ID 462 - 215

DASH: A New High Speed Genomic Sequence Search and Alignment Tool

PAUL GARDNER-STEPHEN AND GREG KNOWLES
Embedded Systems Laboratory
School of Informatics & Engineering,
Flinders University
GPO BOX 2100, Adelaide 5001
AUSTRALIA
gardners@infoeng.flinders.edu.au gknowles@infoeng.flinders.edu.au

Abstract: In this paper we introduce several features of our first generation diagonal aggregating search heuristic system, Dash[17], which results in order of magnitude speed improvement for nucleotide searches when compared to NCBI-Blast 2.2.6[1,7]. Heuristic algorithms such as Blast and FastA[8] are indispensable for searching large genomic databases. Not surprisingly the significant contributor to search time for such algorithms is the dynamic programming evaluations. Indeed NCBI-Blast typically spends around 76% of its time budget in this area. Improving the efficiency of dynamic programming activities provides an opportunity to significantly reduce search times and help offset the effects of the continuing exponential growth in database sizes. Features which contribute to the speed of Dash include almost complete mitigation of the dominant dynamic programming activities involved in a typical genomic search, efficient sequence comparison and scoring, optimisation of index size through sub-entropy encoding of index data, and by preventing repeated evaluation of dynamic programming search space.

Key-Words: - Bio-Informatics, Sequence Alignment, Blast, Genome, FastA

1 Introduction

Like Cafe[2] and Flash[3], Dash uses an index of the database to facilitate fast retrieval. In this paper we detail exploratory work which has yielded order of magnitude speed improvement compared to the common NCBI-Blast nblast program. We present a method for coding index lookup information substantially below entropy, and consider the benefits of a more sparse encoding of the index body using four bits per base instead of the traditional two. The sparse representation allows for rapid comparison of all combinations of possible bases, and when augmented with a modest lookup table allows the comparison of four consecutive nucleotides simultaneously. Rapid approximate scoring of alignments detected in this way is also considered.

This comparison technique is applied to Dash in conjunction with an algorithm which allows a significant reduction in dynamic programming evaluation effort combined with re-using previous dynamic programming effort and is illustrated by searching the human genome[5]. Speed and sensitivity of Dash is compared with Blast illustrating the promise of the methods presented.

2. Design Discussion

2.1 Index Design

The Dash index stores nucleotides with a 4-bit coding. The index itself is divided into a number of sub-indexes of approximately 10-20MB for manageability. A one-hot coding for each of the principle nucleotides (G, A , C and T) is employed. This is then extended to encode the 15 official IUPAC-IUB single-letter base codes[16], using the natural binary combinations which result (see Table 1). A consequence of this representation is that the compatibility of any pair of these codes can be tested with a binary AND operation. For example, H (not-

G) and Y (Pyrimidine bases: T or C) match because 1110 AND 1100 = 1100. i.e. both H and Y allow T or C. Similarly, C and G do not match because 1000 and 0001 = 0000. This allows Dash to detect such similarities in a computationally efficient manner.

Table 1: IUPAC-IUB codes[16] and Their 4-bit Representation

IUPAC	Base	Description	Code
G	Guanine	G	0001
A	Adenine	A	0010
T	Thymine	T	0100
C	Cytosine	C	1000
R	Purine	A or G	0011
Y	Pyrimidine	C or T or U[1]	1100
M	Amino	A or C	1010
K	Ketone	G or T	0101
S	Strong interaction	C or G	1001
W	Weak Interaction	A or T	0110
H	Not-G	A, C or T	1110
B	Not-A	C, G or T	1101
V	Not-T	A, C or G	1011
D	Not-C	A, G or T	0111
N	Any	A, C, G or T	1111

Sequence descriptions and boundaries are stored in a specific section at the end of each sub-index. The list of occurrences for each possible 8 base sub-sequence (8-tuple) is then recorded to allow for rapid location of homology between a query sequence and the indexed database. The list for each tuple is referred to as a *column*.

[1]U is the RNA equivalent of T

The index data is approximately an order of magnitude larger than the database itself. To tackle this, the index data is stored compressed, a concept successfully pioneered by Cafe[2]. To determine an appropriate compression algorithm for Dash, the entropy characteristics were explored when indexing a draft of the Human Genome[5]. The entropy of the list of occurrences of any given 8-tuple in the index was found to be ~83% of original message length (OML), when representing occurrences as 32 bit unsigned integer absolute offsets. The natural step of using relative offsets, instead of absolute, reduces the entropy to ~70% of OML. By using absolute offsets and considering each byte position of the 32 bit values separately, the total entropy of each byte position was found to be ~67% of OML. This reflects the fact that the higher order bytes are much less likely to change than lower order bytes, which are increasingly random in distribution.

A scheme of compressing each occurrence was devised consisting of a 2 bit value which indicates the number of bytes which differ, beginning with the least significant. The bytes are then recorded after the 2 bit length field. This results in a compressed message size ~59% of OML, substantially better than can be obtained by traditional entropy coding techniques. Further, by coalescing the length fields into bytes, followed by the replacement bytes for each of the occurrences referred to by the length field byte, decompression can be achieved almost exclusively with byte aligned operations. This aids computational efficiency on general purpose computers. The entropy of the final compressed data stream was found to be ~99% of the compressed message length.

Fig. 1 shows how this encoding scheme works for eight occurrences of a tuple. The column of 8 digit hexadecimal values are the column values, i.e. the successive offsets at which the tuple occurred relative to the start of the database. Encoding consists of comparing each value with the one before (above) it to determine the minimum number of bytes beginning from the least significant (rightmost) which must be replaced. This is indicated by the shaded bytes.

When the process begins 00000000 is the implied first offset. Note that for the sixth value (00AB0DC) the lowest byte (DC) is identical with the previous value (000581DC). However, three bytes must be replaced, because the higher order bytes differ (i.e. 0AB0 versus 0581). This

inefficiency is not addressed at this stage due to its infrequency. Once the number of bytes which require replacement has been determined (# column), the binary control codes are generated (ctl column), and the byte stream for replacement constructed. Finally this information is aggregated to form a control byte (control value), followed by the associated replacement byte stream. This process is repeated for each four column values and each control value and data stream is stored sequentially as the output. In this example the input stream is compressed from *8 x 4 = 32* bytes to 20 bytes, including the two control bytes.

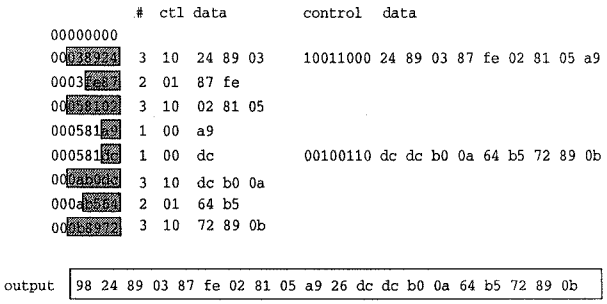

Figure 1: Index Compression Encoding Scheme

2.2 Dynamic Programming Mitigation

2.2.1 Avoiding Dynamic Programming

The Dash system is designed around the principle of considering genomic sequences alignments to typically consist of regions of high (close) homology interspersed with regions of low homology. This in some ways models the comparison and identification of protein super-families where one or more functional domains exist in all members of a given super family.

A consequence of this model when applied to the genomic search process is that it becomes logical to first identify all regions of very close homology. If such regions of close homology are assumed to not contain gaps, then they can be rapidly identified without the aid of an exhaustive dynamic programming search. Dash performs this step efficiently using the index structures described above to identify all occurrences of each sequential overlapping 8-tuple in the database. This is similar to the first step of the FastA[8] algorithm, excepting for the use of a fixed value for *k*, and the employment of an index to speed the process. Each of these occurrences represents an *alignment candidate* (AC). Every alignment candidate is then extended in a non-gapped fashion to achieve a maximum scoring local alignment, referred to as the *extended alignment candidate* (EAC). This process is made more efficient through rapid comparison of sequence segments as described later. Finally, any EAC which have a score below some minimum threshold are discarded at this point.

Once the list of EACs has been obtained, a joining procedure is performed for each sequence with multiple EACs. This joining procedure differs from FastA in that the exact dynamic programming score is calculated for the inter-EAC regions. This is affordable due to the small area of each region. Also, compatibility of EACs is determined initially by considering the distance between them, and as the dynamic programming is evaluated by removing such candidates that have a maximum dynamic programming score below some threshold. The score and path for the best join from each EAC is recorded.

At this point we have a list of EACs and their dynamic programming scores, and also of the best paths between the end points of the EACs. This information allows *amalgamated alignments* (AA) to be assembled, and their exact dynamic programming score calculated, without any further dynamic programming effort. This is in contrast to both FastA and NCBI-Blast which both perform a constrained dynamic programming search of the region around an EAC, not fully utilising the information they have already obtained about the alignment.

A natural question which arises is that of the optimality of joining regions of very close homology end to end, with out considering alternative join paths which may deviate part way along a region of very close homology. One observation which can be made is that if any alternative alignment involving a premature deviation from a very close homology alignment would introduce at least one penalty in leaving the existing alignment (i.e. a gap creation penalty), and would hence require a longer region of very close homology than the region lost through the deviation, and hence is unlikely to occur. In fact, for scoring systems where the gap creation and gap extension penalty are identical it can be shown that deviation will always result in a lower score.

Over all this approach results in the direct dynamic programming time budget of Dash being practically

0. In fact, typically less than 0.00001% of the potential dynamic programming space being evaluated, contributing less than 1% to the total execution time. The dominant remaining time expense (~90%) becomes the identification of the EACs. As a consequence a number of techniques are applied to minimise the work of EAC identification by reducing the number of alignment candidates.

2.2.3 Rapid Scoring of Non-Exact and Non-Specific Matches

When performing dynamic programming expansion or alignment extension, the most desirable situation is when the two sequences significantly concur, and without gaps. Determining when regions of this nature occur, and scoring them appropriately in a computationally efficient manner can be problematic, especially when matches between any of the 15 IUPAC-IUB[16] codes are allowed. The 4-bit per base representation used by *Dash* assists in this process. By performing a binary AND of any two bases it is possible to determine whether they match according to their IUPAC-IUB codes. Further, by constructing a 16 bit lookup table it is possible to determine, for a four base region of the query and subject sequence, how many bases match, as illustrated by Fig.2.

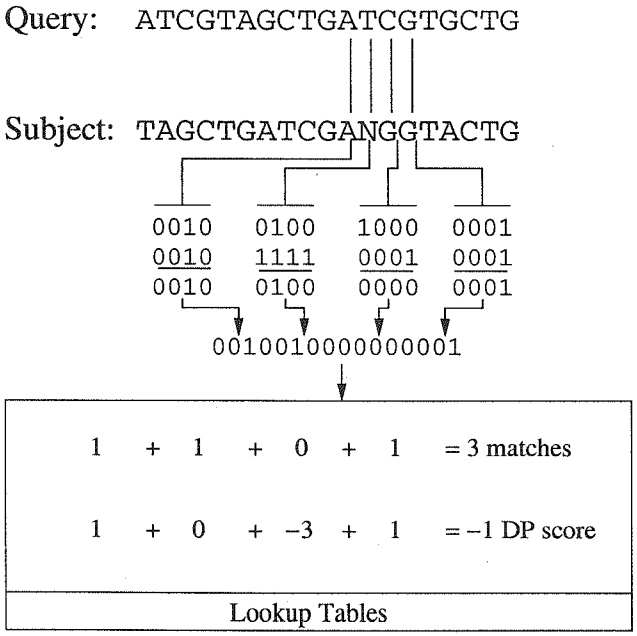

Figure 2: Example of Fast Dynamic Programming Expansion

When determining the dynamic programming score for a given local alignment a second lookup table is also used to calculated the moderated dynamic programming scores for low complexity alignments, e.g. where the subject sequence has a stream of N's. Such regions contribute nothing more than place holding to the alignment. If not treated appropriately, they can result in meaningless high scoring alignments, such as in Fig. 3. Alignments of varying specificity can be accommodated by the look up table. Partially specific matches (e.g. C vs S) are rewarded a portion of the standard reward for a match, based on the probability of a match against them as approximated by counting the number of set bits in the 4 bit representation (Table 2). Matches against N's are neither rewarded with a positive score nor penalised with a negative one.

The effect of this is illustrated in the dynamic programming calculation of Fig. 2 where the alignment of T vs N is given a score of zero, the specific base alignments +1 each , and the mis-match -3, resulting in a combined score of -1. Fig. 4 shows an alignment moderated in this way, resulting in a 367 nucleotide exact alignment being more appropriately scored 332, due it is inclusion of 35 indiscriminate N bases. For scoring systems which work in bits of entropy, those scores can be looked up instead. Given modern computer memory and cache sizes, a 64KB lookup table is of acceptable cost.

Bits clear in base	*P(mismatch)*	*% of Reward*
3 (A,C,G,T)	¾	100
2 (R,Y,M,K,S,W)	½	50
1 (H,B,V,D)	¼	25
0 (N)	0	0

Table 2: Differential Scoring against Uncertain Bases

```
>gnl|UG|Hs#S4066180     AGENCOURT_6580922     Homo
sapiens   cDNA,   5'   end   /clone=IMAGE:5469280
/clone_end=5'/gb=BM554163/gi=18793524/ug=Hs.3216
77 /len=1396

Score = 396

Identities = 10/400 (2%)

Strand = Plus / Plus
```

```
Query: 4156 tggcctgggcgaagagggagag ... tcg 4213
Sbjct: 854  tgncctgggcgnnnnntnnnnn ... nnn 911
Query: 4214 attccccgagaaggaaaaggaa ... gga 4271
Sbjct: 912  nnnnnnnnnnnnnnnnnnnnn ... nnn 969
  |    |      |                    |    |
Query: 4504 tcctcagctctcggcccatcgc ... cgc 4555
Sbjct: 1202 nnnnnnnnnnnnnnnnnnnnn ... nnn 1253
```

Figure 4: Incorrectly Scored Blast Alignment

```
>gnl|UG|Hs#S564724   zp89b06.r1   Homo   sapiens
cDNA,5'   end   /clone=IMAGE:627347   /clone_end=5'
/gb=AA190378 /gi=1779227 /ug=Hs.25 /len=367

Score = 332
Identities = 367/367 (100%)
Strand = Plus / Plus

Query: 13   aaattctggtgactggcatagg ... atc    72
Sbjct: 1    aaattctggtgactggcatagg ... atc    60
Query: 73   aacaatgtcgccaaagcccatg ... acc   132
Sbjct: 61   aacaatgtcgccaaagcccatg ... acc   120
Query: 133  cgtgaaagcnatgatttatncc ... aga   191
Sbjct: 121  cgtgaaagcnatgatttatncc ... aga   179
  |    |      |                    |    |
Query: 369  aagtgtcctGC                      379
Sbjct: 357  aagtgtcctGC                      367
```

Figure 5: Correctly Scored Dash Alignment

3 Method

3.1 Index Construction

The index was built with a maximum sub-index size of 20,000 sequences or 20,000,000 nucleotides, whichever was encountered first. The filtering threshold for the exclusion of frequent 8-tuples was set at 1.50 random expected frequency.

3.2 Test Cases & Speed Comparison

To test the speed and sensitivity of the algorithm features described above Dash was compared with NCBI-Blast 2.2.6[7] blastn. Both Dash and the blastn program were executed in a single processor thread to allow direct comparison of the amount of work each algorithm requires to perform a given search to be measured in CPU seconds. Each were required to perform the same 200 randomly selected queries against a draft of the human genome[5]. These queries were obtained by randomly selecting 50-50,000 base sections from the human genome data, ignoring sequence boundaries. Only nucleotide-nucleotide searching was considered. The user space run time of each program was then recorded for later comparison. All tests were performed on a single processor AMD Opteron 1.4GHz system with 1MB cache and 1GB of main memory running RedHat Linux 7.2 in 32 bit mode.

2.3.3 Sensitivity Measure

The sensitivity of Dash was measured relative to the Blast 2.2.6 nblast program. For each test case the largest alignment from the first 100 sequences returned were compared. The percentage of each of hit which program B returned which program A also identified were summed to calculate the score for program A, and vice versa. Therefore a score of 100 is perfect.

4 Results

4..1 Effect of Query Sequence Length on Dash Performance

The relationship between Dash's speed advantage compared to Blast, and the query sequence length is given in Table 3. In fact, Dash's edge is greatest for shorter query sequences, especially below 500-1000bp, where the advantage consistently exceeds an order of magnitude. This is particularly pertinent as it is very common to search for queries with lengths below this bound. At the other extreme Dash's advantage decays to a mean of eight times faster than Blast.

	Mean	Median	Max	Min
Dash/Blast Run-Time	8.09	7.02	32.21	2.64
Dash Top-100 Score	95.26	99.43	99.9	6.98
Blast Top-100 score	96.21	99.98	100	4

Table 3: Dash Speed-Up and Sensitivity versus Blast

5 Conclusions

In conclusion, Dash shows consistent sensitivity compared to Blast. When dynamic programming mitigation measures are engaged, impressive and consistent speed improvements exceeding an order of magnitude are realised for the common case, and around eight times in other cases. Preliminary moderation of alignment scores to take into account indiscriminate bases has also been successfully demonstrated.

Possible areas of future work include extending the scoring scheme to efficiently calculate the entropy of an alignment during the dynamic programming stage, and assessing the sensitivity through a broader testing regime to identify the relative strengths and weaknesses of the algorithm. Comparing the algorithms performance when attempting to identify members of a protein super-family would provide a valuable basis for determining the acceptability of the heuristic algorithm presented from a biological application perspective.

Acknowledgements

The authors would like to acknowledge the support of the CSSIP (Co-operative Centre for Signal, Sensor and Information Processing, Australia) *Firmware for Genomics* project.

References

[1] Time Logic Corporation technology overview, http://www.timelogic.com/ technology.html

[2] Williams Hugh E, Zobel Justin, Indexing and Retrieval for Genomic Databases, *Knowledge & Data Engineering*, No. 14(1), 2002, pp. 63-78.

[3] A. Califano and I. Rigoutsos, FLASH: A fast look-up algorithm for string homology', *In International Conference on Intelligent Systems for Molecular Biology*, pp. 56-64, Bethesda, MD.

[4] FLAG Fast Local Alignment for Gigabases, http://flag.itri.org.tw

[5] Human Genome, Working Draft as at 19 June 2002,ftp://ftp.ncbi.nih.gov/repository/UniGene/ Hs.seq.all.gz

[6] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. (1990) Basic local alignment search tool, *J. Mol. Biol.*, 215, pp.403-410, 1990.

[7] Blast 2.2.6: Altschol, Stephen F., Thomas L. Madden, Alejandro A. Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997), Gapped BLAST and PSI-BLAST: a new generation of protein database search programs, *Nucleic Acids Res.* 25, pp.3389-3402,1997.

[8] Rapid and sensitive sequence comparison with FASTP and FASTA: Pearson, W. R., *Methods in Enzymology,* 183, pp.63-98, 1990.

[9] Smith TF, Waterman MS, Identification of common molecular subsequences, *The Journal of Molecular Biology,* March 25; No. 147(1), 1981, pp 195-197, 1981

[10] Galisson, ``The Fasta and Blast programs"

[11] Dwan C, Raghavan S ,Blasting vs Watering Down: An analysis of BLAST performance vs Smith-Waterman

[12] Wootton, J. C. and S. Federhen, Statistics of local complexity in amino acid sequences and sequence databases, *Computers in Chemistry* 17, pp.149-163, 1993.

[13] Wootton, J. C. and S. Federhen, Analysis of compositionally biased regions in sequence databases, *Methods in Enzymology* 266, pp. 554-571, 1996.

[14] Hancock, J. M. and J. S. Armstrong, SIMPLE34: an improved and enhanced implementation for VAX and Sun computers of the SIMPLE algorithm for analysis of clustered repetitive motifs in nucleotide sequences, *Comput Appl Biosci* 10, pp.67-70, 1994.

[15] Claverie, J-M. and States, D.J., Information enhancement methods in large scale sequence analysis, *Computers in Chemistry*, (1993), 17, pp.191-201, 1993

[16] IUPAC-IUB single-letter base codes, from http://www.molbiotech.chalmers.se/programs/ kbb055_strukturbiokemi/BI_C1.pdf

[17] Gardner-Stephen, P. M. and Knowles, G., A Novel Architecture for Genomic Sequence Searching & Alignment, *Advances in Computer Systems Architecture*, 2003.