Improvement of 2-Hit Algorithm for Basic Local Alignment Search Tool

Deepak Garg
Lecturer, CSED, TIET, Patiala
Deepakgarg@ieee.org

Dr. S.C.Saxena
Director, TIET, Patiala
director@tiet.ac.in

Abstract

Basic Local Alignment Search tool is a popular tool used for determining the patterns in genomic sequences. As the data is increasing exponentially, the need for advanced and complex algorithms for improving the accuracy, speed and sensitivity of pattern discovery tools in bioinformatics is also increasing.

The detailed analysis of the various parameters that take part in any sequence analysis is done in the paper. Threshold, dropoff score, sequence length, database length, lambda, wordsize, seedword and expect are some of the important ones that has been analysed in detail. .

Currently the initialization of the word matches in a pairwise sequence alignemnt works either on single hit or 2-Hit algorithm. Instead if we use 3-hit or n-hit in general then the results improve in general & improve dramatically for some specific species & sequences.

The BLAST algorithm extends the hit until that hit lies within an alignment with score sufficient to be reported until the running alignment's score has dropped more than X below the maximum score yet attained. This extension step is computationally quite costly and typically accounts for more than 90% of BLAST's execution time. It is therefore desirable to reduce the number of extensions performed. The Three- hit algorithm further increases the speed and sensitivity of the BLAST. So this approach can be extended further to N-Hit algorithm. But criteria for the value for N must be decided. This criterion would decide up to which value the N can be extended.

1. Introduction

In the plethora of tools available for data mining in bioinformatics, BLAST was chosen due to its unmatched speed, sensitivity and popularity. Though the performance of BLAST was best in its class of tools but still there was scope of improvement in it. In order to do so the working of BLAST and its variants were understood. Moreover, a lot of parameters, on which the speed and sensitivity of BLAST depends, were analyzed. It needs to be understood here, that the amount of research that has gone into the BLAST, is tremendous. Many people have put in years of efforts to formulate the core of BLAST. So it was painful to find the further scope & enhancements available in this field. In this paper an attempt has been made to give suggestions regarding the parameters and working of BLAST to improve its performance. There are various parameters that are having contextual relation with the areas other then the algorithm design and theory of computer science; the analysis of these parameters was limited from the viewpoint of a computer engineer. That's why the improvements are limited to those parameters.

2. Important Parameters

(i) T, the threshold parameter

T is referred to as the neighborhood word score threshold (Altschul et al., 1990). It is the minimum score that a word pair in the segment pair should have. Actually we can adjust the value of T to control the size of the neighborhood and

therefore the number of word hits in the search space. The lower value of T increases the chance that a segment pair with a score of at least S will contain a word pair with a score of at least T. Thus, a small value for T increases the number of hits. But this in turn increases the execution time of the algorithm because there will be more words generated by the query sequence and therefore more hits. On the other hand, higher values of T progressively reduces word hits and reduce the search space [15]. So the proper value of T depends on the balance between speed and sensitivity. It also depends on the values in the scoring matrix.

If the value for T is not chosen carefully, though -- *i.e.*, if T is set just a little bit too low -- a combinatorial explosion in neighborhood words will soon lead to the depletion of all available memory. Even if the neighborhood word list *does* fit in memory, however, its sheer size may produce an adverse effect on speed, due to the consequent loss of processor cache efficiency.

(ii) W, word size

Word size is roughly the minimal length of an identical match an alignment must contain if it is to be found by the algorithm. It controls the number of word hits. The query sequence and every database sequence is split up into every possible "word" of a selected size. The default word size is 11 bp for DNA (it must be >=7).

(iii) X, drop off

This value provides a cutoff threshold for the extension algorithm tree exploration. When the score of a given branch drops below the current best score minus the X dropoff, the exploration of this branch stops. This variable represents the recent alignment history [16]. Specifically, it represents how much the score is allowed to drop off since the last maximum.

A very large value of X doesn't increase the score and requires more computation. It is generally a good idea to use a large value, which reduces the risk of premature termination and is better way to increase speed than with the seeding parameters. However, W,T and 2-hit are better for controlling speed than X.

X not only depends on the substitution scores, but also gap initiation and extension costs. We general need to adjust this parameter in following two situations:

- If we align sequences that are nearly identical and we want to prevent the extensions from straying into nonidentical sequences, we can set the various X values very low. If we try to align very distant sequences and have already adjusted W, T and the scoring matrix to allow additional sensitivity, it makes sense to also increase the various X values.

(iv) λ, lambda

λ, is a matrix specific constant required to convert a raw score to normalized score. Raw score can be a misleading quantity because scaling factors are arbitrary. A *normalized score, corresponding* to the original load score, is therefore a more useful measure. Lambda is approximately the inverse of the original scaling factor, but its value may be slightly different due to integer rounding errors. When calculating target frequencies from multiple alignments, the sum of all target frequencies naturally sums to 1.

$$\Sigma\Sigma\, q_{ij} = 1 \qquad \ldots\ldots\ldots(1)$$

The score of two amino acids is the log-odds ratio of the observed and expected frequencies. The same equation is presented in Equation, but the lod score is replaced by the product of lambda and the raw score.

$$\lambda S_{ij} = \log_e (q_{ij} / p_i\, p_j) \qquad \ldots\ldots\ldots(2)$$

Equation (1)rearranges Equation (2) to solve for pair-wise frequency.

$q_{ij} = p_i p_j e^{\lambda S_{ij}}$ ……(3)

From Equation 3,we can see that a pair-wise frequency (q_{ij}) is implied from individual amino acid frequencies (p_i and p_j)and a normalized score (λS_{ij}).The key to solving for lambda is to provide the individual amino acid frequencies (p_i and p_j)and find a value for lambda where the sum of the implied target frequencies equals one. The formulation is given in Equation 4.

$\Sigma\Sigma\, q_{ij} = \Sigma\Sigma\, p_i p_j e^{\lambda S_{ij}} = 1$ ……(4)

Normally, once lambda is estimated, it is used to calculate the Expect of every HSP in the BLAST report. Unfortunately, the residue frequencies of some proteins deviate widely from the residue frequencies used to construct the original scoring matrix. Recently, some versions of PSI-BLAST and BLASTP have therefore begun to use the query and subject sequence amino acid compositions to calculate a *composition based lambda* [18] **.**These "hit-specific "lambdas have been shown to improve BLAST sensitivity, so this approach may see wider use in the near future. Lambda is also used in calculating the Expect by using the equation $E = kmne^{-\lambda S}$. Here Lambda may be thought of as the expected increase in reliability of an alignment associated with a unit increase in alignment score. Reliability in this case is expressed in units of information, such as bits or nats, with one nat being equivalent to 1/log(2) (roughly 1.44) bits.

(v) k, Adjustment

A small adjustment (*k)* takes into account the fact that optimal local alignment scores for alignments that start at different places in the two sequences may be highly correlated. For example, a high-scoring alignment starting at residues 1,1 implies a pretty high alignment score for an alignment starting at residues 2,2 as well.

3. Number of hit words

The central idea of the BLAST algorithm is that a statistically significant alignment is likely to contain a high-scoring pair of aligned words. BLAST first scans the database for *words* (typically of length three for proteins) that score at least *T* when aligned with some word within the query sequence. Any aligned word pair satisfying this condition is called a *hit*. The second step of the algorithm checks whether each hit lies within an alignment with score sufficient to be reported. This is done by extending a hit in both directions until the running alignment's score has dropped more than *X* below the maximum score yet attained. This extension step is computationally quite costly; with the *T* and *X* parameters necessary to attain reasonable sensitivity to weak alignments, the extension step typically accounts for >90% of BLAST's execution time. It is therefore desirable to reduce the number of extensions performed. In order to achieve this, the Two-hit algorithm was proposed.

Two-hit algorithm was based on the observation that an HSP of interest is much longer than a single word pair, and may therefore entail multiple hits on the same diagonal and within a relatively short distance of one another. (The *diagonal* of a hit involving words starting at positions ($x1$, $x2$) of the database and query sequences may be defined as $x1 - x2$. The *distance* between two hits on the same diagonal is the difference between their first coordinates.) This signature may be used to locate HSPs more efficiently. Specifically, a window length *A* is chosen, and it invokes an extension only when two non-overlapping hits are found within distance *A* of one another on the same diagonal. Any hit that overlaps the most recent one is ignored. Efficient execution requires an array to record, for each diagonal, the first

coordinate of the most recent hit found [16]. Since database sequences are scanned sequentially, this coordinate always increases for successive hits.

Because we require two hits rather than one to invoke an extension, the threshold parameter *T* must be lowered to retain comparable sensitivity. The effect is that many more single hits are found, but only a small fraction has an associated second hit on the same diagonal that triggers an extension. The great majority of hits may be dismissed after the minor calculation of looking up, for the appropriate diagonal, the coordinate of the most recent hit, checking whether it is within distance *A* of the current hit's coordinate, and finally replacing the old with the new coordinate. Empirically, the computation saved by requiring fewer extensions more than offsets the extra computation required to process the larger number of hits.

The Two-hit algorithm's overall performance speed was found to be three times faster than the original One-hit algorithm. The reason for it was that it saves a lot of computation time in extension, as it has to extend very few word hits, which lie on the same diagonal within a particular distance. The number of word hits produced by Two-hit algorithm more than the number of hits produced by the One-hit algorithm and this increased hits increases the computation time to process it. But still the fewer extensions to be made offset this extra computation time required in processing the larger number of hits. The following Fig illustrates the same fact.

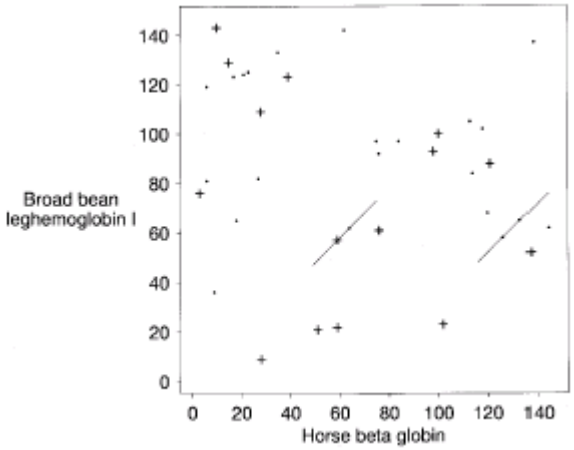

Figure 1 Impact of Two-hit method on the no of hits and the Extension

In the above figure BLAST comparison of broad bean leghemoglobin I (87) (SWISS-PROT accession no. P02232) and horse b-globin (88) (SWISS-PROT accession no. P02062) is done [16]. One-hit algoritm produces the 15 hits with score at least 13 and these are indicated by plus signs whereas Two-hit algoritm produces an additional 22 non-overlapping hits with score at least 11 and these are indicated by dots. Of these 37 hits, only the two indicated pairs are on the same diagonal and within distance 40 of one another. Thus the two-hit heuristic with $T = 11$ triggers only two extensions, in place of the 15 extensions invoked by the one-hit heuristic with $T = 13$ which saves a lot of computation time.

Not only it impacts the speed but also the sensitivity of the search. The probability of missing an HSP is reduced by Two-hit algorithm as the value of threshold T is set to lower value in this case. The following Fig illustrates this fact.

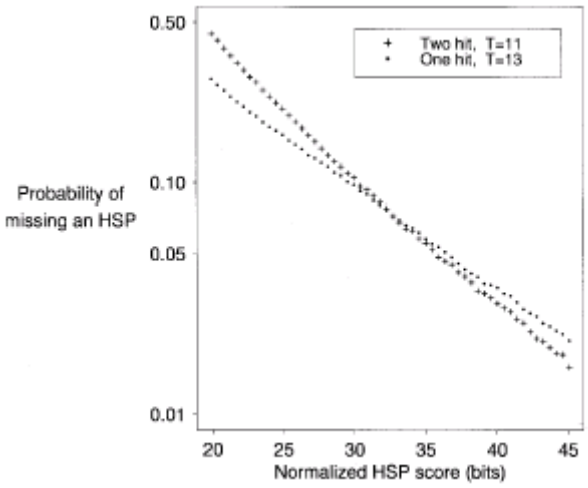

Figure 2 Sensitivity of the two-hit & one-hit heuristics as a function of HSP score

In the above figure normalized scores of Robinson and Robinson (20), 100 000 model amino acid were generated and then the probabilities of missing an HSP using the two-hit heuristic with $T = 11$, and the one-hit heuristic with $T = 13$, were plotted as a function of normalized HSP score [16]. The two-hit method is more sensitive for HSPs with score at least 33 bits .

The disadvantage with this approach is that the number of hits produced could be very large for a large query and database. So in that case the main memory can run out of space. Another disadvantage is that we can miss weak similarities if either they don't produce two word hits or the threshold value is not set to a low value.

4. Filter

Low-complexity regions, such as proline- or glycine-rich regions or acidic or basic regions, can yield *tremendous* numbers of spurious matches between sequences that have no other similarity between them. The statistics break down when such decidedly non-random sequences appear; furthermore, search times may be needlessly increased. To avoid spurious matching and make the statistics more robust, low-complexity regions can be filtered from the query sequence. Filtering eliminates statistically significant but biologically uninteresting reports from the blast output (e.g., hits against common acidic-, basic- or proline-rich regions), leaving the more biologically interesting regions of the query sequence available for specific matching against database sequences. Filtering is only applied to the query sequence (or its translation products), not to database sequences. Filtering should not be expected to always yield an effect. Furthermore, in some cases, sequences are masked in their entirety, indicating that the statistical significance of any matches reported against the unfiltered query sequence should be suspect.

Sometimes we need to mask the human repeats (LINE's and SINE's). It is especially useful for human sequences that may contain these repeats.

We extend the thought behind Two-hit algorithm to Three-hit algorithm. According to the Three-hit algorithm we choose a window length *A*, and it invokes an extension only when three non-overlapping hits are found on the same diagonal and the difference between each of them is *A*. Any hit that overlaps the most recent one is ignored. Efficient

execution requires an array to record, for each diagonal, the first coordinate of the most recent hit found. Since database sequences are scanned sequentially, this coordinate always increases for successive hits.

Because we require three hits rather than one or two to invoke an extension, the threshold parameter *T* must be lowered to retain comparable sensitivity. The effect is that many more single hits are found, but only a small fraction has an associated second hit on the same diagonal that triggers an extension. And this fraction is further reduced because a very few have associated third hit. The great majority of hits may be dismissed after the minor calculation of looking up, for the appropriate diagonal, the coordinate of the most recent hit, checking whether it is within distance *A* of the current hit's coordinate, and finally replacing the old with the new coordinate. Empirically, the computation saved by requiring fewer extensions more than offsets the extra computation required to process the larger number of hits. The steps for proposed Three-hit algorithm are as follows:

Step 1: Choose an appropriate value for window length A.

Step2. Lower the value of threshold T in order to yield more hits.

Step3. Seek triplets of non-overlapping hits found with distance A of one another on the same diagonal

Step 4. Invoke (ungapped) extension to determine if hits lie within a statistically significant alignment with query.

Step 5. Extend until alignment score has dropped more than or equal to X below max score yet attained.

The Three-hit algorithm's overall performance speed was found to be two times faster than the Two-hit algorithm. The reason for it was that it saves a lot of computation time in extension, as it has to extend very few word hits, which lie on the same diagonal within a particular distance. The number of word hits produced by Three-hit are more than the number of hits produced by the Two-hit algorithm and this increased hits increases the computation time to process it. But still the fewer extensions to be made offset this extra computation time required in processing the larger number of hits.

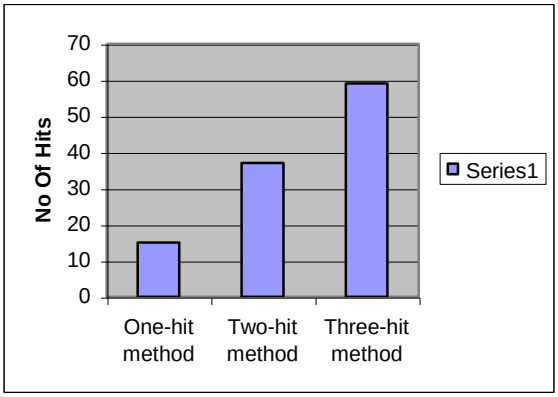

Figure 3 Impact of Different methods on No of Hits

In the above figure BLAST comparison of broad bean leghemoglobin I (87) (SWISS-PROT accession no. P02232) and horse b-globin (88) (SWISS-PROT accession no. P02062) is done. One-hit algorithm produces the 15 hits with score at least 13 and whereas Two-hit algorithm produces an additional 22 non-overlapping hits with score at least 11 and finally Three-hit algorithm produces 18 additional non-overlapping hits.

The probability of missing an HSP is further reduced by Three-hit algorithm as the value of threshold T is further lowered in this case as compared to Two-hit algorithm. The disadvantage with this approach is that the number of hits produced could be very large for a large query and database. So in that case the main memory can run out of

space. Another disadvantage is that we can miss weak similarities if either they don't produce three word hits or the threshold value is not set to a low value. Moreover, if the sequences being compared are not very similar, the Three-hit algorithm is theoretically disadvantaged at finding short regions of similarity, such as individual protein domains and short coding exons in vertebrate genome sequence, particularly if T is not lowered to compensate for the use of the Three-hit algorithm. The Three- hit algorithm further increases the speed and sensitivity of the BLAST. So this approach can be extended further to N-Hit algorithm. But criteria for the value for N must be decided. This criterion would decide up to which value the N can be extended.

References

1.Jason, Bruce, Dennis, "Pattern Discovery in Bimolecular Data", Oxford University Press, New York 1999.

2. Jean Michel Claverie and Cedric Notredame " Bioinformatics A beginner's Guide". Wiley Publishing, Inc. 2003.

3. Jiawei Han, Micheline Kamber amd Simon Fraser University "Data Mining Concepts and Techniques" Morgan Kaufmann Publishers, USA 2001

4. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy "Advances in Knowledge Discovery and Data Mining." AAAI/MIT Press, 1996.

5. J. Han and M. Kamber "Data Mining: Concepts and Techniques". Morgan Kaufmann, 2000.

6. T. Imielinski and H. Mannila "A database perspective on knowledge discovery. Communications of ACM", 39:58-64, 1996.

7. G. Piatetsky-Shapiro, U. Fayyad, and P. Smith "Data mining to knowledge discovery: An overview". In U.M. Fayyad, et al. (eds.), Advances in Knowledge Discovery and Data Mining, 1-35. AAAI/MIT Press, 1996.

8. G. Piatetsky-Shapiro and W. J. " Frawley Knowledge Discovery in Databases." AAAI/MIT Press, 1991.

9. Jagadish et al., "Special Issue on Data Reduction Techniques. Bulletin of the Technical Committee on Data Engineering, 20(4)", December 1997

10. Dan E. Krane, Michael L. Raymer " Fundamental concepts of Bioinformatics" Pearson Education, 2003

11. Scott Markel, Darryl Leon " Sequence Analysis In a Nutshell" O'reilly 2003.

12. Kent, W. James "BLAT- The BLAST- like Alignment Tool" Genome Research 12 (4):656-664" 2002.

13.Higgins, D.J. Thompson " ClustalW: Improving thesensitivity of progressive multiple sequence alignment through sequence weighting, position-specific

14. Gilbert, D. G. "ReadSeq version 2, an improved biosequence conversion tool " Bionet.Software(Aug), 1999.

15 Stephen F. Altschul, warner Gish, Webb Miller " Basic Local Alignment Tool" Mol Biol 215 , 403-410, 1990.

16 Stephen F. Altschul, Thomas L. Madden, Alejandro A. Shäffer 1, and David J. Lipman "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs" 3389–3402 Nucleic Acids Research, Vol. 25, No. 17, 1997.

17.Zheng Zhang, Alejandro A. Schäffer 1 , Webb Miller, Thomas L. Madden 2 , David J. Lipman 2, and Stephen F. Altschul 2, * "Protein sequence similarity searches using patterns as seeds" Nucleic Acids Research, Vol. 26, No. 1, 1998.

18.Stephen F. Altscul, Ralf Bundschuh, Rolf Olsen and Tenrence Ha "The estimation for local alignment score distributions" Nucleic Acids Research, Vol. 29, No. 2, 2001.

19Anna R. Panchenko “Finding weak similarities between proteins by sequence profile comparison” Nucleic Acids Research, 2003, Vol. 31, No. 2, 2002

20.Blast-help group, NCBI User Service “BLAST Program Selection Guide” 2004