William Chen
wcchen@stanford.edu
Biochem218
Final Project
6/6/2002

A comparison of search algorithms in identifying all the proteins from the PROSITE protein database

Abstract

Commonly used methods for searching protein databases include the Smith-Waterman, the Gapped BLAST, and Ungapped BLAST algorithms. In order to evaluate their efficiency in finding the members of protein families, a Perl script was written to take all the protein families from the PROSITE protein database and perform a sequence similarity search using each of these three algorithms. The results for each search were then compared to each other to determine if there are any patterns of improved performance for one algorithm over the others. Analysis of this dataset of over 1500 protein families finds that the Smith-Waterman searches were the best overall and consistently performed better on the larger protein families, performing significantly better on families with greater than 300 members. However, no consistent pattern for the Ungapped BLAST or the Gapped BLAST searches was found.

Introduction

Upon sequencing a previously uncharacterized protein, one of the first methods of analysis commonly used is to run a similarity search for other proteins already deposited in the various databases available. High degrees of similarity are often due to common ancestry (homology) preserved over evolution (Shpaer et al., 1996). If homologous proteins are identified, this simple search can potentially yield great amounts of information about the newly sequenced protein due to the fact that homologous proteins share a common three-dimensional structure and also often share common active sites or binding domains (Pearson, 1997). In addition, homologous proteins may also share common functions due to evolutionary conservation (Pearson, 1997). Having this type of information can greatly aid a researcher, potentially reducing months of work to a single afternoon; however, it is key that the proteins identified are truly homologous and not spurious false hits.

Common protein similarity searches utilize the BLAST (Basic Local Alignment Search Tool) (Altschul et al., 1990) or Smith-Waterman (Smith and Waterman, 1981) algorithms. Many studies have been conducted to evaluate the performance of these algorithms in identifying sequence similarities (Shpaer et al. , 1996; Pearson, 1995). Key considerations in evaluation include the ability to identify between true homologues (sensitivity) and unrelated proteins (selectivity), and the time necessary to perform the search (speed).

Shpaer et al. (1996) found that protein searches using BLAST (BLASTP) have increased speed over Smith-Waterman, but do not always produce the most accurate possible results. Some of the lost accuracy can be attributed to the fact that the BLAST program often finds several alignments, that when combined, are statistically significant, but if any one of these alignments is missed, the overall result may be also be missed (Altschul et al., 1997). Since then, Altschul et al. (1997) has introduced Gapped BLAST, which allows for the generation of gapped alignments, making it necessary to initially find only one rather than all the ungapped alignments in order to produce a significant result. Although the Smith-Waterman search algorithm is more sensitive than the BLAST, it is extremely computationally demanding, making it difficult to implement without special-purpose computer hardware and software (Shpaer et al, 1996).

As more and more proteins are sequenced and characterized, most of them can be grouped into a limited number of families on the basis of similarities in their sequences. Proteins or protein domains belonging to a particular family generally share functional attributes and are derived from a common ancestor. One database containing listings of protein families and domains is PROSITE (http://www.expasy.ch/prosite/). PROSITE has used patterns and profile matrices to identify over 1000 protein families or domains.

With the continual development of more and larger protein families and domains, it is possible to examine how well the Gapped BLAST (GP), Ungapped BLAST (UG), and Smith-Waterman (SW) algorithms perform in identifying all the members of these families using a single starting protein for each family. The purpose of this study is to compare the performance of these three algorithms and determine if any one of the algorithms consistently outperforms the other two based on the protein family identifying pattern or type.

Methods

Perl Scripts

The first perl script (Appendix 1A) was written to identify all the protein families or domains listed within PROSITE (http://us.expasy.org/cgi-bin/prosite-list.pl) (Figure 1). The second perl script (Appendix 1B) was then used to generate a list of every family member associated with its identifying PROSITE cross-reference for each data entry. For example, the LIM domain signature and profile would have two data entries, each associated with a cross-reference (Figures 2 and 3).

The perl script then stored the amino acid sequence of the first true positive family member identified on the cross-reference site and performed a Gapped BLAST, Ungapped BLAST, and Smith-Waterman search using the Decypher server (http://decypher.stanford.edu/index_by_algo.htm). The default settings were used in all cases except to switch between Ungapped and Gapped BLAST, in the "Max Scores" and "Max Alignments" (see below), and "Filter Query" which was switched to "on" for SW to match the BLAST search default.

In order to evaluate the search results, a modification of the "missed@equivalence" point (Pearson, 1995) was calculated. Briefly, for a particular data entry, the true positives and false negatives together were used as the "gold standard" dataset. The total number of true positives and false negatives was entered in the "Max Scores" field. If a protein was found in either list, then, it was counted as a "hit", and the total number of "hits" was calculated for each search algorithm. In most cases, the number of "misses" was equivalent to the number of unrelated sequences found in the search. However, in some instances when using BLAST, the number of sequences located in the search was less than the "Max Scores" value. Therefore, in order to compare the three algorithms, a percent score was instead calculated based on (number of hits)/(total possible hits).

Analysis

For each data entry, the perl script output included the percent score for each search method and a ranking of how the methods compared to each other (Figure 4). After every protein family or domain was run through the perl script, the output was visually scanned for 1) cases where one method outperforms one or both other methods by a large percentage, 2) families which are extremely large (> 300 members), and 3) cases where the searches seemingly have failed.

Members of the third class could further be subdivided into 1) cases where the percentage was 0 for one or more the searches or 2) cases in which at least one member was found for each search (see Results). For the former, rerunning the perl script often returned different results (i.e. the Decypher server most likely timed out during the original search); however in a few instances, rerunning still produced no results. Upon further examination, these families proved to have amino acid sequences that were either too short to perform searches on or had highly repetitive sequences which were filtered out so that not even the original member was found.

For members of the second class (total = 33), each was run through the perl script a second time with the “Filter Query” option turned to “off”. Finally, each member of the second and third classes were looked up on the PROSITE site in order to examine their pattern and entry description.

Results

General findings

The total number of protein families and domains listed on the PROSITE site is 1136. Of these, 800 contain a single data entry while 336 contain multiple data entries. Overall, in 823 out of 1500 total data entries examined, all three search algorithms performed equally well (Table 1A). In searches where the algorithms found unequal numbers of family members, the SW algorithm performed the best in all combinations of the results, GP second best, and UG worst (Table 1B). In the majority of families examined, all three algorithms locate greater than 80% of the total “gold standard” proteins. Quite often, the hit percentage was 100% for all three algorithms when families had few members (less than 100). The largest observed family in which all three methods found all members was PS00657 (104 out of 104), which is the Fork Head Family. Therefore, at a superficial level, the algorithms are all performing quite well.

Class one

Class one families were instances where one algorithm outperformed one or both of the other algorithms by a large percentage. Consistent with earlier results by Pearson (1995) and Shpaer et al. (1996), each algorithm had certain protein families in which it performed better than the others.

Upon examining the specific families in which GP or UG performed the best, no clear pattern or family type could be identified that would explain why GP or UG performed the best. Both lists included families that were characterized by short amino acid patterns with and without gaps included in the signature patterns as well as families defined by a domain matrix. The only consistent characteristic was that the families all had less than 100 members. When GP performed the best, the largest difference in performance was with the Armadillo Repeat Family in which GP found 21 of 40 proteins versus 13 of 40 for both UG and SW. When UG was the best, the largest difference was with the Long Hematopoietin Receptor, Soluble Alpha Chains Family signature, where UG found 15 of 19, SW 8 of 19, and GP 11 of 19.

As with GP and UG, when SW performed the best, there was no discernible pattern or family type based on signature patterns and descriptions of the families. However, in contrast to GP and UG, SW performed the best not only in small families but also in large families as well. It was observed that as families began to contain more than 100 members, both GP and UG had more difficulty locating “gold standard” members while SW performance did not suffer as greatly. In fact, the larger the family, the more likely it was that SW performed better than UG or GP. Some examples of this contrast include PS01132, the Actin Family, where SW finds 254 of 255 (99%), while UG finds only 164 (64%) and GP 162 (64%), a greater than 30% drop in performance, and PS00030, the Eukaryotic RNA Recognition Motif Family where SW finds 213 of 297 (72%), GP 165 (56%), UG 145(49%), a greater than 20% drop.

Class two

When families had greater than 300 members SW overwhelmingly performed better than both GP and UG. In all 33 cases where families contained more than 300 members, SW outperformed both GP and UG by an average of 25% and 32%, respectively. Most strikingly in the two families with over 1000 members, PS50262, the G-protein Coupled Receptors Family (1209 members) and PS50011, the Protein Kinases Family (1330 members), SW finds over 900 members for both families, while UG finds only 456 and 503 and GP only 446 and 384. This represents a greater than 400 member difference between SW and UG or GP.

In order to determine if filtering the query had a large effect on the performance of these algorithms in the large families, all 33 families were rerun through the perl script with “filter query” set to “off” and compared to the original search. In some cases, turning off the filter option resulted in more hits, but in other cases, it resulted in less hits, reducing the number of hits by as many as 52 in one case. Additionally, for many of the families, one algorithm would perform better while the other two would perform worse or vice versa. Therefore, removing the filter query option did not seem to have any discernible consistent effect.

Class Three

Class three families included those which failed to produce a hit in one or more of the algorithms (discussed in Methods) and those in which all three algorithms hit only a small fraction of the total possible family members. For those in which only a small fraction of members were found (most often just 1), there were a number of possible explanations.

In some cases, the first member of the family matched more than one protein family profile such as CRI4_MAIZE which comes from PS50050, the TNFR/NGFR Family but also has a protein kinase domain. Therefore many of the highest scoring matches are to other protein kinases rather than to the other TNFR/NGFR family members.

In other cases the first member was the only representative from an organism in that family such as CWL1_SCHPO from PS50845, the Reticulon Family which is a protein from *Schizossaccharomyces pombe* while the other nine members of the family are from human, mouse or rat. Therefore, it is possible that although a pattern detectable by the PROSITE search was maintained through evolution, there is not enough sequence conservation to detect the other members by any of the three algorithms.

Still other cases were such that there may be other highly conserved members of the family which may not be inputted into PROSITE yet. These other members had higher scores than the “gold standard” members which brought the total hit percentage down. MGSA_BACSU from PS01335, Methylglyoxal Synthase Active Site Family is one such case.

Finally, not all cases could be deduced easily as to why the searches failed to produce more than a few hits. They could be combinations of reasons which are beyond the scope of this study.

Discussion and Conclusion

The goal of this study was to determine if there are any discernible patterns in the performance of the Smith-Waterman, Gapped BLAST, and Ungapped BLAST algorithms in finding all the members of the protein families from the PROSITE protein database. Based on the limited analysis performed, there did not seem to be any patterns associated with the cases in which GP or UG performed exceptionally compared to the other searches. Although the PROSITE search patterns used to define a particular family may contain numerous gaps, this did not seem to affect the performance of UG versus GP in finding family members.

However, SW was consistently the best when the families were large (over 100). In cases where there were over 300 members, SW was the always the best. Additionally SW was the best overall in finding the most family members. Based on these results, in order to find the most biologically significant hits with a given protein sequence, it is best to use the SW algorithm. It would also be interesting to know if there were any proteins found by GP or UG that were not found by SW and also whether a combination of the results by all three is much better than SW alone.

Turning on or off the “filter query” option had variable results when used on the large protein families. Although in some cases the algorithms made more matches, in others, the results were much worse. Therefore, it is unclear as to whether or not the “filter” option is beneficial when performing the searches.

Another consideration unaccounted for is the fact that the SW algorithm will report all matches up to the number entered in the “Max Alignments” option. GP and UG, on the other hand, will only report as many as have e-value scores less than the “Expectation” cutoff (default is 10). Many of the SW hits may have p-values of 1, leading to more false positives. A solution to this problem would be to use an unmodified missed@equivalence which would take into account this variability.

Further detailed analysis of the families and member proteins may prove to yield patterns not detected in this analysis and may warrant more study. Additionally, this search should also be repeated with a different starting protein to determine the consistency of these results.

These search algorithms are a good starting place when searching for homologous proteins; however as demonstrated by the class three protein families, even if a protein is a member of a family, the highest scoring hits may not be matches to other members of the family. It would be prudent, therefore, to also make use of the many other computational tools available, such as searching on PROSITE, BLOCKS, or PFAM for conserved domains.

Although beyond the scope of this study, additional considerations that should be taken into account include changing the gap opening and extension penalties, changing the scoring matrices used, or adjusting the expectation value (Altschul et al., 1994). Also, numerous other search algorithms based on SW or on BLAST have been proposed, such as PSI-BLAST (Altschul et al., 1997) BALLAST (Plewniak et al., 2000), and SALSA (Rognes and Seeberg, 1998). In addition, implementing methods such as generalized affine gap costs (Altschul, 1998) may improve search performance. If any information about the protein is known, such as membrane localization, there may also be specific algorithm modifications available (Hedman et al., 2001), which could improve detection of homologous proteins.

The overall goal in a study like this is the eventual improvement of the methods used in protein similarity searches so as to find the most biologically significant hits. To that end, any further analysis using the dataset generated from this study should be sure to keep this in mind.

PDOC00382	2	LIM domain signature and profile
PDOC00924	2	NF-kappa-B/Rel/dorsal family signature and profile
PDOC00302	2	MADS-box domain signature and profile
PDOC50126	2	S1 domain profiles
PDOC00972	3	T-box domain signatures and profile
PDOC00479	1	TEA domain signature
PDOC00624	1	Transcription factor TFIIB repeat signature
PDOC00303	1	Transcription factor TFIID repeat signature
PDOC00383	1	TFIIS zinc ribbon domain signature
PDOC00991	1	TSC-22 / dip / bun family signature

Figure 1: Example of PROSITE family entries
Column one indicates the family accession number, column two indicates how many data entries (patterns, rules and profiles/matrices) are described, and column three is a short description of the family or domain.

LIM domain signature and profile

PROSITE cross-reference(s)

PS00478; LIM_DOMAIN_1
PS50023; LIM_DOMAIN_2

Documentation

```
Recently [1,2] a number of proteins have  been  found  to  contain a
conserved cysteine-rich domain of about 60 amino-acid residues. These
proteins are:

***Rest of documentation omitted***
```

Description of pattern(s) and/or profile(s)

Consensus pattern	C-x(2)-C-x(15,21)-[FYWH]-H-x(2)-[CH]-x(2)-C-x(2)-C-x(3)- [LIVMF] [The 5 C's and the H bind zinc]
Sequences known to belong to this class detected by the pattern	ALL.
Other sequence(s) detected in SWISS-PROT	NONE.
Sequences known to belong to this class detected by the profile	ALL.
Other sequence(s) detected in SWISS-PROT	3.
Note	this documentation entry is linked to both signature patterns and a profile. As the profile is much more sensitive than the patterns, you should use it if you have access to the necessary software tools to do so.

Last update

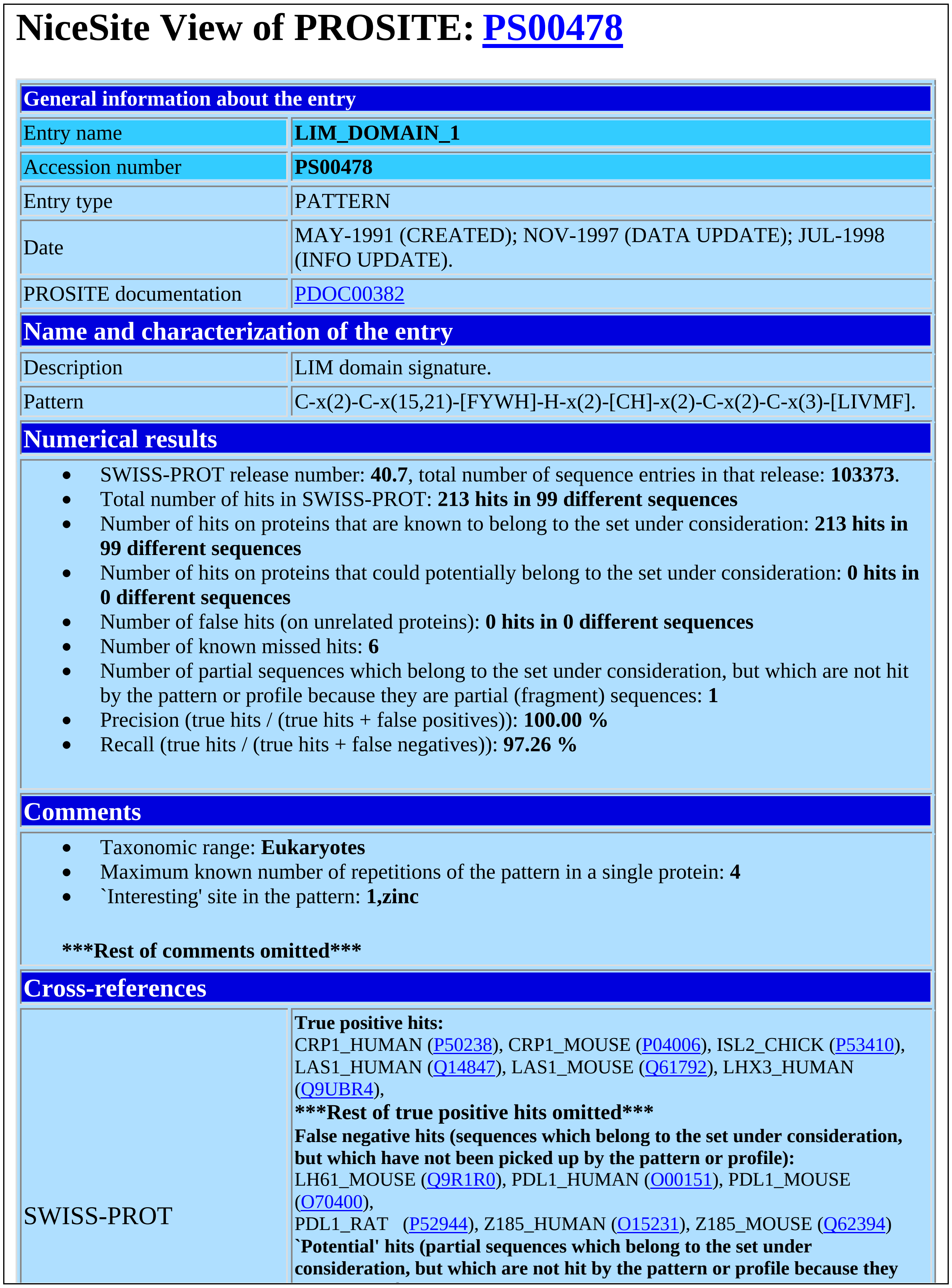
July 1999 / Text revised.

References

```
***References omitted***
```

Copyright

Figure 2: Example of single family entry
The LIM domain signature and profile has two data entries, and therefore two different cross-references are listed.



NiceSite View of PROSITE: PS00478

General information about the entry	
Entry name	**LIM_DOMAIN_1**
Accession number	**PS00478**
Entry type	PATTERN
Date	MAY-1991 (CREATED); NOV-1997 (DATA UPDATE); JUL-1998 (INFO UPDATE).
PROSITE documentation	PDOC00382

Name and characterization of the entry	
Description	LIM domain signature.
Pattern	C-x(2)-C-x(15,21)-[FYWH]-H-x(2)-[CH]-x(2)-C-x(2)-C-x(3)-[LIVMF].

Numerical results

- SWISS-PROT release number: **40.7**, total number of sequence entries in that release: **103373**.
- Total number of hits in SWISS-PROT: **213 hits in 99 different sequences**
- Number of hits on proteins that are known to belong to the set under consideration: **213 hits in 99 different sequences**
- Number of hits on proteins that could potentially belong to the set under consideration: **0 hits in 0 different sequences**
- Number of false hits (on unrelated proteins): **0 hits in 0 different sequences**
- Number of known missed hits: **6**
- Number of partial sequences which belong to the set under consideration, but which are not hit by the pattern or profile because they are partial (fragment) sequences: **1**
- Precision (true hits / (true hits + false positives)): **100.00 %**
- Recall (true hits / (true hits + false negatives)): **97.26 %**

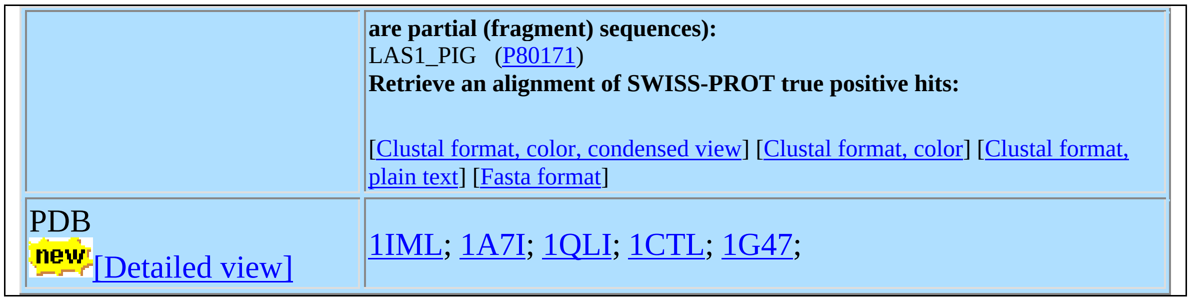
Comments

- Taxonomic range: **Eukaryotes**
- Maximum known number of repetitions of the pattern in a single protein: **4**
- `Interesting' site in the pattern: **1,zinc**

*****Rest of comments omitted*****

Cross-references

SWISS-PROT	**True positive hits:** CRP1_HUMAN (P50238), CRP1_MOUSE (P04006), ISL2_CHICK (P53410), LAS1_HUMAN (Q14847), LAS1_MOUSE (Q61792), LHX3_HUMAN (Q9UBR4), *****Rest of true positive hits omitted***** **False negative hits (sequences which belong to the set under consideration, but which have not been picked up by the pattern or profile):** LH61_MOUSE (Q9R1R0), PDL1_HUMAN (O00151), PDL1_MOUSE (O70400), PDL1_RAT (P52944), Z185_HUMAN (O15231), Z185_MOUSE (Q62394) **`Potential' hits (partial sequences which belong to the set under consideration, but which are not hit by the pattern or profile because they**



	are partial (fragment) sequences): LAS1_PIG (P80171) **Retrieve an alignment of SWISS-PROT true positive hits:** [Clustal format, color, condensed view] [Clustal format, color] [Clustal format, plain text] [Fasta format]
PDB new [Detailed view]	1IML; 1A7I; 1QLI; 1CTL; 1G47;

Figure 3: Example of single data entry
The first LIM domain signature cross-reference is shown with the pattern used to locate the family members. True positives and false negatives are listed at the bottom of the entry and make up the “gold standard” for members of the family.

```
PS00290
SW found: 328 out of 366                  percent: 0.896174863387978
Gapped BLAST found: 109 out of 366        percent: 0.297814207650273
Ungapped BLAST found: 120 out of 366      percent: 0.327868852459016
SW > UG > GP
```

Figure 4: Example of Perl output

The output from running the perl script on PS00290 shows that SW successfully finds 328 of 366 family members while GP only finds 109 and UG 120. In this case, SW is better than UG, which, in turn, is better than GP.

Table 1: Comparison of performance between Smith-Waterman(SW), Ungapped Blast(UG) and Gapped Blast(GP).

A		B	
SW > GP > UG	89	SW best alone	165
SW > UG > GP	76	GP best alone	75
GP > SW > UG	46	UG best alone	56
GP > UG > SW	29		
UG > GP > SW	28	SW best or equal to best	409
UG > SW > GP	28	GP best or equal to best	276
SW = GP > UG	121	UG best or equal to best	219
SW = GP = UG	823		
SW > GP = UG	67	SW worst alone	107
SW = UG > GP	56	GP worst alone	160
GP > SW = UG	30	UG worst alone	256
GP = UG > SW	50		
UG > GP = SW	57	SW worst or equal to worst	194
		GP worst or equal to worst	284
		UG worst or equal to worst	353

Appendix 1
Perl Scripts Written:

A. First Script

```
#!/bin/perl -w

#This simple script takes as input the source code to
#http://us.expasy.org/cgi-bin/prosite-list.pl and parses it to make a
#list of the form: <Accession num> <num of data entries> <description>
#Please note that my Perl is extremely limited, so code is not very
#written.

use strict;
use LWP::UserAgent;

#this is the source code saved as a .txt file
my $input = 'prosite_pro_fam.txt';
my $count = 0;
my $flag1 = 0;
my $HREF = 'HREF';

open (FILE, $input);

foreach(<FILE>)
{
    my @test = '';
    my @test2 = '';
    @test[0] = 'B';
    @test2[0] = '';
    $count += 1;
    if ($count == 52)
    {$flag1 = 1;}

    if ($flag1 == 1)
    {
      my @data = split ("<", $_);

      @test = split (" ", @data[1]);
      if (@test[0] =~ "A")
      {
          @test2 = split ("=", @test[1]);

          if (@test2[0] =~ "HREF")
          {

            my @data2 = split (">", @data[1]);
            my @data3 = split ("  ", @data[2]);
            print "@data2[1] @data3[1] @data3[2]";
            #$flag1 = 0;
      }}}



}
```

B. Second Script

```
#!/bin/perl –w

#This code takes as input an accession number followed by the number of
data #entries followed by a description in a list format. This list can
be generated #by running the preceding perl script. This being the most
complex Perl code
#that I’ve ever written, it’s a bit unwieldy.

use strict;
use LWP::UserAgent;

use HTTP::Request::Common qw(GET);
use HTTP::Request::Common qw(POST);

my %name_fam;
my %name_gene;
my %name_num;
my @keynames = ();
my $counter = 0;
my $getfamcounter = 0;

#Hashes to store data
my %SW_results;
my %Gapped_results;
my %Ungapped_results;
my $compare;
my $gappedalignment;

#counters for SW vs UG vs GP
my $c1 = 0;
my $c2 = 0;
my $c3 = 0;
my $c4 = 0;
my $c5 = 0;
my $c6 = 0;
my $c7 = 0;
my $c8 = 0;
my $c9 = 0;
my $c10 = 0;
my $c11 = 0;
my $c12 = 0;
my $c13 = 0;

#this is the url which post goes to.
my $url = 'http://decypher2.stanford.edu/cgi-win/CGI.exe';

#test query seq
my $seq = '';
my $numproteins = 1;

my %cgi;


#main
```

```
{
    my $file = 'profams.txt';
    my $count1 = 0;
    my $count2 = 0;
    my $count3 = 0;
    my $html;

    open (FILE, $file);
    foreach(<FILE>)
    {
      $count3 += 1;
      my @data = split (" ", $_);
      if ($data[1] > '1')
      {
          $count1 += 1;
          my @site = split (" ", $data[0]);
          #print $site[1];
          my $html_name = "http://us.expasy.org/cgi-bin/get-prosite-
raw.pl?$site[0]";
          $html = getHTML($html_name);

          getmult($html);

      }
      else
      {
          my @site = split (" ", $data[0]);
          #print $site[1];
          my $html_name = "http://us.expasy.org/cgi-bin/get-prosite-
raw.pl?$site[0]";
          $html = getHTML($html_name);
          my $next_site = parse_site($html);

          $html_name = "http://us.expasy.org/cgi-bin/get-prosite-
raw.pl?$next_site";
          #print $html_name, "\n";
          $html = getHTML($html_name);
          getfam($html);
          $count2 += 1;
      }
    }

    foreach(@keynames)
    {

      $numproteins = $name_num{$_};

      if ($numproteins > 1)
      {
          print $_, "\n";

          #print $name_gene{$_};
          $seq = $name_gene{$_};

          #print $name_fam{$_}, "\n";
          $compare = $name_fam{$_};
          $SW_results{$_} = runsmithwaterman();
          $gappedalignment = 'T';
          $Gapped_results{$_} = runBLASTP();
          $gappedalignment = 'F';
          $Ungapped_results{$_} = runBLASTP();

          my $swpercent = ($SW_results{$_}/$name_num{$_});
          print "SW found: $SW_results{$_} out of $name_num{$_}
\t\tpercent: $swpercent\n";
          my $gppercent = ($Gapped_results{$_}/$name_num{$_});
          print "Gapped BLAST found: $Gapped_results{$_} out of
$name_num{$_}\tpercent: $gppercent\n";
          my $ugpercent = ($Ungapped_results{$_}/$name_num{$_});
          print "Ungapped BLAST found: $Ungapped_results{$_} out of
$name_num{$_}\tpercent: $ugpercent\n";


          if (($swpercent > $gppercent) && ($gppercent > $ugpercent))
          {
            print "SW > GP > UG\n";
            $c1++;
          }
          if (($swpercent > $ugpercent) && ($gppercent < $ugpercent))
          {
            print "SW > UG > GP\n";
            $c2++;
          }
          if (($swpercent < $gppercent) && ($swpercent > $ugpercent))
          {
            print "GP > SW > UG\n";
            $c3++;
          }
          if (($ugpercent < $gppercent) && ($swpercent < $ugpercent))
          {
            print "GP > UG > SW\n";
            $c4++;
          }
          if (($ugpercent > $gppercent) && ($gppercent > $swpercent))
          {
            print "UG > GP > SW\n";
            $c5++;
          }
          if (($ugpercent > $swpercent) && ($gppercent < $swpercent))
          {
            print "UG > SW > GP\n";
            $c6++;
          }

          if (($swpercent == $gppercent) && ($gppercent > $ugpercent))
          {
            print "SW = GP > UG\n";
            $c7++;
          }
          if (($swpercent == $gppercent) && ($gppercent == $ugpercent))
          {
            print "SW = GP = UG\n";
            $c8++;
          }
          if (($swpercent > $gppercent) && ($gppercent == $ugpercent))
          {
            print "SW > GP = UG\n";
            $c9++;
          }
          if (($swpercent == $ugpercent) && ($gppercent < $ugpercent))
          {
            print "SW = UG > GP\n";
            $c10++;
          }
          if (($swpercent < $gppercent) && ($swpercent == $ugpercent))
          {
            print "GP > SW = UG\n";
            $c11++;
          }
          if (($ugpercent == $gppercent) && ($ugpercent > $swpercent))
          {
            print "GP = UG > SW\n";
            $c12++;
          }
          if (($ugpercent > $gppercent) && ($gppercent == $swpercent))
          {
            print "UG > GP = SW\n";
            $c13++;
          }
      }
    }
    print "SW > GP > UG : $c1\n";
    print "SW > UG > GP : $c2\n";
    print "GP > SW > UG : $c3\n";
    print "GP > UG > SW : $c4\n";
    print "UG > GP > SW : $c5\n";
    print "UG > SW > GP : $c6\n";
    print "SW = GP > UG : $c7\n";
    print "SW = GP = UG : $c8\n";
    print "SW > GP = UG : $c9\n";
    print "SW = UG > GP : $c10\n";
    print "GP > SW = UG : $c11\n";
    print "GP = UG > SW : $c12\n";
    print "UG > GP = SW : $c13\n";
    print "number of singles, multiples, total: $count2, $count1,
$count3\n";
}

#This subroutine parses the html site when there is only a single
family entry
sub parse_site
{
    my $text = shift();
    my $flag1 = 0;
    my @lines = split("\n", $text);
    my @value;
    my @value2;

    foreach(@lines)
    {
```

```
    if ($flag1 == 1)
    {
      #print $_;
      @value = split('{', $_);
      #print $value[1];
      @value2 = split(";", $value[1]);
      #print $value2[0];

    }
    $flag1++;
    }
    return $value2[0];
}

#This subroutine sets the values for running a Smith-Waterman search
sub runsmithwaterman
{
%cgi =(

       Algorithm => 'SW',
       QueryType => 'AA',
       TargetType => 'AA',
       QuerySearch => '1',
       OutputFormat => 'MAXSCORE PERCENTAGE',
       Comment => 'Smith-Waterman Similarity Search',

       ReplyVia => 'BROWSER',
       ReplyFormat => 'TEXT',
       AATargetSet => 'SWISSPROT',
       QueryFormat => 'FASTA/PEARSON',

       QueryText => $seq,
       QueryFilter => 'T',
       MatchCharacter => '1',
       MaxScores => $numproteins,
       AAMatrix => 'BLOSUM62.MAA',
       MaxAlignments => '0',
       OpenPenalty => '12',
       Threshold => '1',
       ExtendPenalty => '2',
       AlignmentThreshold => '20',
       FieldRecord => 'FIELDRECORD',
       command => 'start');

my $temp = runseq();
return parse_SW($temp);

}

#This subroutines sets the values for running a BLAST search
sub runBLASTP
{
%cgi =(
       Algorithm => 'BLASTP',
       QueryType => 'AA',
       Comment => 'NCBI BLASTP Similarity Search',
       ReplyVia => 'BROWSER',
       ReplyFormat => 'TEXT',
       BlastTargetSet => 'SWISSPROT',
       QueryFormat => 'FASTA/PEARSON',
       QueryText => $seq,
       QueryFilter => 'T',
       AAMatrix => 'BLOSUM62.MAA',
       MaxScores => $numproteins,
       Expectation => '10',
       MaxAlignments => '0',
       ExtensionThreshold => '0 (use default)',
       GappedAlignment => $gappedalignment,
       WordSize => '0 (use default)',
       OpenPenalty => '0 (use default)',
       ExtendPenalty => '0 (use default)',
       Threshold => '1',
       ShowGI => 'T',
       FieldRecord => 'FIELDRECORD',

       command => 'start');


my $temp = runseq();
return parse_blast($temp);

}

#subroutine that runs the post and returns the search results
sub runseq
{
my %cgiArgs = %cgi;

#this runs POST and returns an html
my $html = &doCGI($url, \%cgiArgs);


}

#subroutine to get family with multiple data entries
sub getmult {
    my $text = shift();
    my @lines = split(/\n/, $text);
    my $flag1 = 0;
    my $flag2 = 0;

    foreach(@lines)
    {
      if ($flag1 == 1)
      {
      my @value = split('{', $_);
      if ($value[1] =~ /BEGIN/)
      {$flag1 = 0;}

      if ($flag1 == 1)
      {
          my @value2 = split(";", $value[1]);

          #print $value2[0];
          my $html_name = "http://us.expasy.org/cgi-bin/get-prosite-
raw.pl?$value2[0]";
          my $html = getHTML($html_name);
          #print $html, "here!!!getmult\n";
          getfam($html);
      }
    }
    if ($flag2 == 0)
    {
      $flag2 = 1;
      $flag1 = 1;
    }}
}

#Parses output from a blast search using the decypher server
sub parse_blast
{
    my $text = shift();
    my $flag1 = 0;
    my @lines = split(/\n/, $text);
    my $flag2 = 0;
    my $bigcount = 0;
    my $flag3 = 0;
    my $flag4 = 0;

    foreach(@lines)
    {
      if ($_ =~ /Database: SwissProt Release 40.14/)
          {
            $flag1 = 0;
          }
      if ($flag1 == 1)
      {
          my @records = split (/\s+/, $_);
          #print $records[1], "\n";

           #This checks to see if the protein found in the search is a
member
           #of the gold standard list.

          if($records[1])
          {
            if ($compare =~ /$records[1]/i)
          {
            #print $records[1];
            ++$bigcount;
          }
          $flag2++;
          if ($flag2 == $numproteins)
          {
            $flag1 = 0;
          }
      }
      }

      if ($flag3 == 1)
      {
          $flag1 = 1;
          $flag3 = 0;
      }
      if ($_ =~ /^Sequences/)
      {
          $flag3 = 1;
      }
    }
    return $bigcount;
}

#Parses the output using a smith-waterman search on the decypher server
sub parse_SW
{
    my $text = shift();
    my $flag1 = 0;
    my @lines = split(/\n/, $text);
    my $flag2 = 0;
    my $bigcount = 0;
    my $flag3 = 0;

    foreach(@lines)
    {
      if ($flag1 == 1)
      {
          #print $_, "\n";
          my @records = split (/\s+/, $_);
          #print $records[8], "\n";

           #This checks to see if the protein found in the search is a
member
           #of the gold standard list.
           if ($compare =~ /$records[8]/i)
          {
            #print $records[8];
            ++$bigcount;
          }
          $flag2++;
          if ($flag2 == $numproteins)
          {
            $flag1 = 0;
          }
      }
      if ($flag3 == 1)
      {
          $flag1 = 1;
          $flag3 = 0;
      }
      if ($_ =~ /DESCRIPTION/)
      {

          $flag3 = 1;
      }
    }
    return $bigcount;
}
```

```
#subroutine to get all the protein members of the family and store as a
list
sub getfam {

    $getfamcounter++;
    #print "getfam!";
    my $proname = '';
    my $text = shift();
    my @words;
    my @words2;
    my @words3;
    my $family_mem = '';
    my $flag1 = 0;
    my $seq;
    my $flag2 = 0;

    my @lines = split(/\n/, $text);
    #print @lines;

    foreach(@lines){
      if ($_ =~ /^AC /)
      {
          #print $_;
          @words = split (/\s+/, $_);
          #print @words;
          $proname = $words[1];
          chop($proname);
          #print $proname, "\n";
          $flag1 = 1;
          $name_gene{$proname} = '';
          #print "@keynames keynames\n";
          push (@keynames, $proname);
          $counter = 0;
      }
           if ($_ =~ /DR /)
           {
             @words = split ("; ", $_);
             @words2 = (0,0,0,0,0,0,0,0,0);
             @words2 = split ("   ", $words[0]);
             #print $words[2], "\n";
             @words3 = (0,0,0,0,0,0,0,0,0);
             if ($words[2])
             {
                 @words3 = split (";", $words[2]);
             }
             #print @words3, "\n";
             $family_mem .= truepos($words2[1]);
             $family_mem .= truepos($words[1]);
             $family_mem .= truepos($words3[0]);
             #print "$words2[1]\n$words[1]\n$words3[0]\n";

             if ($flag1 == 1)
             {
                 my @words4 = split ", ", $words2[1];
                 my $html_site = $words4[0];
                 my $html = getHTML("http://us.expasy.org/cgi-bin/get-
sprot-fasta?$html_site");
```

```
                 $flag1 = 0;

                 my @fasta_seq = split ('.', $html);
                 $name_gene{$proname} = $html;
             }
           }
    }
    $name_fam{$proname} = $family_mem;
    $name_num{$proname} = $counter;
#print $name_fam{$proname};
#print $name_gene{$proname};
}

sub truepos {
my $text2 = shift();
my @words4;
if ($text2)
{
    @words4 = split (", ", $text2);
}
my $value = '';
if($words4[2])
{
    if ($words4[2] =~ /T/)
    {
      $value = $words4[0];
      $counter++
    }

    if ($words4[2] =~ /N/)
    {
      $value = $words4[0];
      $counter++
    }
}
return $value;
}

my $ua = undef;
sub getHTML {
    my $url = shift();
    unless ($ua) {
      $ua = new LWP::UserAgent;
      $ua->agent("QuoteBot/0.1 " . $ua->agent);
    }
    # Create a request the Object Oriented way:
    my $req = HTTP::Request->new(GET => $url);


    # Pass request to the user agent and get a response back
    my $res = $ua->request($req);

    # Check the outcome of the response
    if ($res->is_success) {
      return $res->content();
    } else {
      print STDERR"Bad luck this time\n" . $res->as_string();
    }}
sub doCGI {

#METHOD="POST" ENCTYPE="multipart/form-data"
    my $url = shift();

    my %args = %{shift()};

    unless ($ua) {

      $ua = new LWP::UserAgent;

      $ua->agent("BlatBot/0.1 " . $ua->agent);
    }

    # Create a request
#    my $req = new HTTP::Request POST => $url;
#    $req->content_type('multipart/form-data');
#    $req->content('match=www&errors=0');

    #print "here in post";
    my $req = POST $url, [ %args ];
    # Pass request to the user agent and get a response back
    my $res = $ua->request($req);

    # Check the outcome of the response
    if ($res->is_success) {
      return $res->content();
    } else {
      print STDERR"Bad luck this time\n" . $res->as_string();
    }
}

#END
```

References

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and D.J. Lipman. (1990). Basic Local Alignment Search Tool.

Altschul, S.F., Boguski, M.S., Gish, W., and J.C. Wooton. (1994) Issues in searching molecular sequence databases. *Nature Genetics.* 6:119-129.

Altschul, S.F., Madden, T.L., Schaffer, A.A., Zhang, J., Zhang, Z., Miller, W., and D.J. Lipman. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nuc. Acid Res.* 25(17):3389-3402.

Altschul, S.F. (1998). Generalized affine Gap costs for protein sequence alignment. *PROTEINS: Structure, Function and Genetics.* 32:88-96.

Hedman, M., Deloof, H., Von Heijne, G., and A.. Elofsson. (2001). Improved detection of homologous membrane proteins by inclusion of information from topology predictions. *Protein Sci.* 11:652-658.

Pearson, W. R. (1995). Comparison of methods for searching protein sequence databases. *Protein Sci* 4(6): 1145-60.

Pearson, W. R. (1996). Effective Protein Sequence Comparison. *Methods in Enzymology*. R. Doolittle. New York, Academic Press. 266: 227-258.

Pearson, W. R. (1997). Identifying distantly related protein sequences. *Comput Appl Biosci* 13(4): 325-32.

Plewniak, F., Thompson, J.D., and O. Poch. (2000). Ballast: blast post-processing based on locally conserved elements. *Bioinformatics.* 16(9):750-759.

Rognes, T. and E. Seeberg. (1998). SALSA: improved protein database searching by a new algorithm for assembly of sequence fragments into gapped alignments. *Bioinformatics.* 14(10):839-845.

Shpaer, E. G. (1997). GeneAssist. Smith-Waterman and other database similarity searches and identification of motifs. *Methods Mol Biol* 70: 173-87.

Shpaer, E. G., M. Robinson, D. Yee, et al. (1996). Sensitivity and selectivity in protein similarity searches: a comparison of Smith-Waterman in hardware to BLAST and FASTA. *Genomics* 38(2): 179-91.

Smith, T.F. and M.S. Waterman. (1981). Identification of common molecular subsequences. *J. Mol. Biol.* 147:195-197.