we must add code to receive it. Subroutines receive arguments via the special `@_` array. Any variables that are passed to the subroutine will be stored in this array. Just as we don't like to use `@ARGV` throughout our script (because the name isn't very meaningful) we also want to extract any variables from `@_` and assign them to variables with new names.

Line 11 shows a typical list assignment, the first element of the `@_` array is copied to `$seq`. You may see some programs using the `shift()` function to remove elements of the `@_` array in one of two ways:

```
my $seq = shift(@_);
my $seq = shift;
```

In the second example, `shift()` is used without specifying an array name. If no array is specified the `shift` function uses the `@_` array by default.

Note that we use `$seq` again as a variable name within the subroutine, we'll explain why in the next section. For now, just accept that the `$seq` in the subroutine is unrelated to the other `$seq`.

Anything passed to the `@_` array is copied. This means that line 7 is effectively sending a copy of `$seq` to the `gc()` subroutine. In other words, `$seq` is unchanged by anything that happens in `gc()`. You can test this by adding a `print "$seq\n"` statement after line 7.

Task P20.3

The previous program demonstrated a much better use of subroutines, but it is still not ideal. Maybe we don't always want to print the value of GC% as soon as we calculate it. In general, we often want a subroutine to calculate something and send that back to wherever we called the subroutine from. We can do this in Perl by using `return` values within a subroutine. Let's make a script that uses the melting temperature code that we saw earlier in [P15.1](#), but that now puts this code in its own subroutine:

```
1.  #!/usr/bin/perl
2.  # tm.pl
3.  use strict; use warnings;
4.
5.  while (my $seq = <>) {
6.      chomp($seq);
7.      my $tm = tm($seq);
8.      print "Tm = $tm\n";
9.  }
10.
11. # calculate Tm
12. sub tm{
13.     my $seq = shift;
14.     my $A = $seq =~ tr/A/A/;
15.     my $C = $seq =~ tr/C/C/;
16.     my $G = $seq =~ tr/G/G/;
17.     my $T = $seq =~ tr/T/T/;
18.     my $tm = 2 * ($A + $T) + 4 * ($C + $G); # simple Tm formula
19.     return($tm)
20. }
```

First of all, let's look at line 19. This line returns a value from the subroutine using the `return()` function. You can use `return` anywhere in the subroutine and it will exit at that point and return to wherever the subroutine was called from. I.e. if there was a `print` statement on line 20, it would never be performed. You can return multiple values in a `return` statement or even none. Sometimes we just return 1 or 0 to indicate success or failure.

So what does Perl do with returned values? If we now look at line 7 we can see that the output of the `tm()` function is assigned to a variable. If we had wanted to make our code more concise (which is not always a good thing) we could have replaced lines 7 and 8 with:

```
7. print "TM = ", tm($seq), "\n";
```

When Perl evaluates this code, it knows that the first thing that has to be dealt with is the call to the `tm()` subroutine. When that code is finished, the subroutine will return a value (or potentially a list of values) and in this case we plug that value straight into a `print()` function rather than saving it in a variable. We could also have replaced lines 18 and 19 with the following:

```
18.    return(2 * ($A + $T) + 4 * ($C + $G));
```

In this case, Perl will first make the calculation of the melting temperature and then return the resulting value. Most people find it easier to first store this result into a variable and then return the variable.

Task 20.4

So far we have only ever passed one variable to a subroutine and returned just one thing back to the calling function. It is very common to pass and return multiple arguments. It is also common to have multiple return statements which are all dependent on the outcome of some logical test.

Modify the GC% script in order to pass two things to the subroutine: the sequence plus a GC% threshold (a floating point number which will be stored in a `$threshold` variable within the subroutine). If the GC content is above the value of `$threshold` then we will return “High GC” else we will return “Low GC”.

To simplify things, you can specify the sequence and the threshold value on the command-line (instead of reading a file). We also want the script to print out whether each sequence is high or low GC, but that print statement must not be in the subroutine! You will have to look up how to pass two things to a subroutine. The end of the subroutine will look like the following:

```

sub gc {
    #
    # missing code to go here
    #

    $seq = uc($seq);
    # convert to upper case to be sure
    my $g = $seq =~ tr/G/G/;
    my $c = $seq =~ tr/C/C/;
    my $gc = ($g + $c) / length($seq);

    if($gc > $threshold){
        return("High GC");
    } else{
        return("Low GC");
    }
}

```

Note that if you use multiple return statements (as in this example), they should always be part of a logical test such that only one return statement is ‘seen’ by the code. E.g. the following subroutine would be pointless:

```

sub pointless {
    my ($some_value) = @_;
    return("Yay!"); # subroutine exits at this point
    return("Boo!"); # this line will *never* be evaluated by Perl
}

```

Why use subroutines?

As your programs get longer you might find yourself wanting to do the same thing more than once in your program. Maybe part of your program takes two input sequences and calculates their percentage similarity. Your program might then modify those sequences and then recalculate the percentage similarity. Without subroutines you would have to have the same lines of code in two places in your script. This is a bad idea. Where possible, code should be reused. As soon as you find yourself writing the same code in more than one place, you should think about putting that code in a subroutine.

Subroutines can also help improve the readability of your code. Rather than see all of the details of how you calculate some mathematical function, it might be cleaner to keep that code in a subroutine and this keeps it hidden from the main body of the code.

P21. Lexical variables and scope

By default Perl lets you create variables whenever you need them and they are then available throughout your entire program. We call this global scope. Many people would argue that using global variables is dangerous and certainly not the best way of programming. Instead, we could (and should) use *local* variables. This is something that we have already made you do in nearly all of these Perl scripts. That's because when we include the `use strict;` statement this requires that we need to declare all variables, arrays, and hashes with the `my` keyword. This means that we are declaring a local variable.

Consider the following program that deliberately does not contain the `use strict;` statement. The program takes a sequence and counts how many codons it contains (using a subroutine). Run the program and observe the output.

```
1.  #!/usr/bin/perl
2.  # no_strict.pl
3.  use warnings;
4.
5.  my $seq = "atg att gaa cca tga";
6.  $codons = count_codons($seq);
7.  print "$seq contains $codons codons\n";
8.
9.  sub count_codons {
10.    $seq = shift;
11.    $seq = uc($seq);  # convert to upper case to be sure
12.    $seq =~ s/\s+//g; # remove all whitespace from sequence
13.    $codons = length($seq) / 3;
14.    return($codons);
15. }
```

Line 7 prints \$seq but it now prints the version of \$seq that was modified in the subroutine (on lines 11–12). Without the `my` declarations, the two \$seq variables are no longer separate entities. This is probably not the behavior that we wanted. When we *don't* declare variables with `my`, they become *global variables*. Changing a global variable in any one part of the program changes it everywhere else. We should never do this, it is just about the worst thing you can do as a programmer.

To make sure we do not affect other parts of a program, we will always choose to make variables inside a function exist only within that function. The `my` keyword does this for us and it creates what is known as a lexical variable. These are variables that live and die within a set of curly braces (a block). This means that we can reuse variable names to store different things as long as they exist within different blocks of code.

Task P21.1

The following program demonstrates the use of lexical variables:

```
1.  #!/usr/bin/perl
2.  # lexical.pl
3.  use strict; use warnings;
4.
5.  my $x;                      # variable declaration without assignment
6.  $x = 1;                      # variable assignment
7.  my ($y, $z) = (2, 3); # you can declare and assign variables as a list
8.  if ($x < $y) {
9.      my $z = 10;
10.     print "inside:  x = $x, y = $y, z = $z\n";
11. }
12. print "outside: x = $x, y = $y, z = $z\n";
```

It is critical that you completely understand the concept of the scope of a variable. The scope is that part of your code that is allowed “to see” your variable. A variable’s scope starts from the point it is first declared and ends at the next enclosing curly bracket that’s at the same logical level as the variable. In this script, there is a \$z variable that is ‘born’ at line 9 and ‘dies’ at line 11. Importantly, this \$z is not the same as the \$z on line 7. The inner \$z effectively hides the outer \$z as soon as it is declared. As soon as we reach line 12, we are no longer in the scope of the inner \$z and revert to the scope of the outer \$z.

Variables in a wider scope are visible in a narrower scope. So we can see \$x and \$y at line 11. Variables in a narrower scope do not exist in a wider scope. To see this more clearly, try changing lines 9 & 13 to to the following:

```
9. my ($z, $q) = (10,15);  
  
13. print "outside: $x $y $z $q\n";
```

This should produce an error because \$q doesn't exist outside the loop so we shouldn't be able to print it on line 13. After line 11, \$q no longer exists.

Loop Variables

Lexical variables in loops look a little strange because they are declared outside the curly braces:

```
1. for (my $i = 0; $i < 10; $i++) {  
2.     # code inside for loop  
3. }  
4.  
5. foreach my $seq (@seq) {  
6.     # code inside foreach loop  
7. }
```

The loop counter variable \$i is declared on line 1 and dies at line 3. So even though it appears outside the curly braces, its scope is actually the entire loop. Likewise, \$seq is born anew with each iteration of the foreach loop at line 4 and dies each time at line 6. This means that if you have multiple `for` loops in a script you could, and indeed *should*, reuse the same loop counter variable name. The exception to this is when you have nested `for` loops where the convention is to use \$i, then \$j, then \$k etc. E.g.

```
for (my $i = 0; $i < 10; $i++) {  
    # outer loop, using $i  
    for (my $j = 0; $j < 10; $j++) {  
        # inner loop, using $j  
    }  
}
```

Safer programming: `use strict`

All variables should be lexical variables. To ensure this behavior, include `use strict` in *all* your programs. In fact, your programs should always contain a line like this:

```
use strict; use warnings;
```

You may run into someone who thinks that `strict` and `warnings` are a hassle. Feel free to talk to, dine with, or even marry this person, but in no circumstances should you share code with them!

P22. Sliding window algorithms

One of the most common sequence analysis scenarios is to look at the local composition of a sequence rather than the global composition. For example, a genome might be 45% GC, but it might be more GC-rich in CpG islands and less GC-rich in introns. Similarly, a protein may have hydrophobic and hydrophilic regions, and you might want to identify these.

Task P22.1

Create the following program and run it with a few window sizes to observe the smoothing effect of larger window sizes. In this algorithm, there are two loops. The outer loop moves the window along the sequence. The inner loop counts the nucleotides inside the window.

```
1.  #!/usr/bin/perl
2.  # sliding.pl
3.  use strict; use warnings;
4.
5.  die "usage: sliding.pl <window> <seq>" unless @ARGV == 2;
6.
7.  my ($window, $seq) = @ARGV;
8.
9.  # outer loop
10. for (my $i = 0; $i < length($seq) - $window +1; $i++) {
11.     my $gc_count = 0;
12.
13.     # inner loop
14.     for (my $j = 0; $j < $window; $j++) {
15.         my $nt = substr($seq, $i + $j, 1);
16.         $gc_count++ if $nt =~ /[GC]/i;
17.     }
18.
19.     printf "%d\t%.3f\n", $i, $gc_count/$window;
20. }
```

Note that to make sure all the windows are the same size and that the last few windows do not run off the end of the sequence, we have to stop the sliding before it gets to the end. The conditional part of line 10 might look a bit strange `$i < length($seq) - $window +1`, but it's ensuring that we don't go past the end of our sequence.

Task P22.2

Here is an alternative approach using a single loop that reuses our `gc()` function. Replace lines 14–19 with the following two lines and then copy the `gc()` subroutine into the script. This strategy is slightly less efficient because there is some overhead in every function call. But we think you will agree that it reads much better!

```
14. my $subseq = substr($seq, $i, $window);  
15. printf "%d\t%.3f\n", $i, gc($subseq);
```

Task P22.3

Did you notice that both of the previous sliding window algorithms recount the same bases? Imagine a window of 1000 bases. The total number of Cs and Gs is not going to change much as the window slides over one more position. In fact, the number of Gs or Cs can only change by plus or minus 1. Why count 1000 letters when you only need to change one value? You don't have to. If you count the Cs and Gs in the initial window, you can then update the counts as you slide along. This algorithm turns out to much more efficient for large windows. You might want to come back to this task at a later time. It's doesn't introduce any new concepts, but the code is definitely more complicated:

```

1.  #!/usr/bin/perl
2.  # sliding_fast.pl
3.  use strict; use warnings;
4.
5.  die "usage: sliding_fast.pl <window> <seq>" unless @ARGV == 2;
6.  my ($window, $seq) = @ARGV;
7.
8.  # initial window
9.  my $gc_count = 0;
10. for (my $i = 0; $i < $window; $i++) {
11.     my $nt = substr($seq, $i, 1);
12.     if ($nt =~ /[CG]/i) {$gc_count++}
13. }
14. printf "%d\t%.3f\n", 0, $gc_count/$window;
15.
16. # all other windows
17. my $limit = length($seq) - $window + 1;
18. for (my $i = 1; $i < $limit; $i++) {
19.     my $prev = substr($seq, $i - 1, 1);
20.     my $next = substr($seq, $i + $window - 1, 1);
21.     if ($prev =~ /[CG]/i) {$gc_count--}
22.     if ($next =~ /[CG]/i) {$gc_count++}
23.     printf "%d\t%.3f\n", $i - $window + 1, $gc_count/$window;
24. }
```

Sometimes you must choose between readability and speed. Most of the time, you should let readability take precedence. Why? Because readable code is easier to debug and maintain. If you absolutely need something to run faster, there are a variety of possible solutions including (a) buying a faster computer (b) changing the structure of the algorithm (c) programming in a compiled language such as C.

P23. Function libraries

Once you develop some useful functions like `gc()`, you will find that you want to use them again and again. One way to re-use code is to simply copy-paste your functions from one program to another. Since functions are like mini programs, this usually works just fine. But what if you discover an error in the function and now you want to fix all the programs that use it? You'll have to search all your programs and fix each one. Wouldn't it be better if the programs all used the exact same code? Absolutely!

A function library is a file where you keep a group of related functions. Any program you write can use these functions. Having your own personal library makes programming much simpler. But the real power of libraries comes when you use other people's libraries. *The only thing better than your function library is someone else's.* Enough talk, let's create our first library.

Perl uses the term package or module for function library. They (mostly) mean the same thing. All Perl modules are saved with the `.pm` suffix (for Perl module). The first line of a module uses the package statement and the last line is simply `1;`. All of the functions go between those statements. There is no limit to the number of functions you can place in a library.

Task P23.1

Save the following code in your 'Code' directory as `Library.pm`. This is not a particularly descriptive name, but it will do for now.

```
1. package Library;
2. use strict; use warnings;
3.
4. sub gc {
5.     my ($seq) = @_;
6.     $seq = uc($seq); # convert to upper case to be sure
7.     my $g = $seq =~ tr/G/G/;
8.     my $c = $seq =~ tr/C/C/;
9.     my $gc = ($g + $c) / length($seq);
10.    return $gc;
11. }
12.
13. 1;
```

The `gc()` function can now be used in any program you write as long as `Library.pm` is in the same directory as the script that wants to use it. Now let's see how we use libraries.

Task P23.2

Go back to `sliding.pl` ([P22.1](#)) and insert the line `use Library;`. One generally puts such statements at the top of a program, but you can put them anywhere. This simple statement allows the program to use any of the functions in the library. To call `gc()`, we must prepend the function call with the library name `Library::gc()` (as in line 9 below). The reason for this is that we might be using several different libraries, each of which could have their own `gc()` function. Including the library name makes it clear that things such as `Library::gc()` and `OtherLibrary::gc()` are separate functions.

```
1.  #!/usr/bin/perl
2.  # sliding.pl use strict; use warnings;
3.  use Library;
4.
5.  die "usage: sliding.pl <window> <seq>" unless @ARGV == 2;
6.  my ($window, $seq) = @ARGV;
7.  for (my $i = 0; $i < length($seq) - $window +1; $i++) {
8.      my $subseq = substr($seq, $i, $window);
9.      printf "%d\t%.3f\n", $i, Library::gc($subseq);
10. }
```

If you ended up writing several different scripts, all of which needed to calculate the GC content of a sequence, you could use this single piece of code in all of those scripts. This is much cleaner, and more efficient, than maintaining exactly the same code across many different scripts. As a general rule:

If a single script has duplicate code that does the same thing, put the code in a subroutine

If multiple scripts contain the same subroutines, put the subroutine in a library

Project 7: Useful functions

This project has a number of sub-projects, each of which is based around a new useful function that you should add to your library.

Project 7.1

Create a function that reverse-complements a DNA sequence. Ideally, this should even work if the sequence contains nucleotide ambiguity characters such as R, Y, M, K, etc.

Project 7.2

Write a function that computes the entropy of a sequence. The entropy is simply the sum of $\$x * \log(\$x)$, where $\$x$ is the frequency of each letter. Unbiased DNA has 2 bits of entropy. A biased composition results in less < 2 bits. To convert from nats to bits, divide by $\log(2)$. Use the entropy function in combination with a sliding window to find low entropy regions of a sequence.

Project 7.3

Using the standard genetic code, write a function that returns the translation of a nucleotide sequence. Use an 'X' for an ambiguous codon and a '*' for a stop codon. Write a program using this function finds the longest ORF in a sequence.

Project 7.4

Write a function that returns the codon frequencies in a GenBank file. The function should take a file name as the argument, and return the frequencies in either a hash or array. You should be able to modify your code from [Project 6](#).

Project 7.5

Write a program that compares the codon usage of two bacteria. Use Kullback-Leibler distance (relative entropy) to compare the codon frequencies. K-L distance is the sum of $\$x * \log(\$x / \$y)$ where $\$x$ is a codon frequency in one genome and $\$y$ is the the frequency of the same codon in another genome. Experiment with several bacteria.

Project 7.6

Membrane-spanning regions of proteins are hydrophobic. To find potential trans-membrane domains, create a Kyte-Doolittle hydropathy function. For this you will need to look up the hydrophobicity of each amino acid and calculate the average hydrophobicity in a sliding window. The function should print all the hydrophobicity values. Alternatively (and better), the function can return an array of values.

P24. Interacting with other programs

Let's say you want to run BLAST 1000 times and retrieve the output. No problem, there are a number of ways to get information from other programs into your program. The simplest one is the backticks operator ``. This looks like an apostrophe but is actually a different character and will be hiding somewhere on your keyboard. Whatever you put in backticks will be executed in the Unix shell, and the output will be returned to you in an array or scalar depending on how you asked for it.

Task P24.1

Let's try an example of capturing the output of the ls command in two different ways:

```
1. #!/usr/bin/perl
2. # system.pl
3. use strict; use warnings;
4.
5. my @files = `ls`;
6. print "@files\n";
7. my $file_count = `ls | wc`;
8. print "$file_count\n";
```

One line 5, we capture the program output in list context, and you should see that each element of @files is a separate file or directory that was returned by the ls command.

On line 7 we capture program output in scalar context and all output is assigned to a single variable.

Another way to run an external program is with a `system()` function call. Whatever you put into a `system()` call is run just like the Unix command line. Unlike the `open()` function which returns 0 when it fails, the `system()` function returns 0 when it succeeds (there are good reasons for this, but for now let's just be angry about it). It is generally preferable to use the `system()` function rather than backticks as this gives you more control of testing whether the Unix command that you run actually worked or not. Add the following line to your program:

```
9. system("ls > foo") == 0 or die "Command failed\n";
```

You will now have a file called foo in the directory where you ran the script that contains the contents of the ls command. To get this into your program we can use the open() function as we have seen before. But this time, we will do something slightly different with it:

```
10. open(my $in, "<", "foo") or die "Can't open foo\n";
11. my @files = <$in>; # reads the entire file into @files
12. close $in;
13.
14. foreach my $file (@files) {
15.     print "$file\n"
16. }
```

On line 11 we introduce a shorthand for reading all the lines of a file at once. Be careful with this because you could run out of memory if you slurp up a big genome.

You will most commonly use filehandles to read from files, or write to files. However, filehandles can also be used in connection with ‘pipes’ which act just like pipes in Unix. So you can establish a filehandle which acts as a pipe that receives input from a Unix command (go back to the Unix lesson [U34](#) if you need a reminder).

The following code connects a filehandle to the output of the ls command and then read the output, one line at a time:

```
16. # filehandle '$in' will now receive output from the 'ls' command
17. open(my $in, "ls |") or die "Can't open pipe from ls command";
18. while (my $line = <$in>) {
19.     print "file: ", $line;
20. }
21. close($in);
```

If we reverse things and put the pipe *before* the command/script name, then we can even use open() to send commands to a program!

```
22. # the filehandle '$out' will now connect to the Unix wc command
23. open(my $out, "| wc") or die "Can't open pipe to wc command";
24. print $out "this sentence has 1 line, 10 words, and 51 letters\n";
25. close $out;
```

P25. Options processing

It is useful for your programs to have command-line options that allow them to behave in different ways. For example, `ls` lists the current directory, but if you want to see which files sorted by date, you type `ls -lt`. Your Perl programs can have this same behavior. There are two built-in modules for processing command line options, `Getopt::Std` and `Getopt::Long`.

Do you wonder what the `::` means in `Getopt::Std`? This is a scope divider. It's like a sub-folder. So `Getopt::Std` and `Getopt::Long` both exist inside a hierarchy with `Getopt` as the parent. It happens that there is a folder called `Getopt` and inside this are files called `Std.pm` and `Long.pm`. If you like hierarchy, you can make your libraries have this kind of structure, but it is not usually necessary.

Take a minute to view the documentation for `Getopt::Std` and `Getopt::Long`. You don't have to read them in depth, but just know that you can read the Perl documentation for most modules with a quick command-line:

```
$ perldoc Getopt::Std  
$ perldoc Getopt::Long
```

To use `Getopt::Std`, you must first define global variables called `$opt_something` where the *something* is a single letter. For example if you wanted a command-line option `-v` to indicate that the program should display its version number, you need a global variable called `$opt_v`. To define a global variable, you can use the `use vars` method or the `our` method (sort of like the `my` keyword except for global rather than lexical variables). Both syntaxes are displayed below.

You also have to tell `Getopt::Std` that you want to parse the command-line. You do this with the `getopts()` function. The syntax is a little strange. If the option takes arguments, you follow the letter with a colon. So, `getopts('x')` signals that `-x` takes no arguments while `getopts('x:')` signals that `-x` requires an argument. The example below shows how you can mix both kinds.

Try running the program below with a bunch of different options and see what happens. Note that the options are removed from the command-line. So `@ARGV` never contains the options.

```
1.  #!/usr/bin/perl
2.  # getopt.pl
3.  use strict; use warnings;
4.  use Getopt::Std;
5.  use vars qw($opt_h $opt_v);
6.  our $opt_p; # alternative to 'use vars'
7.  getopts('hvp:');
8.
9.  my $VERSION = "1.0"; # it's a good idea to version your programs
10.
11. my $usage =
12. usage: getopt.pl [options] <arguments...>
13. options:
14.     -h    help
15.     -v    version
16.     -p    <some parameter>
17. ";
18.
19. if ($opt_h) {
20.     print $usage; # it's common to provide a -h to give help
21.     exit;
22. }
23.
24. if ($opt_v) {
25.     print "version ", $VERSION, "\n";
26.     exit;
27. }
28.
29. if ($opt_p) {
30.     print "Parameter is: $opt_p\n"
31. }
32.
33. print "Other arguments were: @ARGV\n";
```

Line 11 does something with a variable assignment that we have not seen before. The \$usage variable is assigned the contents of several lines of text. The closing " that ends the assignment does not appear until line 17. This is an easier way of getting newlines into a variable.

Lines 21 and 26 introduce the `exit()` function. This terminates the program immediately without producing an error message (unlike the `die()` function). We use the `exit()` function at places when we deliberately want to stop the program.

P26. References and complex data structures

One of the reasons Perl is so powerful is that you can create complex data structures very easily. In this final section, we give a brief introduction to references, which are the foundation of complex data structures and other advanced programming concepts. Once you start to use references, you will find that they open up a whole new level of programming.

Multi-dimensional Arrays

So far, all of our arrays have been one-dimensional. But you can make them multi-dimensional with ease. If you assume the extra dimensions exist, Perl will create them for you.

```
my @matrix;
$matrix[0][0] = 1;
$matrix[0][1] = 5;
$matrix[1][0] = 3;
$matrix[1][1] = 2;
for (my $i = 0; $i <= 1; $i++) {
    for (my $j = 0; $j <= 1; $j++) {
        print "value at $i, $j is $matrix[$i][$j]\n";
    }
}
```

References

Up to now, we have never passed two arrays or hashes to a subroutine. Why? Because the arrays would get damaged by passing through `@_`. Consider the following code:

```
sub compare_two_arrays {
    my (@a, @b) = @_;
}
```

The intent is to fill up @a and @b from @_ . Unfortunately, in list context, Perl cannot determine the size of the arrays. So what happens is that @a gets all of the data and @b gets none. But surely, we want to be able to make comparisons of arrays. To do this, we must turn an array into a scalar value. This is done quite simply with the backslash \ operator. To dereference a particular element of the array, we use the arrow operator -> and square brackets.

```
my @array = qw(cat dog cow);
my $array_ref = \@array;
print $array_ref->[0], "\n"; # prints cat
```

We can also create references to hashes and dereference them with the arrow operator. Note that we use curly brackets here to show that the scalar value is a reference to a hash:

```
my %hash = (cat => 'meow', dog => 'woof', cow => 'moo');
my $hash_ref = \%hash;
print $hash_ref->{cat}, "\n"; # prints meow
```

To dereference the entire array or hash, rather than a specific element, we use the {} operator as follows:

```
print join("\t", @{$array_ref}), "\n";
foreach my $key (keys %{$hash_ref}) {
    print $key, "\t", $hash_ref->{$key}, "\n";
}
```

Anonymous Data

You can create a multi-dimensional array in a single statement:

```
my @matrix = ( [1, 5], [3, 2], );
```

Here, the arrays in square brackets are references to arrays. But these arrays have no names, so they are called anonymous arrays.

In a multi-dimensional array, the first dimension is a *reference* to other dimensions. References are scalar values, but they *point* to arrays, hashes, and some other types. To *dereference* a scalar, you use the `->` notation. In a multi-dimensional context, the `->` symbols are implied. Previously, we used `$matrix[0][0]`, but this can also be understood more explicitly as `$matrix[0]->[0]`. But use the former syntax, not the latter.

When constructing multi-dimensional structures, the various dimensions can be hashes or arrays, or a mixture. The dimensions need not even be the same size:

```
my @matrix = (
    [1, 2],
    {cat => 'meow', dog => 'woof', cow => 'moo'},
    [{hello => 'world'}, {foo => 'bar'}],
);
print $matrix[0][1], "\n";
print $matrix[1]{cat}, "\n";
print $matrix[2][1]{foo}, "\n";
```

Records

One of the most common places you will see a reference is a hash reference. These are used to store record-like data.

```
my @authors = (
    {first => 'Ian', last => 'Korf', middle => 'F'},
    {first => 'Keith', last => 'Bradnam', middle => 'R'},
);
foreach $author (@authors) {
    print $author->{last}, ", ", $author->{first}, " ", $author->{middle}, ".\n";
}
```

What next?

If you've come this far, you've done very well. You can now pick up a variety of Perl books and start to learn more advanced and specialized topics. As always you will learn Perl much more quickly if you have some real-world problems that you need to write a script for. This doesn't have to be work related, if you have any text files that contain data of some sort, then you can probably think of a Perl script to do something with that data. E.g. you could work out the average rating of each artist in your iTunes library by writing a script to parse the 'iTunes Music Library.xml' file that is produced by iTunes.

Sometimes the best way of improving your Perl is when you have to fix or improve someone else's script. Seeing how other people code will give you ideas and make you realize what works well and what doesn't. There is a lot of freely available Perl code on the web (just search Google for "Perl script to do x, y, and z") and you will often find that you can adapt from other people's code. But it usually is much more fun to write your own!

Troubleshooting guide

Introduction

The next few pages list many of the common error messages that you might see if you are having problems with your Perl script. They are broadly divided into three categories:

1. Errors that are caused before your Perl code is even evaluated
2. Errors in the code itself (most commonly, very simple syntax errors)
3. Other mistakes (sometimes achieved by great feats of stupidity)

If there is a problem with your script, you will sometimes see a lot of errors appear when you try to run it. It pays to try to understand these error messages. With time, you will become quicker at fixing errors, or at least knowing where to look first.

Many code editors are specifically designed for working with programming languages, and they can help you hear and see problems as you create them. E.g. they might beep to warn you if you have entered too many closing brackets or parentheses. They almost certainly will color code that appears between pairs of quotation marks, so you quickly see if you have typed one quotation mark too many.

How to troubleshoot

Programming languages like Perl have sophisticated, and therefore complicated, debugging tools. But for simple scripts, these tools can be overkill. Here is some simpler advice to how to go about fixing your scripts:

1. Stay calm and don't blame the computer. In nearly all cases, the computer is only ever doing what you have told it to do. Keeping a clear head will help you find the problem.
2. Check and re-check your code. Most errors are due to simple typos in your script, and sometimes you will be looking at the error without realizing that it is the error.
3. Start with the *first* error message that you see. Subsequent error messages often all stem from the first problem in your script. Fix one, and you may fix them all.
4. If you think a problem is due to an error on a single line of code, then you can comment out that line (by adding a # character to the start of the line). Then save and re-run your script to see if it now works. If it does, then you have confirmed which line contained the problem. Note that this is not appropriate for commenting out a single line of a block of

- code, e.g. the first line of an `if` statement.
5. Sometimes a program will partly work, but fail at some point within your code. Consider adding simple `print()` statements to work out where the program is failing.

Pre-Perl error messages

Permission denied

Do you have the permission to run your script, i.e. have you run `chmod` to add executable permissions? This won't affect scripts located on a USB flash drive, but in most 'real world' situations, you will always need to run the `chmod` command after creating your Perl script.

bad interpreter: No such file or directory

Most commonly caused by a typo in the first line of your script. The first line of a Perl script should let the Unix system know where it can find a copy of the Perl program that will understand your code. This will usually be `/usr/bin/perl`. If you miss the first forward slash, then the Unix system will try finding Perl (which is the interpreter of your code) in the wrong place, and hence things will fail.

command not found

You've either mistyped the name of the program or your program is not in a directory that the Unix system knows about (technically speaking the directory is not in your path). Make sure that scripts are kept in the `Code` directory (otherwise you will need to run them by using the `perl` command itself, e.g. `perl myscript.pl`).

Within-Perl error messages

Missing right curly or square bracket at script.pl line X , or... Unmatched right curly bracket at script.pl line X

Hopefully these two error messages are both very obvious. '[These are square brackets]' and '{these are curly brackets}'. Unless you are using them as text characters (e.g. within a `print` statement), then they always come in pairs. Make sure that yours are in pairs.

syntax error at script.pl line X, near YYY

Syntax errors are among the most frequent errors that you will see. On the plus side, they are usually very easy to fix. On the negative side, they can sometimes be very hard to spot as they frequently involve a single character that is either missing or surplus to requirements. Most commonly they might be because of:

- unmatched parentheses — like brackets, items that are in (parentheses) should always be a double act.
- missing semi-colon — If you start writing some code, then it has to end (at some point) with a semi-colon. The main exceptions to this rule are for the very first line of a script `#!/usr/bin/perl` or when a line ends in a closing curly bracket `}`. Also note that you can write one line of Perl code across several lines of your text editor, but this is still one line of code, and so needs one semi-colon.
- missing comma - Perl uses commas in many different ways, have you forgotten to include one in a place where Perl requires one?
- inventing new Perl commands and operators - if you write `if ($a === $b)`, then you have invented a new operator `==` which will cause a syntax error as Perl will have no idea what you mean.

Can't find string terminator '"' anywhere before EOF at script.pl line X

Did you make sure that you have pairs of quotation mark characters? If you have an odd number of single or double quotes characters, then you might see this error.

use of uninitialized variable in...

Your scripts will do many things with variables. You will add their values, calculate their lengths, and print their contents to the screen. But what if the variable doesn't actually contain any data? Maybe you were expecting to fill it with data from the command-line or from processing a file, but something went wrong? If you try doing something with a variable that contains no data, you will see this error.

Global symbol \$variable requires explicit package name at

You wouldn't happen to be using the strict package and not declaring a variable with `my` would you? If you definitely have included `use strict;` then maybe check that all your variable names are spelled correctly. You might have introduced a variable as `my $apple` but then later incorrectly referred to it as `$appple`.

Other errors

Program changes not saved

If you make changes to your program but don't save them, then those changes will not be applied when you run the script. Always check that the script you are running is saved before you run it. If you are using a graphical text editor on an Apple computer, then you will always see a black dot within the red 'close window' icon on the top left of a window when there are any unsaved changes.

Program that you are editing is not the same as program you are running.

Occasionally, you might make copies of your programs and your directory might end up with programs named things like script1.pl, script1b.pl, script2.pl, new_script2.pl. This is a bad habit to get into and you might find yourself editing one script but trying to run another script. You will become very frustrated when every change you make to your script has seemingly no effect.

Program runs with no errors but doesn't print any output

It might seem mysterious when your Perl program which you so carefully wrote, doesn't seem to do anything. It is therefore worth asking yourself the question 'did I ask it to do anything?'. More specifically, have you made sure your program is printing out any output. Making your program calculate the answer to the life, the universe, and everything is one thing...but if you don't print out the answer, then it will remain a mystery.

Table of common Perl error messages

Error message	Description/Solution
<i>Argument "xyz" isn't numeric...</i>	Perl is expecting a number, and you have given Perl something else, e.g. some text, or a variable containing text.
<i>Array found where operator expected...</i>	An operator (e.g. +, ==, >, eq.) is missing and an array name has been used instead
<i>bad interpreter: No such file or directory...</i>	Check the 1st line of your script <code>#!/usr/bin/perl</code> . You have probably made a typo?
<i>Bareword found where operator expected...</i>	Most likely due to a simple typo in a Perl operator. e.g. typing 'eqq' rather than 'eq'
<i>Can't find string terminator '"' anywhere before EOF...</i>	Probably a mismatched pair of quotation marks. These should come in pairs.
<i>Can't locate xyz.pm in ...</i>	You've added a 'use' statement, but the module name you are trying to use does not exist. Typo?
<i>command not found</i>	Possible typo when you typed the script name in the terminal. Or the script might not be in the Code directory.
<i>Global symbol \$variable requires explicit package name at...</i>	You probably forgot to include the <code>use strict</code>
<i>_Permission denied... _</i>	Have you run the chmod command to give your script executable permission?
<i>print () on</i>	

<i>unopened filehandle...</i>	If your script is printing output to a file, you have to first open a filehandle for the output file
<i>_Scalar found where operator expected...</i>	An operator (e.g. +, ==, >, eq.) is missing and a variable or array/hash element has been used instead
<i>Search pattern not terminated...</i>	When you use the matching operator =~, there should be a pair of forward slashes surrounding the search pattern
<i>String found where operator expected...</i>	An operator (e.g. +, ==, >, <=, eq, etc.) is missing, and some text has been added in it's place
<i>syntax error at...</i>	Often due to a missing semi-colon/comma, or other typo (e.g. typing 'iff' instead of 'if')
<i>use of uninitialized variable in...</i>	You are working with a variable (or array/hash element) that doesn't contain any data, even though it probably should. This is more common when the data is coming from a file or is specified on the command-line.

Version history

3.1.1 - 10/30/12 - Mostly small changes:

- Correction to Unix pipeline in [Project 4](#)
- New [First steps](#) section to clarify what should be done first with the downloaded files
- Removed part that wrongly stated that Factorial function will be used more than once in [Project 0](#)
- Fixed typo in [Project 1](#)
- Clarified example [P13.1](#)
- Clarified example [P14.3](#)

3.1.0 - 9/27/12 - Three new projects added. Updated information about book. Various small typos fixed.

3.0.0 - 3/5/12 - Major update to switch to using Markdown as the preferred underlying text format for this primer. This has also allowed us to tidy up some smaller typos which have been reported to us and make some of the styles that we use a little more consistent. Other changes:

- updated section on code editors
- changed text to use ‘we’ rather than ‘I’ (there are two authors!)
- added more hyperlinks to Perl functions
- added more hyperlinks between documents sections (where relevant)
- reworded many examples to make things clearer
- added Monty Python reference (a Python reference in Perl!)
- added small section to introduce newer style of filehandle in addition to older style
- added small section on the (preferred) 3-argument form of opening files
- several new snippets of example code to help explain regular expression metacharacters in a bit more detail
- Expanded the advice for [Project 4](#) and clarified the steps involved
- Expanded explanations regarding subroutines [P20](#) and lexical scope [P21](#).

2.3.7 - 2/21/10 - Some minor changes (thanks to Claudius Kerth for raising most of these):

- removed unnecessary square brackets from `tr` example of task [P6.12](#)

- fixed a link to a wrong section number in [Project 3](#)
- simplified code in task [P18.1](#)
- fixed bugs in tasks [P20.1] and [P22.3](#).

2.3.6 - 10/27/10 - Fixed some small typos, corrected some line numbers for a script and added one more clue in a section which seems to confuse lots of people (Task 6.12). Also added links to Fraise (the successor to Smultron) and explained that Smultron is only Mac OS 10.5 compatible whereas Fraise is only Mac OS 10.6 compatible.

2.3.5 - 5/24/10 - Fixed a single typo in exercise P20 which would prevent the script working properly.

2.3.4 - 11/13/09 - a couple of typo fixes and slight restructuring to transliteration routine

2.3.3 - 10/30/09 - Expanded on arrays and loops sections. More explanatory text is given with more examples.

2.3.2 - 10/16/09 - Added a new section on how to trouble-shoot problematic Perl scripts, with explanations of common error messages. Plus some more minor typo fixes.

2.3.1 - 10/9/09 - Minor typo fixes

2.3 - 9/29/09 - One new Perl task added to introduce the die function slightly earlier. Added new Unix task to learn about converting newline characters.

2.2.1 - 8/2/09 - Fixed incorrect numbering for list of projects in Project 5 section

2.2 - 7/28/09 - Big change in that all examples (apart from first few) now have ‘use strict’. Changed some examples to be more biologically relevant. Added more hyperlinks for Perl functions. Added graphical example of arrays. Expanded explanations in many examples, particularly the section on subroutines which gains many new examples.

2.1 - 7/22/09 - Added Preamble section to explain how to go about this course on a Windows machine. Added author bios. Changed directory structure for course files so that everything is contained within one parent directory (“Unix_and_Perl_course”). Broke several of the Unix sections into smaller sections

2.05 - 7/17/09 - Some minor typos fixed

2.04 - 7/16/09 - Fixed minor typos. Expanded section on variables, and offered advice on variable names. Simplified some print examples.

2.03 - 7/15/09 - Fixed minor typos. Expanded table of useful commands. Expanded explanation of tr operator, @ARGV, and escaping via backslash character. Fixed E.coli project example. Mentioned how to spot unsaved files in Mac editors. Moved tables inline.

2.02 - 7/14/09 - ‘use warnings’ reinstated to all scripts. Table of contents added. Hash bang line also included in all scripts. A few new sections added to Unix part of course, including a table of commonly used Unix commands. Lots of small of formatting changes to stop sections splitting over pages where possible.

2.01 - 7/13/09 - Miscellaneous typos fixed and text reworded to clarify.

2.00 - 7/12/09 - Revision based on feedback from course. Switched to PDF documentation rather than HTML. Smaller, more focused exercises.

1.00 - First taught course to grad students in UC Davis in Fall 2008

0.5 - Brief Unix/Perl training material written to help new students who join our lab