Biological Sequence Analysis

CSEP 521: Applied Algorithms – Final Project

Archie Russell (0638782), Jason Hogg (0641054)

Introduction

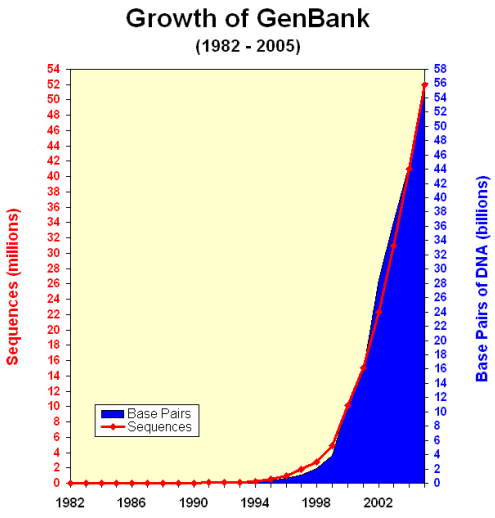
Background

The schematic for every living organism is stored in long molecules known as *chromosomes* made of a substance known as *DNA* (**d**eoxyribo**n**ucleic **a**cid. Each cell in an organism has a complete copy of its DNA, also known as its *genome*which is conveniently modeled as a *sequence* of symbols (alternately referred to as *nucleotides* or *bases*) in the DNA alphabet {A,C,T,G}. In humans, and most mammals, this sequence is about 3 billion bases long.

Through a process known as *protein synthesis*, instructions in our DNA, known as *genes*, are interpreted by machinery in our bodies and transformed first into intermediate molecules called *RNA*, and then into structures called *proteins*, which are the primary actors in living systems. These molecules can also be modeled as sequences of symbols.

These genes determine core characteristics of the development of an organism, as well as its day-to-day operations. For example, the Hox[1] gene family determines where limbs and other body parts will grow in a developing fetus, and defects in the BRCA1 gene are implicated in breast cancer.

The rate of data acquisition has been immense. As of 2006, GenBank, the national public repository for sequence information, had accumulated over 130 billion bases of DNA and RNA sequence, a 200-fold increase over the 1996 total[2]. Over 10,000 species have at least one sequence in GenBank, and 50 species have at least 100,000 sequences[3]. Complete genome sequences, each roughly 3-gigabases in size, are available for human, mouse, rat, dog, cat, cow, chicken, elephant, chimpanzee, as well as many non-mammals. Technology continues to make *sequencing* (the process of determining sequences) cheaper; the first complete sequencing of the human genome cost $2.7 billion[4], achieving the same for $10,000 is now on the horizon[5].



[1] Wikipedia article on Hox genes: http://en.wikipedia.org/wiki/Hox
[2] Statistics from http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html and http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=Nucleotide
[3] dbEST summary: http://www.ncbi.nlm.nih.gov/dbEST/dbEST_summary.html
[4] Human Genome Project FAQ http://www.genome.gov/11006943
[5] The Archon X-prize: http://genomics.xprize.org/ (this is not an outlandish goal)

Sequence Similarity – Why does it matter?

One of the most common computational operations on sequences is a *sequence similarity* search. A typical search compares one *query* sequence to a larger *database* of sequences and tries to find *alignments* between the query and database that reflect similarities between the two. These searches are valuable to researchers because of the nature of evolution, and the nature of scientific research. All known species, whether yeast, puffer-fish, mice, or humans appear to be related to each other (see Figure 1[6], but it is more convenient to do scientific experiments on yeast or mice than on humans. Researchers may be able to determine a gene in mouse which is responsible for a condition such as obesity; a sequence similarity search helps pinpoint the analogues of that gene in humans. Similarity searches are also used within a single species; portions of a gene, called *domains,* are often duplicated among several genes. Finally, similarity searches allow researchers to infer which parts of our DNA are important. Relevant portions of our genome are less likely to have changed over time than less-important regions. A similarity search that detects regions in common between evolutionarily distant species has probably also detected very important genomic regions.

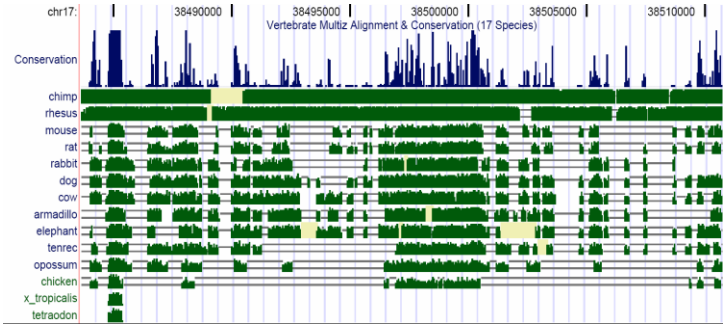


Figure 1: Similarity between human BRCA1 gene region and other species. Green regions are similar.

In order to perform these kinds of sequence analysis several factors must be considered. These include how to score individual matches across sequences, whether to perform global or local searches, what type of algorithm to use and finally how to evaluate results to determine the statistical significance of an alignment score.

We provide a high-level introduction to these processes with the goal being to simply to provide context for our detailed evaluation of the Smith-Waterman and BLAST algorithms. A complete description of each of these processes can be found in papers written by their authors[8,11].

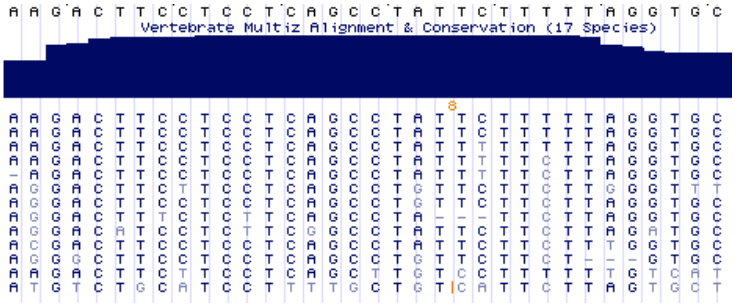


Figure 2a: Zoomed-in view of highly similar area from above, each row of ACTG corresponds to one species.

Scoring model

When comparing sequences we are looking for evidence that portions of a sequence may be related even though mutations may have occurred between the sequences. DNA can change in one of two ways. The first type of change is called a *substitution*, whereby a base of one type is substituted for another type. The second type of change is based on *gap*, whereby an insertion or deletion of a base is made to the original sequence.

---

[6] Kent, W.J., Sugnet, C. W., Furey, T. S., Roskin, K.M., Pringle, T. H., Zahler, A. M., and Haussler, D. The Human Genome Browser at UCSC. *Genome Research* **12**(6), 996-1006 (2002).

[7] http://en.wikipedia.org/wiki/BRCA1

[8] Smith, T.F. and Waterman, M.S. 1981, Identification of common molecular subsequences, *Jounal of Molecular Biology*, 147:195-197

The most fundamental scoring mechanism for evaluating *substitutions* uses *substitution matrices*. A substitution matrix for DNA sequences establishes a score for each pair of bases encountered in two input sequences. The score is positive if the bases are identical and usually negative if they are different. Various models exist for establishing substitution matrices based on the type of data being analyzed. For our example, we'll use a subset of the default scoring matrix, illustrated in Table 1.

Different approaches also exist for penalizing *gaps* in sequences. The first approach known as a linear score, simply applies a linear penalty of d based on the length of the gap. A second approach known as an affine score can be used to reduce the penalty burden where long insertions or deletions may occur.

Alignment algorithms

Alignment algorithms are used to determine the optimal alignment of a pair of sequences based on a particular scoring mechanism. Fundamentally, two different alignment problems exist. Global Alignment finds the best alignment from start to end of both sequences (with provision for gaps). Local Alignment is used to find subsequences of a sequence that have the best alignment

The Needleman-Wunsch algorithm is a dynamic algorithm that can be used to analyze global alignments, and was covered in class. The Smith-Waterman algorithm modifies the Needleman-Wunsch algorithm to allow it to search for local alignments. Local Alignment often works better for sequence similarity searches, because over the eons, stretches of DNA have been chopped up and re-arranged in ways that don't make for good global alignments. Although these dynamic algorithms will find optimal matches, they are computationally expensive, especially for large datasets. As such, heuristic algorithms such as BLAST have also been developed to reduce the computational burden on evaluations. We will discuss Smith-Waterman and BLAST in more detail in the rest of the paper.

Significance of scores

Having found an optimal alignment it is important to determine whether the alignment is statistically significant or whether the alignment is no more likely than we'd expect by random chance and uninteresting to a user. Two models exist for performing this analysis – a Bayesian approach and a statistical approach. Discussion of these algorithms is out of scope for this paper, however the importance for evaluating significance of scores is included simply to complete the discussion on the process by which analysis is performed.

Smith-Waterman

Motivation

The Smith-Waterman algorithm is a dynamic algorithm that is used to find local alignments of a subsequence within a larger sequence. Dynamic algorithms are guaranteed to find the optimal scoring alignment but can be computationally resource intensive. Smith-Waterman is based on the Needleman-Wunsch[9] algorithm which is used for obtaining optimal *global* alignment across pairs of sequences that may include gaps.

Design of the algorithm

As with all dynamic-programming algorithms Smith-Waterman establishes an optimal overall solution using optimal solutions to smaller sub-problems – or in this case optimal alignments on smaller subsequences. The algorithm establishes an *i * j* matrix designated *F*, where *i* and *j* are the widths of the respective subsequence and sequence. As we will describe shortly, the completed matrix F can be searched for optimal alignments.

[9] Needleman S.B. and Wunsch, C.D. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* 48:443-453

For Smith-Waterman we start by initializing the following cells of *F*:

- F(0,0) = 0
- F(x, 0) = 0, where 0 <= x < i
- F(0, y) = 0 , where 0 <= y < j

The values for the remaining cells are calculated using the following recursive algorithm:

$$F(i,j) = max \begin{cases} 0, \\ F(i-1,j-1) + s(x_i, y_i), \\ F(i-1,j) - d, \\ F(i,j-1) - d. \end{cases}$$

The score $s(x_i, y_i)$ is found using a substitution matrix as mentioned briefly earlier. A substitution matrix establishes a log-odds ratio providing a score for the alignment between two bases across sequences. A better alignment will have a higher score than a lower alignment.

The value for *d* is based on the approach used for gap penalization. Two approaches exist. The first approach is to use a linear score in which a penalty *g* is simply multiplied by the size of the gap *d*.

$$\gamma(g) = -gd$$

A second approach is based on an affine score which reduces the penalty burden where long insertions or deletions may occur in one of the sequences.

$$\gamma(g) = -d - (g-1)e$$

After we have evaluated each of the four equations for each cell *F(i, j)* we take the maximum value and assign that to *F(i, j)*. Where a cell contains a non-zero record we also include a pointer to the cell that represents the sub-problem (either {(i-1, j-1) (i-1, j) (i, j-1)} that served as the basis for this problem. This pointer is used to support back tracking which is used as the basis for establishing the optimal sub-sequence.

Once the matrix *F* is complete we then search the entire matrix *F(i, j)* for the highest value. This marks the start of our optimal alignment. At this point we must back track through the matrix following the pointers established earlier - until we find a zero score which marks the end of the optimal sequence.

Complexity Analysis

The Smith-Waterman algorithm requires the establishment of a matrix of size n * m, where n and m are the length of the query and target sequences. Each cell within F requires the comparison for maximum value of 4 values which are calculated in constant time. After the m*n matrix is populated, Smith-Waterman searches the matrix looking for the highest value and then constructs the optimal alignment. As such the algorithm requires O(nm) time and O(nm) memory. Some of the literature makes the assumption that n and m are roughly the same, simplifying the complexity to $O(n^2)$. This is reasonable for most comparisons of protein sequences, but this isn't always appropriate; the typical size of a gene is orders of magnitude smaller than the genome. For example the BRCA1[7] gene is about 80,000 bases long, much shorter than the 3-gigabase human genome.

Example

In this example we will show how Smith-Waterman would search for the best local alignment between the sequences GACTAC and CTA.

We will use a subset of the default scoring matrix that is used by the FASTA algorithm package (weizmann). The subset focuses only on the bases {A, C, G, T} (the FASTA package also includes "incomplete" symbols in its scoring matrix, for instance an R corresponds to either A or G[10]). . FASTA includes multiple programs, including SSEARCH which is an implementation of the Smith-Waterman package. For the sake of simplicity, we will also assume a linear gap penalizing value of g=1.

	A	C	G	T
A	5	-4	-4	-4
C	-4	5	-4	-4
G	-4	-4	5	-4
T	-4	-4	-4	5

Table 1: Scoring matrix used in our example

Table 2 - F(i, j) values

F(i, j)	0	1 **(C)**	2 **(T)**	3 **(A)**
0	0	0	0	0
1 **(G)**	0	0	0	0
2 **(A)**	**0**	0	0	5
3 **(C)**	0	**5**	0	4 (3,2)
4 **(T)**	0	**4 (1,3)**	**10 (1,4)**	9 (2,4)
5 **(A)**	0	2 (1,4)	9 (2,4)	**15 (2,4)**
6 **(C)**	0	5	7 (2,5)	14 (3,5)

Values F(i,j) are included inside Table 2. Cell references are included where F(I,j) is based on the value from another cell. The values in bold/red indicate the values found after locating the maximum score, 15 at cell (3,5). Arrows illustrate back-tracking and the optimal local alignment of the sequence CTA within the sequence GACTAC.

BLAST

Motivation

While Smith-Waterman produces optimal results, its O(MN) running time can be exorbitant. BLAST, an acronym for **B**asic **L**ocal **A**lignment **S**earch **T**ool, was built with this in mind[11]. BLAST uses a *heuristic* approach. It will find most of the results that an optimal approach finds, but in an order of magnitude less time. However, this comes at a cost; BLAST will miss a small fraction of the results found by optimal algorithms. [12]

Algorithm Design

The BLAST approach is to find short, but very high-similarity "seed" matches between a query and database sequence, following this with an extension of these matches into less similar regions, until it finds a maximal segment pair (MSP), which BLAST defines as "the highest scoring pair of idneticcal length segments chosen from

[10]Nomenclature for Incompletely Specified Bases in Nucleic Acid Sequences http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html#302

[11] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. 1990. Basic local alignment search tool. *Journal of Moleculary Biology* 215:403-410

[12] E.g. a comparison that scientists might be interested in is comparing a single mRNA sequence about 10,000 base-pairs in length to the complete collection of "EST" sequences of about 20 million members, each 500 bases in length. Smith-Waterman on these inputs could result in something approaching 10^{13} matrix cell comparisons. A *single* search like this could take hours or days to complete

two sequences". The key insight behind the design of BLAST is that most *statistically significant* hits -- hits that a user might be interested in -- will have at least one high-quality short "seed" match.

Breaking query sequence into "words"

A typical[13] BLAST run first divides a DNA input sequence contiguous "words" of length *w* (12 has been empirically determined as a good value for *w*). An *n* letter sequence will be broken into *n* - *w* + 1 words:

```
AATTGGACTGATTAAGGT....
```

is broken into the following words:

```
AATTGGACTGAT
 ATTGGACTGATT
  TTGGACTGATTA
   TGGACTGATTAA
    (etc)
```

With *w*=12, and 4 symbols in the DNA alphabet, we can map each of the possible 4^{12} (16,777,216) words to an integer. This integer can be used as an index into a table with pointers into the query sequence where the query sequence contains the words. In practice, for a given query many of these 12-mers won't occur at all; so the table is actually much smaller.[14]

Scanning the database for hits

Once the table of query words is assembled, BLAST scans its database *linearly* for matches to these query words. Compression of the DNA alphabet to two bits per symbol is straightforward (A=00, C=01, etc.). Using this scheme, the BLAST database represents eight bases using only two bytes of memory. With w=12, each hit must contain at least one eight-letter hit aligned to a byte boundary. Comparison to words on boundaries can usually be performed much faster than non-aligned comparisons which require additional shift operations, so BLAST only compares database sequences aligned to byte boundaries to the query words.

BLAST's authors also describe a finite-state machine approach to database scanning, which we've skipped for simplicity.[15]

Filtering uninformative words

Some words appear in so many regions that they add little information but can trigger BLAST to begin many slow extension steps that don't produce interesting results. An example is the "repeat" sequence ATATATAT which appears tens of thousands of times throughout the genome. The program that prepares sequence databases for BLAST tabulates the frequency of 8-letter words, and creates a list of "uninformative" words that are used to filter portions of the query sequences.

Hit extension and scoring

For each "hit" BLAST finds, between the query and database, BLAST uses a dynamic programming approach to extend the hit in both directions. The extension terminates when reaching a score that falls more than some distance below the best score among all shorter extensions. This extension process can be slow, especially if many hits are found, but has the advantage of allowing "gaps" in the alignments. Based on the length, number of

---

[13] This varies for protein sequence comparisons. Proteins are broken into (*n*-*w*) words (typically, w=4). Each of these then balloon into the ~50 w-letter words that, if found in the database, would match the query word with score better than some threshold.

[14] The authors don't describe *how;* a simple technique like taking the mod of the integer will suffice.

[15] BLAST apparently uses the 2nd of these, but the authors don't describe it in any significant detail. They reference Hopcroft and Ullman, '79, *Introduction to Automata Theory, Languages and Computation* p42-45

mismatches, and number and size of gaps in the final alignment, BLAST assigns an "expect" value to the hit, expressing how many similar scoring hits BLAST would expect to find with similar sized inputs by random chance.

Complexity

We were unable to find published time-complexities for BLAST. With a moderate sized query sequence, scanning a database of size n happens in O(n) time, but each hit then triggers a potentially expensive extension phase. Because BLAST is intended to improve results *in practice*, looking at running-time from real world results (below) may be more relevant than worst-case style analysis.

Comparing Smith-Waterman and BLAST

Speed

A primary goal in the design of BLAST is performance, and it succeeds with flying colors. For typical inputs, BLAST is *an order of magnitude* faster than Smith-Waterman[17] (see Table 3 below). This increase in performance is especially important when searching large bodies of data. For example, searching the 3-billion base human genome for the 80-thousand base BRCA1 gene that we mentioned earlier with Smith-Waterman would build a matrix with over 200 trillion cells.

Table 3: Library Search Times[16]

Computer	BLASTP	Smith Waterman
DEC Alpha 2100	0.5 min	10.1 min
Sun Sparc10	1.6 min	55.7 min

Times given for a search using the OPSD_HUMAN protein as query and SWISS-PROT release 31 (43,470 sequences) as the database.

Quality of Results

BLAST and Smith-Waterman results, when compared to a respected "gold standard", are roughly similar. BLAST found 379 of the proteins acknowledged by scientists to be related to the protein OPSD_HUMAN, while Smith-Waterman found 394. From a user's point of view, this tradeoff is a no-brainer– BLAST speeds up their work 20-fold without much degradation of results[17]

Table 4: Algorithm Seach Sensitivity

Protein	BLASTP	Smith-Waterman
OPSD_HUMAN	379	394
GTB1_MOUSE	66	63

(Numbers of known family members with expectation < 2.0)

BLAST Drawbacks

Because it requires exact matches over a certain length, BLAST typically misses some results. *Most* researchers don't mind the difference, but some consider these differences quite seriously. When similarities are very weak, such as between very distantly related species, BLAST may miss a larger portion of results. BLAST is much faster than Smith-Waterman, but still slower than a researcher would like. Though BLAST finds alignments with short gaps, it terminates when gaps get large. Because of a phenomenon called "splicing", long gaps are actually very common, causing BLAST to generate many small results rather than long "stitched together" results.

---

[17] Pearson W.R. 1996, Effective protein sequence comparison. *Methods in Enzymology* Vol 266: 277-258 More detailed studies have also been performed showing similar trends.

Further work

Since BLAST's release in 1990, sequence similarity work has continued. One path has been hardware acceleration (or exploitation). Another has been development of specialized algorithms that are faster or better than BLAST for specific problems.

Hardware acceleration/exploitation

Multiple companies have tried to improve sequence similarity performance with either custom hardware or making better use of underutilized hardware in existing computers.

Custom hardware

Paracel, a company that also sold string-matching software to the NSA, formerly sold a custom, field-programmable-gate-array (FPGA) based machine called the GeneMatcher. Paracel claimed Smith-Waterman searches on a GeneMatcher could run hundreds of times faster than BLAST runs on standard hardware. TimeLogic sells a BLAST acceleration board called the DeCypher compatible with SPARC-based systems from Sun MicroSystems. TimeLogic claims speed-up as much as 1500x over BLAST on the same hardware without the DeCypher board.[18]

A Paracel GeneMatcher cost ~$200,000 and required a $50,000 annual service contract. The company was purchased for $283 million in March 2000. Dollar-per-performance these solutions are cheaper than clusters of x86 machines, but they can essentially solve only one problem, limiting their usefulness.

GPUs, SIMD instructions

Since the mid 1990's, most consumer CPUs have been equipped with SIMD instructions (with names like MMX, SSE, Altivec), which perform multiple similar operations simultaneously. Usually these instructions must be called explicitly in C-code or assembler. The SSEARCH implementation of Smith-Waterman incorporates these instructions, claiming speedups of 6x-20x[19]

Multiple labs have explored using GPUs (graphic processing chips) to speed up Smith-Waterman. Speed increases have been claimed from 2-10x, depending on implementation and query size[20][21]

Parallelization

BLAST is an "embarrassingly parallel" algorithm. For the database-scanning phase, both the database and query sequence can be divided into independent sub-problems (hit-extension and scoring are a little more intricate). Newer versions of BLAST accept a command-line parameter specifying the number of processors to use.

Often, BLAST is run across many query sequences; this sort of workload is broken down and run on clusters of commodity x86 hardware very naturally.

New algorithm: BLAT

A notable sequence alignment program that emerged in 2002 is called BLAT[22] (an acronym for **B**LAST-**L**ike **A**lignment **T**ool). BLAST is can align any sort of biological sequence (genomic DNA, mRNA, protein) to a database of any other sort of biological sequence. In contrast, BLAT is designed to use only genomic DNA as a database. Where BLAST indexes its query sequence and scans linearly through its database, BLAT indexes its *database* and scans linearly through its *query sequence*. BLAT stores a compressed version of its complete target genome in

---

[18] TimeLogic benchmarks: http://www.timelogic.com/benchmark_blast.html

[19] Farrar M (2007). "Striped Smith–Waterman speeds database searches six times over other SIMD implementations". *Bioinformatics* **23**: 156-161.

[20] Weiguo Liu, Bertil Schmidt, Gerrit Voss, Andre Schroder, and Wolfgang Muller-Wittig, 2006, Bio-Sequence Database Scanning on a GPU

[21] The Joint Genome Institute has also explored this capability.

[22] Kent, W. James BLAT---The BLAST-Like Alignment Tool *Genome Research* 2002 12: 656-664

main memory; this would have been impractical when BLAST was implemented in 1990. Because startup time can be 10-15 minutes, BLAT typically runs as a "server" process. BLAT also approaches gaps differently, overcoming BLAST's weakness in dealing with long gaps in alignments caused by splicing.

BLAT's performance is much better than BLAST's:

	Time in seconds (doesn't include BLAT database initialization)	% of well-studied genes covered by alignments
BLAT	50	80.8%
BLAST (WU-TBLASTX)	3700	81.7%

Table 5: Aligning 1000 mouse DNA sequences to human chromosome 22

Conclusion

Sequence similarity is a very important problem for biologists. Smith-Waterman and BLAST are two classic algorithms for addressing the problem. Smith-Waterman is slower, but more precise, while BLAST uses a much faster heuristic approach. Other approaches to the problem include hardware acceleration and special-purpose algorithms.