NP3: A New DNA Compression Algorithm

Paul Gardner-Stephen and Greg Knowles
Embedded Systems Laboratory,
School of Informatics and Engineering, Flinders University,
GPO Box 2100, Adelaide 5001, Australia

Abstract:- NP3 is a nucleotide database compression algorithm which takes advantage of the redundancy among sequences in genomic databases. We demonstrate the effectiveness of the NP3 algorithm by compressing the bodies of the UniGene [1] transcriptome database at 0.71 bits per base, while offering sequential and random access decompression speeds peaking in excess of 500,000 sequences per second. This is significantly better than the results offered by existing algorithms[2, 3, 4, 5, 6].

Key-Words:- Data compression, Blast, DNA, UniGene, Human Genome, zip

1 Introduction

There is little doubt that genomic databases are growing in size at a rate which exceeds that of both Moore's Law and the increase in hard disk storage capacities. There is also the rising spectre of mass sequencing systems entering production in the future. If computer systems are swamped by current data volumes, then such systems capable of sequencing perhaps 10^8 bases per hour threaten to drown them. However, it can be observed that while the volume of sequencing data is increasing exponentially, that the increase in entropy is much closer to linear. The homology between species being sequenced, the homology among individuals and varieties of organisms as well as the homology of sequences within an organism together suggest that a significant proportion of the entropy presented by individual sequences is in fact common. It therefore follows that it should be possible to produce compression algorithms which take advantage of this inter-sequence miscibility to produce extremely compact representations. As the volume of sequencing grows, and the redundancy increases it is not unreasonable to envisage compression ratios an order of magnitude better than the 1.6 to 1.8 bits/base of current algorithms. This is the motivation of NP3, to produce an algorithm which focuses on harnessing the inter-sequence redundancy which is present and will continue to grow as aggregate database sizes continue to balloon. Variants of this approach have been employed in the past, for example the EMBL-ALIGN database efficiently stores groups of sequences as multiple alignments.

The general data flow of the NP3 Algorithm proceeds as follows: the Fasta format database is initially parsed, separated into sequence description and body streams which are then handled separately. The remaining processes are elucidated in the corresponding text below which describes how the compressed representations of the description and body data are generated, collated and segmented to produce an NP3 file.

Finally, as genetic databases are continually evolving we have designed the NP3 file format to be easy to patch, so that changes can be incorporated without the need to re-compress the whole database.

1.1 Discovery and Indexing of Redundancy in Sequence Bodies

Redundancy is discovered within and among sequences by searching for matching regions in a window of sequence data spanning of the order of several hundred sequences or kilo bases, which ever is the lesser. This allows the discovery of redundancy between sequences with substantially conserved regions. Consider Figure 1 where three sequences share varying portions of a common subsequence, as indicated by the striped region. In this case sequence #2 contains a region which also oc-

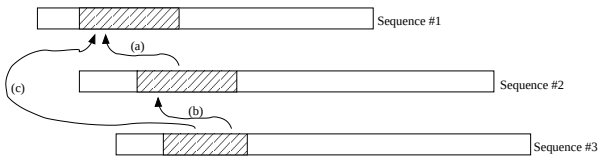

Figure 1: Example of Chained Inter-Sequence Redundancy. Sequence #3 has two options for sourcing the redundant matter (the shaded section of each sequence). The source most distant to the sequence (i.e. most proximal to the start of the database) is always chosen in preference (Sequence #1 in this case).

curs in sequence #1, as indicated by arrow (a). Similarly sequence #3 contains a portion of that same region, which occurs in both sequences #1 and #2 (arrows b and c). Where such multi-way redundancies occur, they are linked to the most distant sequence. In the example, this means that the redundancies of both sequences #2 and #3 get recorded against sequence #1. This approach of maximising the distance to the host sequence acts to minimise the number of hops required to eventually find the redundant sequence fragment. This helps to keep the decompression time to a minimum when extracting a random sequence from the database.

The recorded maximal alignment information from the redundancy discover phase is fed into a dynamic programming system (see Section 1.2). There it is combined with information about how many bits it takes to encode each redundancy type to determine an optimal, i.e. lowest bit count, token stream to represent the sequence. Some portions of a given sequence may only be recurrent in such short fragments that using those redundancies would be unprofitable. The dynamic programming algorithm takes this into account and determines which fragments to encode by reference, and which are short enough to be preferentially encoded directly in the token stream. Directly coded sequence is encoded three bases per byte with a two bit identifier prefix, or four bases per byte for longer spans. Run Length Encoding (RLE) is used where possible to encode repeating residues or residue pairs to reduce the message length. Processing of wild cards and case change events is described later. Finally, in order to facilitate the fastest random access decompression, references to recent sequences are only used if they actually result in a lower bit count. If the bit count is equal to using another encoding, e.g. run length encoding, then that encoding will be

Input Sequence	`ACAAAAAAAAGTCGTG`	
Offset Zero	`ACA`	Short Direct Encoding: final offset=3 cost=8 bits
	`ACAAAAAAAAGTCGTG`	Long Direct Encoding: final offset=16 cost=48 bits
Offset One	`CAA`	Short Direct Encoding: final offset=4 cost=8 bits
Offset Two	`AAAAAAAA`	Run Length Encoding: final offset=10 cost=16 bits
	`AAA`	Short Direct Encoding: final offset=5 cost=8 bits
	`AAAAAAAAG`	Redundant (Recurring) Content: final offset=11 cost=16 bits
•••	•••	•••

Figure 2: Partial Exploration of Possible Token Encodings.

used. The benefit for random access decompression arises by reducing the frequency of inter-sequence references which require the extra computation of resolution.

Selecting an optimal set of encoding methods for a broad class of inputs is not trivial. We do not claim that we have selected an optimal encoding scheme at this time. However, as the results will later show, we have devised an effective encoding scheme which provides sound compression performance without unduly sacrificing decompression speed. The full encoding scheme is presented in Table 1, with various features explained in the following paragraphs. In the table various bit positions are marked as containing distant relative address (‘a’), local relative offset (‘r’), number of bases encoded (‘n’), extra address or length bits (‘x’), change of case (‘c’), or a base represented in either 2 bit (‘bb’) or 4 bit (‘base’) format.

1.2 Compute Optimal Encoding Method (Tokenisation) for Each Sequence

The optimisation process determines the best set of coding possibilities at each offset of a sequence. The reach of these possibilities, i.e. how many residues can be encoded at that point using the available code, and the bit cost of doing so are both recorded. Let’s apply this to the sequence ACAAAAAAAAGTCGTG, as an example. Figure 2 illustrates the discovery of some of the coding possibilities.

Once the cost of each possible transition has been identified, we can construct a table which gives the cumulative cost of the beginning of the sequence (start offset= 0) to any final offset, and use dynamic programming to determine the optimal (minimal) cost from offset zero to any given point in the sequence.

Token Number	Binary Value	Bits	Description
1	00aaaaaa aaaaaaaa aaaaaaaa nnnnnnnn	32	Redundancy with address encoded as a 22 bit value. The lower 13 bits are the offset in a first sequence, and the upper bits encode the sequence number of the first sequence relative to the current sequence. Therefore it is possible to address up to 512 recent sequences.
2	011nnnnn nnnnnnnn bb$\times n$	$16+2\times n$	A series of 2 bit encoded bases. This is how long stretches of non-redundant sequence is encoded with an efficiency approaching that of direct coding. n must be a multiple of 4.
3	100000bb	8	A single 2 bit encoded base.
4	100001rr nnnnnnnn	16	Redundancy with address encoded as a 2 bit value relative to the previously accessed address. Useful for encoding regions with short indels. This is an optimisation which is used frequently to reduce the size of the address.
5	011nnnnn basebase	8	A run of identical base pairs, e.g. GCGCGC...
6	10001001 basennnn	16	A run of up to 16 identical bases.
7	1000101c 0000base	16	A single 4 bit encoded base with an implied case shift before it, and an optional case shift after (4 bit encoding allows the use of all 16 IUPAC codes for nucleotide bases).
8	100011cc basebase	16	Two 4 bit encoded bases with an implied case shift before the first, and optionally after each base.
9	1001bbbb	8	Two 2 bit encoded bases.
10	10100xxx	8	The extend token is placed immediately after any other token to add three extra bits to the length fields, e.g., a maximum range of 256 becomes 2048.
11	101010aa nnnnnnnn	16	Redundancy with address encoded by the specified one of four entries from the recent address table.
12	101011aa rrrrrrrr nnnnnnnn	24	Redundancy with address encoded by the specified one of four entries from the recent address table, plus an 8 bit relative offset (rrrrrrrr).
13	1011rrrr rrrrrrrr nnnnnnnn	24	Redundancy with address encoded by the last sequence address used, plus 12 bit offset.
14	11bbbbbb	8	Three 2 bit encoded bases.

Table 1: NP3 Binary Encoding Scheme for Genomic Sequence Data.

1.3 NP3 Encoding Methods

The most apparent characteristic of our method is the use of variable length, byte aligned tokens. The decision to do this was driven by the desire for fast decompression. While it is true that this potentially makes a trade-off against compression performance, we believe that the utility of the algorithm is not dependent solely on that criteria. Rather, access time is also an important characteristic of any data storage system. In the event that a better coding method is later developed, it is relatively trivial to replace the incumbent method. The NP3 file header already records information about the encoding system version to allow backward compatibility in that situation. Indeed the separation of the application programming interface (API) from the underlying storage mechanism is another of the goals of the NP3 project. To increase the probability of local relative addressing when encoding redundancies, the four most recently accessed addresses are maintained during the tokenisation or detokenisation of a sequence. The penultimate three codes in Table 1 provide this functionality. The four addresses in this list give starting points for rendundant coding. If these points are wisely chosen then the magnitude of any relative address will be much less than the 21 bit address required if the first code were used. This allows for efficient construction of one sequence from regions obtained from alternating sequences. These four addresses are initialised to point to the start of, the current sequence (to aid self-redundancy, i.e., repeats), the last 32 bases of the previous sequence to allow trivial encoding of the overlap region at the start and end of long segmented sequences, the start of the furthest sequence which can be addressed, and finally the start of the previous sequence, to aid efficiency when encoding a series of similar sequences. These optimisations enhance the likelihood of using the shorter relative addressing encoding methods.

Direct encoding is supported using 8 bits for up to three bases, or 16 bits plus 2 bits per base for longer extents. The shorter spans are used so that all longer spans start at an offset which is multiple of four bases from the start of the sequence. This enables the use of simple byte copying when decompressing to 2 bits per base format, for sequence search and alignment purposes. It also simplifies the work of the optimiser by reducing the number of coding possibilities (Section 1.2). The address of redundancies can be encoded either in absolute or relative form. The magnitude of relative addresses is minimised by referring to the nearest end of the redundancy. A list of recently referenced addresses is maintained to further increase the likelihood of small magnitude relative addresses. The addresses in this list are incremented as processing advances in an attempt to predict the address of subsequent references. Short spans of dissimilarity in otherwise identical spans of sequence can therefore be compactly encoded. The need to store lengthy relative addresses is elided by the prediction of their value. Even when the prediction is not completely accurate it still acts to reduce the magnitude of the relative address, and thus the number of bits required to encode it. It is acknowledged that this is one area where using byte aligned tokens does effect compression.

A common practice for efficiently storing wild card characters is to use 2 bits to store each residue, wild card or not, and then to have a table of wild cards to superimpose on that data when it is extracted. This method, proposed by Williams et al[7] allows wild cards to be very efficiently encoded and relies on and exploits the relatively rarity of their occurrence. The method presented here differs from that approach in that these events are stored in-line with the sequence representation. Particularly, in the case of the N wild card, it is common for long contiguous runs to occur. Such situations typically constitute the majority of wild cards present in a database. Therefore, the storage of wild cards has been merged with Run Length Encoding (RLE). That is, to store even a single wild card results in an RLE token. This avoids the need to maintain a table of wild card "patches" to apply to each sequence.

Preservation of case is not usually dealt with in genomic compression, since it generally contributes only meta-data to a sequence. However, as it occurs rarely it should be possible to record this information in an economical manner. Empirical evidence suggests that case changes tend to be either applied to a single residue, or last many residues. Thus, case changes are recorded in NP3 with a two byte token. This token dictates the initial case change, and contains the next two bases.

If the case change is for only one base, that fact is included in the token. In this way the common case of two consecutive case changes, i.e. an isolated character appearing in reverse case, is dealt with by a single token of 16 bits.

The optimisation framework just described makes it relatively simple to employ any encoding method desired, and obtain the best results that it can afford. This is due to the decoupling of the optimiser from the actual bit patterns required. The optimiser is only aware of the cost of each option at a given point. This provides flexibility in later improving the coding scheme, without requiring significant change to the optimiser. It also makes the NP3 approach readily adaptable to other types of data, e.g., protein or natural languages.

2 Results

To assess the performance of NP3 versus existing compression algorithms the UniGene* Human transcriptome database was compressed using each tool†. The space used by each algiruthm to represent the sequence bodies only (as distinct from the sequence descriptions) is presented, where possible. These results are given in Table 2. They show that NP3 produces a significantly smaller sized result, compressing the sequence bodies at only three quarters of a bit per base. This compares exceptionally well with the 2.09 bits/base achieved by the BLAST[8] formatdb program, which is the only format surveyed which allows for rapid random access retrieval of individual sequences in a large compressed database. Furthermore, the NP3 file only compresses by 3% when subjected to bzip2, suggesting that there are no gross inefficiencies in the coding methods or the file format. NP3 compresses the database to 70% of the size achieved by of the next most effective algorithm (bzip2), and in contrast offers random accessibility.

Format	Sequence Bodies		Time
	MB	Bits/Base	
Fasta (ASCII)	1,886	8.06	na
BLAST[8] (formatdb)	489	2.09	6:09
DNAcompress [5]	459	1.96	30:00
gzip	300	1.28	6:08
bzip2[9]	285	1.21	13:23
NP3	172	0.71	240

Table 2: Comparison of NP3 File Size With Other Formats for UniGene (1.96×10^9 residues). NP3 produces a significantly smaller file than any of the other methods, almost 10 times smaller than the original Fasta file version of the database.

Retrieval speeds of individual sequences (without their FASTA descriptions) peaked at over 160,000 sequences per second. For sequential access, similar speed is obtained even if sequence descriptions are required. NP3 offers competitive compression in situations where current algorithms struggle due to the lack of intra-sequence redundancy. For example, bacteriophage database GBPHG, taken from a recent GenBank [10] release, consists of the entire genome of a collection of bacteriophages. DNACompress was able to compress this database only slightly, requiring 1.95 bits per base. In comparison NP3 was able to compress this difficult data at 1.68 bits per base, by leveraging the homology not only within each genome, but also among them.

Where NP3 does perform relatively poorly is for very small databases, e.g. hundreds of kilo-bases or less, and where intra-sequence homology is the dominant compression opportunity. The set of sequences considered by [5] for DNACompress and various other DNA compression algorithms which focus on intra-sequence homology are typical of this. Where DNACompress is able to represent that set of 11 sequences using 1.73 bits per base NP3 requires 1.88. The authors plan to engage this intra-sequence homology issue in a subsequent version of the NP3 program.. We have shown it is possible to completely extract the 1.95×10^9 bases and 470MB of descriptions of the more than 3 million sequences of this database in less than 40 seconds.

*The UniGene database is a sorted transcriptome with substantial local redundancy which is why all the algorithms here (NP3, gzip, bzip) are able to compress it more efficiently than they are for general DNA databases.

†The academic version of DNAcompress, while written in Java, is packaged to run only on Windows. Windows does not offer true CPU time accounting, so obtaining an accurate time was not possible. The processor used in this case was a 1.77GHz AMD Athlon, which runs BLAST in almost identical time to a 1.8GHz AMD Opteron.

NP3 Compression was performed in parallel on a Linux cluster with 16 1.8GHz AMD Opteron processors.

3 Conclusions and Future Research

The results demonstrate that NP3 is capable of producing significantly smaller lossless representations of than existing methods for highly clustered nucleotide databases, such as UniGene, while preserving easy and rapid random access to sequences. Although compression is expensive (in time), NP3 has a parallel build mode that ensures that compression times are kept to a minimum. The results here were obtained on a cluster of 16 1.8GHz Opteron processors. Also the ease of updatability implies that the full database need only be rebuilt infrequently.

The authors recognise that a bit aligned format would be more efficient, but possibly at the cost of decompression speed. More generally, gains are possible (and quite likely) through further exploration of coding method space. In particular, the discrepancy between NP3 and DNAcompress is a clear indication that despite high efficacy at harnessing inter-sequence redundancy, there is room for improvement in the addressing of intra-sequence redundancy. The pre-clustered UniGene databases are especially favourable to the compression techniques used here, suggesting that reordering and/or clustering of similar sequences to increase the redundancy will achieve significant gains for non-clustered databases. This is an area in which we are currently working.

Finally, work by Kuri-Morales et al[11] for protein databases shows that there is significant inter-sequence homology which can be leveraged to produce compact representations, suggesting that this should be a fruitful avenue of pursuit.

References

[1] NCBI. UniGene Transcription Database. ftp://ftp.ncbi.nih.gov/repository/UniGene/Homo_sapiens, June 2002.

[2] Grumbach S. and Tahi F. A new challenge for compression algorithms: genetic sequences. *J. Inform. Process. Management*, vol. 30, 1994, pp. 866–875.

[3] Matsumoto T., Sadakane K., and Imai H. Biological sequence compression algorithms. *Genome Informatics Workshop, Universal Academy Press*, 2000, pp. 43,52.

[4] Chen X., Kwong S., and Li M. A compression algorithm for DNA sequences. *IEEE Engineering in Medicine and Biology Magazine*, vol. 20, 2001, pp. 61,66.

[5] Chen X., Li M., Ma B., and Tromp J. DNACompress: fast and effective DNA sequence compression. *Bioinformatics Application Note, Oxford University Press*, vol. 18(12), 2002, pp. 1696–1698.

[6] Korodi G. and Tabus I. An Efficient Normalized Maximum Likelihood Algorithm for DNA Sequence Compression. *ACM Transactions on Information Systems*, 2005, pp. 3–34.

[7] Williams H.E. and Zobel J. Compression of Nucleotide Databases for Fast Searching. *Bioninformatics*, vol. 13, 1997, pp. 549–554.

[8] Altschul S.F., Madden T.L., Schaeffer A.A., Zhang J., Zhang Z., Miller W., and Lipman D.J. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, vol. 25, 1997, pp. 3389–3402. URL `http://www.ncbi.nlm.nih.gov/BLAST`.

[9] Seward J., Burrows M., Wheeler D., Fenwick P., Moffat A., Neal R., and Witten I. The BZIP2 Compression Program, circa 2001. URL `http://sources.redhat.com/bzip2`.

[10] Benson D.A., Karsch-Mizrachi I., Lipman D.J., Ostell J., Rapp B.A., and Wheeler D.L. GenBank. *Nucleic Acids Research*, vol. 28(1), January 2000, pp. 15–18.

[11] Kuri-Morales A.F. and Ortiz-Posadas M.R. A New Approach for Representation in Biological Sequences. *WSEAS Transactions on Biology and Biomedicine*, vol. 3, January 2006, pp. 31–36.