Referências

[AQT06] **"AutomatedQA, AQTime"**. Disponível via http://www.automatedqa.com/products/aqtime/index.asp Acessado em dezembro de 2006.

[ASN05] **"ASN.1 Description"**. Disponível via: http://asn1.elibel.tm.fr/xml/. Acessado em dezembro de 2006.

[AZM+02] Don Adjeroh, Yong Zhang, Amar Mukherjee, Matt Powell E Tim Bell. **"DNA Sequence Compression using the Burrows-Wheeler Transform"**. Proceedings of the IEEE Computer Society Bioinformatics Conference. pp.303, 2002.

[BH04] S. J. Bedathur, J. R. Haritsa**. "Search-Optimized Persistent Suffix Tree Storage for Biological Applications"**. Technical Report TR-2004-04. Indian Institute of Science. 2004.

[BKY03] J. Bedell, I. Korf, Ma. Yandell. "**BLAST"**. Ed. O'Reilly. 2003.

[Bla05] **"BLASTGRES"** Biosequence data management in PostgreSQL. Disponível via https://sourceforge.net/project/showfiles.php?group_id=156045, Acessado em dezembro de 2006.

[BW94] M. Burrows and D.J. Wheeler. **"A Block-sorting Lossless Data Compression Algorithm"**. Digital System research Center. 1994.

[BWT06] Funcionamento do algoritmo BWT. Disponível via http://www.ime.usp.br/~fli/bwt.html. Acessado em dezembro de 2006.

[BWTa06] Funcionamento do algoritmo BWT. Disponível via http://marknelson.us/1996/09/01/bwt/ . Acessado em dezembro de 2006.

[CLM+02] X. Chen, M. Li, B. MA, J. Tromp **"DNACompress: fast and effective DNA sequence compression"**. Bioinformatics 18, pp.1696–1698. 2002.

[Cor85] Gordon V. Cormack. **"Data compression on a database system".** Communications of the ACM archive. Vol. 28, pp. 1336-1342. 1985.

[Cos02] R.L.C. Costa. **"Alocação de dados e distribuição de carga para execução paralela da estratégia BLAST de comparação de seqüências".** Dissertação de Mestrado, PUC-Rio, 2002.

[CRD+04] G. Cooper, M. Raymer, T. Doom, D. Krane and N. Futamura **"Indexing Genomic Databases".** Proceedings of the Fourth IEEE Symposium on Bioinformatics and Bioengineering. pp 587, 2004.

[CT97] Cleary J.G. e Teahan W.J., **"Unbounded length contexts for PPM"**. The Computer Journal, 40(2/3), pp. 67-75, 1997.

[CWC06] Michael Cameron, Hugh E. Williams e Adam Cannane. **"A Deterministic Finite Automaton for Faster Protein Hit Detection in BLAST"**. Journal of Computational Biology. Vol. 13, No. 4 pp. 965-978. 2006

[DBL+00] S. Delaney, G. Butler, C. Lam, L. Thiel. **"Three Improvements to the BLASTP Search of Genome Databases"**. 12° International Conference on Scientific and Statistical Database Management. pp.14 . 2000

[DDB05] DNA Databank of Japan, **"DDBJ"**. Disponível via http://www.ddbj.nig.ac.jp/. Acessado em dezembro de 2006.

[EBI05] European Bioinformatics Institute, **"EMBL – Nucleotide Sequence Database"**. Disponível via http://www.ebi.ac.uk/embl/. Acessado em dezembro de 2006.

[FAB05] European Bioinformatics Institute. **"Formato de Arquivos de biosseqüências".** Disponível via http://www.ebi.ac.uk/help/formats_frame.html. Acessado em dezembro de 2006.

[FIL06] **"FileMon"**. Disponível via http://www.sysinternals.com/Utilities/Filemon.html. Acessado em dezembro de 2006.

[For05] **"Manual de Referência FORMATDB".** Disponível via http://bioinformatics.ubc.ca/resources/tools/formatdb. Acessado em dezembro de 2006.

[GHS95] Gautam Ray, Jayant R. Haritsa e S. Seshadri. **"Database Compression: A Performance Enhancement Tool".** Proceedings of 12th Intl. Conference on Management of Data (COMAD). pp. 106-125, 1995.

[GJ97] Cleary J.G.e Teahan W.J., **"Unbounded length contexts for PPM".** The Computer Journal, 40(2/3), pp.67-75, 1997

[GOD05] GENE ONTOLOGY. **"Gene Ontology Database"**. Disponível via http://www.godatabase.org/dev/database/. Acessado em dezembro de 2006.

[Gre05] Laboratory of Phil Green, **"Documentation for Phrap"**. http://www.phrap.org/. Acessado em dezembro de 2006.

[GT93] S. Grumbach, S. F. Tahi. **"Compression of DNA sequences".** Proceedings of the Data Compression Conference, pp. 340–350,1993.

[Gut84] Antonin Guttman. **"R-Trees: A dynamic index structure for spatial searching".** Proceedings of the 1984 ACM SIGMOD international conference on Management of data. pp 47-57. 1984.

[HAI01] E. Hunt, M. P. Atkinson, R. W. Irving. **"A Database Index to Large Biological Sequences".** Proceedings of the 27th VLDB Conference, pp.139-148, 2001.

[HM99] Huang, X., Madan, A. **"CAP3: A DNA sequence assembly program"**. Genome Research, 9: pp. 868-877, 1999.

[HPY05] Ruey-Lung Hsiao, Douglas Stott Parker Jr. e Hung-chih Yang. **"Support for BioIndexing in BLASTgres".** 2nd International Workshop on Data Integration in the Life Sciences (DILS), pp 284-287, 2005.

[Huf05] **"Implementações do algoritmo de Huffman"**. http://datacompression.info/Huffman.shtml. Visitado em dezembro de 2006.

[Huf52] Huffman, David A. **"A Method for the Construction of Minimum-Redundancy Codes"**. Proceedings of the IRE. pp 1098-1101, 1952.

[Ken02] W. J. Kent **"BLAT - the BLAST-like alignment tool"**. Genome Research, nro 12, pp. 656-664. 2002.

[KT05] G. Korodi, I. Tabus. **"An Efficient Normalized Maximum Likelihood Algorithm for DNA Sequence Compression"**. ACM Transactions on Information Systems, Vol. 23, No. 1, pp. 3–34, 2005.

[LBC03] M. Lemos, A. Basílio, M. A. Casanova. **"Um estudo dos algoritmos de montagem de fragmentos de DNA"**. Monografia de Ciência da Computação Departamento de Informática, PUC-Rio. 2003.

[LC00] M. Lemos, M. A. Casanova. **"Algoritmos para análises de seqüências".** Monografia de Ciência da Computação. Departamento de Informática, PUC-Rio. 73 pp., 2000.

[LH87] Debra A. Lelewer e Daniel S. Hirschberg. **"Data Compression".** ACM Computing Surveys, Vol.19, Número 3, pp 261-296, 1987.

[LL03] M. Lemos e S. Lifschitz. **"A Study of a Multi-Ring Buffer Management for BLAST".** 1st International Workshop on Biological Data Management, In conjunction with DEXA. pp.5-9, 2003.

[LM03] D. Leon, S. Markel. **Sequence Analysis in a Nutshell**. O'Reilly, 2003.

[LMM+04] M. Lemos, L. F. Seibel, J. A. Macedo, F. Mano, O. Martins, G. Paiva e Silva, P. L. Oliveira, P. Cavalcanti, P. Bisch, M. Bertalan, S. Rossle, A. Coelho. **"BioNotes - Annotation System of Biosequences: Its Application in the Rhodnius prolixus Project".** XX Annual Meeting of the Brazilian Society of Protozoology. 2004.

[LSC03] M. Lemos, L. F. Seibel, M. A. Casanova. **"Sistemas de Anotações em Biossequências"**. Monografia de Ciência da Computação. Departamento de Informática, PUC-Rio. 2003.

[MDS99] J. Thierry-Mieg, R. Durbin, L. D. Stein. **"ACEDB: A Genome Database Management System".** Computing in Science and Engineering. Vol. 1, No. 3, pp. 44-52. 1999.

[MR02] Michele Magrane, Rolf Apweiler. **"Organisation and Standarsation of Information in SWISS-PROT and TREMBL - EMBL Outstation"** – Data Science Journal 1. Vol1, pp. 13-18. 2002.

[MXM03] D. Miranker, W. Xu, R. Mao. **"MoBIoS: A Metric-Space DBMS to Support Biological Discovery"**. Journal Statistical and Scientific Database Management. 2003

[Ncb05] NCBI - National Center of Biotechnology Information. **"BLAST"**. Disponível via http://www.ncbi.nlm.nih.gov/BLAST/. Acessado em dezembro de 2006.

[Ncb05b] NCBI - National Center of Biotechnology Information. **"BLAST Databases"**. Disponível via ftp://ftp.ncbi.nih.gov/blast/db/. Acessado em dezembro de 2006.

[Ncb05c] NCBI - National Center of Biotechnology Information, **"GENBANK".** Disponível via http://www.ncbi.nlm.nih.gov/Genbank/. Acessado em dezembro de 2006.

[NE00] S. Navathe, R. Elmasri **"Fundamentals of Database Systems"**. Terceira edição, Addison Wesley. 2000.

[NP04] S. Navathe, U. Patil. "**Genomic and Proteomic Databases and Applications: A Challenge for Database Technology**". 9th International Conference on Database Systems for Advanced Applications (DASFAA). pp.1-24. 2004.

[OB05] Ç. Özbütün, H. Baer. **"Table Compression in Oracle Database 10g Release 2"**. Oracle White Paper. Oracle Corporation. 2005.

[OF03] O. Ozturk, H. Ferhatosmanoglu **"Effective Indexing and Filtering for Similarity Search in Large Biosequence Databases"**. Proceedings of the Third IEEE Symposium on BioInformatics and BioEngineering, pp 359, 2003.

[PFM06] Microsoft Corporation. **"PFMon - Page Faults Monitor".** Disponível via http://windowssdk.msdn.microsoft.com/en-us/library/ms726579.aspx. Acessado em dezembro de 2006.

[PL88] Pearson, W. R., Lipman, D. J. **"Improved Tools for Biological Sequence Comparison**". Proceedings of the National Academy of Sciences of the United States of America. Vol. 85, pp. 2444-2448. 1988.

[Pos05] **"POSTGRESQL - Sistema Gerenciador de Banco de Dados".** Disponível via http://www.postgresql.org/. Acessado em dezembro de 2006.

[PS91] M. L. Pearsons, D. Soll. **"The Human Genome Project: a paradigm for information management in the life sciences"**. The FASEB journal: official publication of the Federation of American Societies for Experimental Biology. Vol. 5, pp. 35-39, 1991.

[PST+83] Peltola, H., Soderlund, H., Tarhio, J., Ukkonen, E.. "**Algorithms for some string matching problems arising in molecular genetics". Proc.of the 9th IFIP World Computer Congress, pp. 59-64, 1983.**

[PSU84] Peltola, H., Soderlund, H., Ukkonen, E. **"SEQAID: A DNA sequence assembling program based on a mathematical model"**. Nucleic Acids Research, 12, pp. 307-321, 1984.

[RAV06] Rav Ahuja. **"Introducing DB2 9, Part 1: Data compression in DB2"** Disponível via http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0605ahuja/. Acessado em dezembro de 2006.

[RV93] Mark A. Roth e Scott J. Van Horn**. "Database Compression"**. SIGMOD RECORD, Vol. 22, Número 3, pp 31-39, 1993.

[SCD+05] S. M. Stephens, J. Y. Chen, M. G. Davidson, S. Thomas, B. M. Trute. **"Oracle Database 10g: a platform for BLAST search and Regular Expression pattern matching in life sciences"**. Nucleic Acids Research, 33, pp D675–D679. 2005

[SLL00] L. F. Seibel, M. Lemos, S. Lifschitz. **"Genome Databases"**. Tutorial apresentado no SBBD - XV Simpósio Brasileiro de Banco de Dados**.** 2000.

[SWA+95] Sutton, G., White, O., Adams, M., Kerlavage. **TIGR assembler: A new tool for assembling large shotgun sequencing projects**. Genome Science & Technology, Nro.1 pp. 9-19, 1995.

[SWI05] Swiss-Prot Protein knowledgebase. **"SWISS-PROT TrEMBL".** Disponível via http://ca.expasy.org/sprot/. Acessado em dezembro de 2006.

[TDS99] Jean Thierry-Mieg, Richard Durbin, Lincoln D. Stein. **"ACEDB: A Genome Database Management System"**. IEEE Journal Vol 1, No. 3, pp. 44-52; 1999.

[THP04] S. Tata, R. A. Hankins, J. M. Patel. **"Practical Suffix Tree Construction".** Proceedings of the 30th VLDB Conference, pp 36-47, 2004.

[TKR03] I. Tabus, G. Korodi, J. Rissanen. **"DNA sequence compression using the normalized maximum likelihood model for discrete regression".** Proceedings of the Data Compression Conference. Pp. 253–262. 2003

[WB06] I. H. Witten and T. Bell. **"The Calgary/Canterbury text compression corpus"**. Disponível via ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus. Acessado em dezembro de 2006.

[WI+87] Witten, Ian H., e outros. **"Arithmetic Coding for Data Compression".** Communications of the ACM, 30(6) pp.520-540.1987.

[WIT06] **"Teoria da Informação - Compressão de dados".** Disponivel via http://en.wikipedia.org/wiki/Information_theory. Visitado em dezembro de 2006.

[WKH+00] Till Westmann, Donald Kossmann, Sven Helmer e Guido Moerkotte. **"The implementation and performance of compressed databases".** ACM SIGMOD Record . Vol. 29, pp. 55 - 67. 2000.

[WU05] Washington University BLAST Archives, **"WU-BLAST"**. Disponível via http://blast.wustl.edu/. Acessado em dezembro 2006.

[XML05] W3c Domain Architecture. **"XML".** Disponível via http://www.w3.org/XML/. Acessado em dezembro de 2006.

[XML05] Definição de DTDs para intercambio de informação, **"XML"**. Disponível via http://www.ncbi.nlm.nih.gov/data_specs/dtd. Acessado em agosto de 2005.

[ZHN+06] M. Zukowski, S. Heman, N. Nes, P. Boncz. **"Super-Scalar RAM-CPU Cache Compression"**. 22nd International Conference on Data Engineering (ICDE'06) pp. 59. 2006.

[ZL77] J. Ziv e A. Lempel. **"A Universal Algorithm for Sequential Data Compression"**. IEEE Transactions on Information Theory. Vol. IT-23, no. 3, pp. 337-343, 1977.

APÊNDICE A - Características das biosseqüencias

As características de todo ser vivo são determinadas por um conjunto de fatores hereditários denominado de genótipo. Dado um organismo, cada célula armazena uma cópia desse conjunto de informações a qual denominamos de genoma. O genoma de qualquer organismo é composto por uma seqüência DNA, a qual é formada por um encadeamento de quatro nucleotídeos representados pelos caracteres A, T, C e G. O DNA é a fonte básica da informação genética. Um gene, a entidade básica fundamental de hereditariedade, é tipicamente uma seqüência especifica de nucleotídeos que carrega a informação requerida para sintetizar uma proteína ou, em alguns casos, um RNA.

Uma seqüência é uma sucessão de caracteres de um determinado alfabeto. As biosseqüencias podem ser de seqüências de nucleotídeos ou de proteínas. As seqüências de proteínas podem ser obtidas a partir de seqüências de nucleotídeos, já que os aminoácidos são traduzidos a partir de triplas de nucleotídeos[LC00].

As seqüências de nucleotídeos são construídas com o alfabeto Σ = {A, C, T, G}. Na prática, além das 4 letras citadas, existem 11 letras adicionais para descrever as possíveis ambigüidades encontradas na montagem das seqüências, como pode ser visto na Tabela 21 abaixo [BKY03].

Símbolo	Nucleotídeo	Nome
R	A ou G	PuRina
Y	C ou T	PYrimidina
W	A ou T	Limites fracos de hidrogênio
S	G ou C	Limites fortes de hidrogênio
K	G ou T	
M	A ou C	
B	C, G, ou T	não A
D	A, G, ou T	não C
H	A, C, ou T	não G
V	A, C, ou G	não T
N	A, C, G, ou T	Qualquer base (Any)

Tabela 21- Codificação para ambigüidades na leitura de nucleotídeos

O alfabeto das seqüências de proteínas é composto de 20 letras que representam os possíveis aminoácidos que as constituem. Além das 20 letras, existem mais três códigos usados pelos biólogos no seqüenciamento de proteínas. O código 21 identifica o par de aminoácidos Asparagina/Ácido Aspartâmico, e o código 22, o par Glutamina/Ácido Glutâmico. O último código, 23, representa todos os aminoácidos. Na **Tabela 22** abaixo é mostrada a lista de aminoácidos e codificação correspondente [BKY03].

Código	Símbolo	Abreviação	Aminoácido
1	A	Ala	Alanina
2	C	Cys	Cisteína
3	D	Asp	Ácido aspártico (aspartâmico)
4	E	Glu	Ácido Glutâmico
5	F	Phe	Fenilalanina
6	G	Gly	Glicina
7	H	His	Histidina
8	I	Ile	Isoleucina
9	K	Lys	Lisina
10	L	Leu	Leucina
11	M	Met	Metionina
12	N	Asn	Asparagina
13	P	Pro	Prolina
14	Q	Gln	Glutamina
15	R	Arg	Arginina
16	S	Ser	Serina
17	T	Thr	Treonina
18	V	Val	Valina
19	W	Trp	Triptofano
20	Y	Tyr	Tirosina
21	B	Asx	Asparagina/Ácido Aspartâmico
22	Z	Glx	Glutamina/Ácido Glutâmico
23	X	Xaa	Qualquer aminoácido

Tabela 22 - Lista de aminoácidos

Apesar das seqüências de proteínas serem unidimensionais, estas tem forma tridimensional. A representação como uma sucessão de aminoácidos, por exemplo, MLVGSRA , é conhecida como estrutura primária de proteína.

Para representar a estrutura secundária de uma proteína, definido como tipos de estruturas que são encontradas de forma repetitiva em uma proteína, existe um código chamado DSSP, *Dictionary of Protein Secondary Structure*, onde um código de uma letra é associado a cada tipo de estrutura.

A estrutura terciária é a forma completa de uma proteína. É chamada de estrutura nativa ou de conformação. Um dos objetivos da biologia computacional é a predição da estrutura tridimensional de uma proteína a partir de sua estrutura primária. Existe ainda a estrutura quaternária, quando uma proteína é na verdade uma montagem de mais de uma proteína, e este tipo de estrutura se refere à organização destas proteínas. Estes tipos de estrutura são estudados em uma área da Biologia conhecida como Biologia Estrutural.

Propriedades de uma biosseqüência

O DNA apresenta uma estrutura em hélice dupla. Nesta hélice as duas cadeias de DNA estão unidas entre si por ligações formadas entre bases opostas nas duas cadeias ou fitas. O pareamento é muito específico: a base Adenina(A) pareia somente com a base Timina (T), enquanto que a base Guanina (G), pareia com a Citosina (C). Por isso, o número de bases A é igual ao de T, assim como o número de bases C será sempre igual ao de G. Assim, a seqüência de bases de uma fita é complementar à da outra: dada a seqüência numa fita, a seqüência de bases da outra estará prontamente determinada.

As duas fitas são antiparalelas, ou seja, elas "correm" em direções opostas e são replicadas também em direções opostas.

Para nomear as extremidades das fitas foi criada uma nomenclatura baseada na estrutura química das seqüências.

As seqüências de DNA têm marcadores de inicio, chamado de 5’ (5-primo) e final, conhecido como 3’(3-primo). As seqüências são mostradas como na leitura normal de um texto ocidental, da esquerda para a direita, ou seja, a

seqüência é copiada do marcador 5' ao marcador 3'. Assim a ordem entre os caracteres define a posição relativa de cada nucleotídeo.

Para seqüências de proteínas a convenção adotada é N-Terminus, para o inicio da seqüência, e C-Terminus para o final [BKY03]. Estas denominações provêem da estrutura de uma cadeia de proteínas, onde o N-terminus se refere ao grupo de amino (NH2) presente em uma extremidade da cadeia, e C-terminus ao grupo de carboxil(COOH), presente na outra extremidade.

O inicio de uma seqüência de proteínas é marcado sempre pelo aminoácido M (metionina). Porém como toda regra universal da biologia, esta regra pode ter exceção.

Apesar de que é possível identificar nas seqüências regiões com certos padrões de caracteres, sendo esta uma das tarefas dos pesquisadores, não há como prever, a partir de um caractere dado, o próximo caractere na seqüência como em um texto em algum idioma, por exemplo.

O tamanho medido em número de bases, das seqüências de DNA e proteínas é bastante variável, dependendo do organismo e o que a seqüência catalogada representa. Por exemplo, o genoma humano tem aproximadamente 3.5 bilhões de bases[PS91], o genoma da *Drosophila melanogaster*, uma mosca, tem 1.2 milhões de bases. Por outro lado, seqüências parciais podem ser armazenadas, ou seja, ser submetidas a repositórios de seqüências antes da conclusão do projeto de seqüenciamento.

Para fins práticos, dados de bancos públicos de seqüência, como o GenBank[Ncb05c], mostram que na média as seqüências têm 700 bases (nucleotídeos), podendo variar até 300.000 bases. Seqüências maiores a 300.000 bases, são separadas em vários arquivos.

APÊNDICE B – Operações sobre Biosseqüencias

Comparação de seqüências

O alinhamento entre seqüências é um arranjo entre duas ou mais seqüências visando descobrir o grau de similaridade entre estas. As seqüências são dispostas umas sobre as outras de maneira a que as colunas respectivas contenham caracteres idênticos ou similares. Se necessários buracos podem ser inseridos. Algumas restrições devem ser respeitadas, como por exemplo, um buraco não pode ser alinhado com outro buraco, ou ainda, buracos não podem ser inseridos no inicio e final das seqüências. Na Figura 30, podemos ver um exemplo de alinhamento encontrado em [wikipedia]. Os traços (-) representam a inserção de buracos.

```
tcctctgcctctgccatcat---caaccccaaagt
|||| ||| ||||| |||||   ||||||||||||
tcctgtgcatctgcaatcatgggcaaccccaaagt
```

Figura 30 - Alinhamento entre seqüências

Sendo assim, as operações que podem ser realizadas em um alinhamento são: inserção de buracos, alinhamento de letras idênticas e alinhamento de letras diferentes.

Para determinar o grau de similaridade, é definida uma pontuação para cada operação sendo geralmente um valor positivo para letras idênticas, negativo ou zero para letras diferentes, negativo para inserção de um buraco e outro valor negativo para estender um buraco já introduzido. Posteriormente essas pontuações são somadas e tal soma determina o grau de similaridade entre as seqüências.

O alinhamento pode ser global ou local. No alinhamento global as seqüências são comparadas na totalidade da sua extensão, já no local, subconjuntos de caracteres de seqüências são alinhados. Por exemplo, poderíamos alinhar os caracteres das posições 10-20 da primeira seqüência com

os caracteres da posição 100-110 da segunda. Muitas vezes, as técnicas de alinhamento global combinam técnicas de alinhamento local.

No contexto biológico, como já foi dito anteriormente, o alinhamento é usado para medir o grau de similaridade entre duas ou mais seqüências, tendo cada uma das operações uma interpretação biológica. A inserção de buracos, alinhamentos de bases diferentes são entendidos como conseqüência da evolução dos organismos e de alguns fenômenos como a mutação, por exemplo [LC00].

A aplicação mais importante na biologia decorrente do alinhamento é a identificação de seqüências com estrutura ou função desconhecida. Ao encontrar um alinhamento entre duas seqüências pode-se admitir algum ancestral comum entre estas.

Na comparação de seqüências os pesquisadores estão interessados em encontrar o melhor alinhamento entre duas seqüências, o que muitas vezes é uma tarefa difícil do ponto de vista computacional devido ao tamanho das seqüências. Por outro lado, o sucesso da busca está fortemente relacionado ao esquema de pontuação utilizado, conhecido como matriz de pontuação , que reflete o conhecimento biológico.

Existem vários algoritmos baseados na técnica de programação dinâmica que resolvem o problema do alinhamento. O de Needleman-Wunsch, que realiza alinhamentos globais, Smith-Waterman, para alinhamentos locais, e o Framesearch, que é uma extensão do Smith-Waterman para alinhar uma seqüência de proteína com uma de nucleotídeo[LC00].

Baseadas nestes algoritmos foram desenvolvidas várias ferramentas para comparar seqüências. As seqüências são fornecidas como entrada no formato requerido por cada ferramenta e embora existam diferenças entre os critérios de implementação de cada uma, as seqüências deverão ser lidas totalmente para que a comparação possa gerar um resultado biologicamente válido. Também deve ser respeitada a ordem relativa entre os caracteres que forma a cadeia representando as seqüências.

Na prática, com bancos de seqüências cada vez maiores, as soluções baseadas na programação dinâmica, apesar de garantirem a solução ótima, podem ser muito caras, como no caso do algoritmo Smith-Waterman que para

alinhar duas seqüências de longitudes m e n respectivamente tem complexidade de tempo e espaço na ordem O(nm).

Assim, um novo algoritmo que combina métodos estatísticos para acelerar o alinhamento com o algoritmo padrão de Smith-Waterman se tornou muito popular. Este algoritmo é utilizado na ferramenta BLAST. O BLAST recebe como entrada uma seqüência de consultas e uma base de seqüências, contra a qual será comparada a primeira seqüência, e utiliza o mesmo mecanismo de Smith-Waterman para determinar a similaridade entre pequenas regiões das seqüências. Porém para decidir se um alinhamento destas subseqüências deve ser estendido na procura do melhor alinhamento, o BLAST utiliza algumas heurísticas que limitam a extensão a alinhamentos que alcançam um marco ou pontuação mínima. Com isso, o BLAST aumenta o desempenho na comparação, mas perde na sensibilidade dos resultados.

Geração de Anotações

As anotações representam o resultado de uma das atividades mais importantes dos projetos de seqüenciamento de genoma. É a interpretação e/ou relação que os pesquisadores da área conseguem estabelecer entre os dados obtidos e fontes de dados já existentes e outros experimentos.

O resultado de um novo seqüenciamento pode ser armazenado em uma fonte de dados pública ou em repositório próprio associado ao experimento. Primeiramente é comum que os biólogos registrem dados gerais associados ao experimento, como máquinas e características dos materiais utilizados, e em seguida, executem alguns programas sobre as seqüências obtidas como por exemplos os programas que ajudam a descobrir similaridades com seqüências de outros organismos, como citados na seção anterior. Assim são geradas novas anotações, resultantes da análise que os pesquisadores fazem sobre as saídas de tais programas.

As anotações podem ser classificadas em manuais e automáticas. As anotações automáticas são obtidas das fontes de dados externas e dos resultados da execução dos aplicativos. Além dos programas que buscam similaridade, existem programas que descobrem padrões importantes, outros que indicam a possibilidade da presença de um gene, as repetições, etc. Todos estes programas geram resultados que são anotações automáticas[LSC03].

As anotações manuais são feitas pelas comunidades de pesquisadores. Muitas das anotações dependem de programas executados sobre as seqüências. Um exemplo de um dos programas mais utilizados é o BLAST. Ao executar o BLAST e descobrir um alinhamento entre a nova seqüência contra uma fonte de dados, os pesquisadores precisam registrar o fato, associando a anotação a essa região da seqüência. Esses dados podem ser posteriormente consultados e inclusive atualizados por outros pesquisadores interessados no domínio.

Os dados registrados nas anotações variam de acordo ao esquema e funcionalidades de cada sistema de anotação. Uma grande parte dos sistemas de anotações disponíveis permite registrar dados de acordo as características definidas na tabela de funcionalidades desenvolvida pelo NCBI.

A maioria dos sistemas de anotações também conta com um repositório tipo data warehouse seguindo um modelo de dados próprio onde podem ser armazenadas as saídas de alguns programas, as anotações e também o resultado do seqüenciamento. Algumas ferramentas são mais flexíveis e permitem que as anotações sejam feitas mesmo antes do seqüenciamento ser concluído. Deste modo, algumas anotações e resultados podem ser revistos ou atualizados. O Bionotes[LMM+04], por exemplo, permite que várias versões dos dados sejam mantidas, sendo possível realizar a análise dos dados durante a evolução do seqüenciamento de um genoma dado.

Em [LSC03] podem ser encontradas descrições abreviadas dos sistemas de anotações mais utilizados, inclusive apresentando um quadro comparativo.

Embora uma anotação possa ser vista como um novo tipo de dado biológico que pode inclusive ser utilizado em outras operações e serem consultadas aleatoriamente pelos pesquisadores, é importante notar que a sua origem está estreitamente ligado à análise das biosseqüencias.

Montagem de fragmentos

O seqüenciamento de DNA é um processo que determina a ordem dos nucleotídeos em uma amostra. Independente da técnica, utilizando uma máquina automatizada ou por um método mais manual, a maior sub-cadeia de DNA com

qualidade que pode ser determinada em um procedimento em laboratório possui cerca de 600 a 700 bases[LBC03].

A técnica mais utilizada para seqüenciamento hoje em dia é o shotgun, que a partir do DNA que está sendo seqüenciado e alguns reagentes, gera fragmentos de DNA de tamanhos diferentes segundo as bases identificadas e as marcas de iniciador e terminador.

Desta forma, para sequenciar um genoma inteiro, o DNA precisa ser dividido em vários fragmentos pequenos que são seqüenciados individualmente. A partir de vários fragmentos de seqüências, busca-se reconstituir o trecho de DNA do qual esses fragmentos provieram através de comparações entre eles. Este problema é conhecido como montagem de fragmentos ou sequence assembly. Existem vários métodos disponíveis, e cada um apresenta vantagens e desvantagens[LBC03].

Existem diferentes implementações de algoritmos de montagem de fragmentos porém quase todos seguem a mesma idéia baseada em três passos: detecção de sobreposição, layout dos fragmentos e decisão da seqüência consenso e é conhecida como o paradigma overlap-layout-consensus, baseado na teoria de grafos [PST+83, PSU84].

Durante o seqüenciamento podem ocorrer erros de leitura. Para diminuir os erros, várias cópias de um fragmento são geradas e seqüenciadas. O processo para obtenção de cópias de um fragmento é chamado de clonagem.

O primeiro passo no processo de montagem de fragmentos consiste em descobrir, para cada par de fragmentos, quanto o sufixo da primeira casa-se com o prefixo da segunda, baseado no principio de que dois fragmentos que se sobrepõem têm maior probabilidade de terem sido originados na mesma região da seqüência. Devido a possíveis erros de leitura das bases, para este passo é adequado utilizar técnicas de casamento aproximado, como na operação de comparação de seqüências para busca de similaridade. Sendo assim, este problema se resume a encontrar o sufixo de S1, sendo S1 um primeiro fragmento, e o prefixo de S2, outro fragmento, cuja similaridade seja a maior entre todos os pares de sufixos e prefixos de S1 e S2. Uma heurística parecida com a do BLAST poderia encontrar tais pares entre fragmentos vizinhos.

Novamente erros podem ocorrer: as seqüências podem conter muitas regiões repetidas, e assim, similaridades encontradas em alguns casamentos não necessariamente indicar fragmentos vizinhos; e algumas regiões não serem “cobertas”.

O segundo passo utiliza os pares de fragmentos alinhados na primeira etapa. Começa-se pelo primeiro par alinhado, cujos, sufixo e prefixo, correspondentes se sobrepõem, com maior pontuação. Depois, o próximo par com maior pontuação é escolhido e intercalado também. Como resultado pode-se obter uma das figuras mostradas abaixo, com três fragmentos intercalados ou com pares de fragmentos intercalados dois a dois. Assim sucessivamente vai sendo formada uma estrutura de alinhamento, como mostrado na Figura 31.

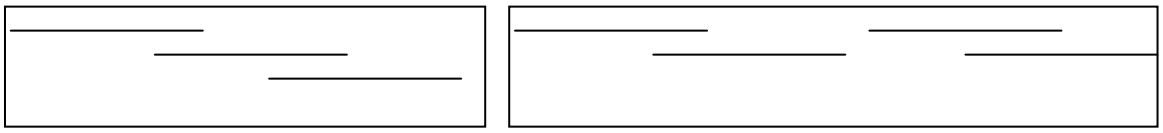

Figura 31- Exemplo de intercalação na montagem de fragmentos

Na etapa de consenso, é construído um alinhamento múltiplo de todos os fragmentos alinhados na etapa anterior e é inferida a partir deste alinhamento a seqüência original.

Os programas mais utilizados são CAp3[HM99], Prhap[Gre05], TIGR Assembly[SWA+95]. Algumas destas ferramentas são conhecidas como assembler, já que atendem a etapa de montagem e funciona em conjunto com outras para os passos prévios, como o Phrap que geralmente é utilizado com o Phred.

APÊNDICE C – Operações sobre Biosseqüencias

Como as biosseqüências são armazenadas

As biosseqüências ao serem armazenadas recebem uma identificação única e passam a fazer parte de um banco de dados de biosseqüências, público ou privado. A partir daí podem ser usadas como informação para catalogar outras seqüências geradas, ser analisadas com outras ferramentas ou comparadas com seqüências de outros repositórios.

Os principais bancos de biosseqüências existem desde a década passada, e estão em constante crescimento. Por exemplo, o SwissProt [SWI05], um banco curado4 de proteínas, que em 2001 (Release 40) registrava 101.602 entradas, atualmente contém 194.317 seqüências representando um crescimento de 90% somente nos últimos 4 anos. No site do GenBank [Ncb05c] é contabilizado atualmente5 quase 100 milhões de seqüências com 90 bilhões de bases, sendo que o banco vem dobrando de tamanho a cada 18 meses desde o seu inicio[Ncb05c]. Existem outros bancos públicos como o DDBJ [DDB05] e EMBL[EBI05]. Uma descrição dos bancos de seqüências mais populares pode ser encontrada neste apêndice.

A maioria dos bancos de dados está disponível em arquivos texto semi-estruturados. Este formato foi adotado inicialmente pela possibilidade de ser manipulado tanto pelas máquinas quanto pelas pessoas. Porém, dada a diversidade de formatos e códigos, e também pelo volume de dados, isto acaba sendo uma difícil tarefa para os pesquisadores da área hoje em dia.

Alguns dos formatos mais usados são ASN.1 [ASN05], XML [XML05] e FASTA [PL88]. Também existem bancos construídos em gerenciadores de bancos de dados relacionais como o Oracle [SCD+05], PostgreSQL [Bla05] e MySQL[GOD05] que incluem novos tipos de dados e operadores na tentativa de

[4] Os arquivos curados são o resultado da comparação com outros arquivos na tentativa de encontrar seqüências equivalentes, e em caso positivo, incluem anotações no cabeçalho são adicionadas para indicar os identificadores das seqüências que estarão representados ali.

[5] Informação obtida no *site* do NCBI em julho de 2005.

dar tratamento adequado às biosseqüências. Mais detalhes sobre os formatos de arquivo usados para persistir as biosseqüências podem ser encontrados no Apêndice C e em [FAB05].

Vale ressaltar também, que as seqüências podem ser armazenadas com formatos específicos ou intermediários para execução de alguns aplicativos. Um exemplo importante é o pré-processamento necessário para a execução do BLAST, feito através da ferramenta FORMATDB [For05]. O FORMATDB cria arquivos binários a partir de um arquivo de seqüências em formato FASTA ou ASN.1 contra o qual será comparado a seqüência de consultas durante a execução do BLAST. Três arquivos são gerados: um com seqüências, outro com a informação do cabeçalho identificador de cada seqüência, e o terceiro com uma estrutura de indexação para o acesso aleatório aos arquivos anteriores. Para execução da família FASTA, os dados também são estruturados de acordo ao formato FASTA.

Bancos de biosseqüencias

É difícil estimar o número de banco de dados biológicos existentes. Hoje em dia existem não somente os bancos de dados de seqüências de nucleotídeos (DNA) e de aminoácidos (proteínas), mas também inúmeros outros com informações bem específicas, como organismos especiais, biossequências específicas, como tRNA e rRNA, enzimas, mutações, famílias de biossequências (filogenia), etc. Além disso, já existem bancos que guardam estruturas tridimensionais das biossequências, como por exemplo o PDB[SLL00].

O GenBank[Ncb05c] é hoje um dos mais importantes repositórios de seqüências de nucleotídeos. É usado como referência no sentido de verificar se uma dada seqüência já está catalogada. Sendo assim, será apresentada nesta seção uma descrição comparativa do GenBank, EMBL e DDBJ, como exemplo de repositório de seqüências e nas seções seguintes será dada ênfase ao modo de persistência para biosseqüências, referenciando quando necessário o repositório correspondente. Para obter mais informações sobre os principais bancos de dados biológicos disponíveis, os trabalhos de Navathe e Patil [NP04], e de Seibel, Lemos e Lifschitz[SLL00] .

- **GenBank/EMBL/DDBJ**

Para descrever as características destas importantes fontes de dados biológicos foram analisadas as seguintes versões:

  - GenBank Release 132,
  - EMBL Release 72,
  - DDBJ Release 51

Os bancos de seqüências públicos GenBank, EMBL e DDBJ compartilham seqüências[Ncb05c]. Existe semelhança entre seus esquemas de armazenamento e também utilizam alguns formatos próprios para transferência, como ASN.1, XML, FASTA.

O formato dos arquivos dos bancos de seqüências GenBank e DDBJ é bastante similar. Cada coluna ou campo é iniciado por um termo que define o tipo de informação contida na linha, como por exemplo, SOURCE, que indica o nome do organismo comumente usado na literatura. Alguns campos são opcionais e outros obrigatórios.

No arquivo do EMBL a organização é um pouco diferente. Os termos são representados como códigos de duas letras que também aparecem no inicio de cada linha.

No EMBL, o item SQ, denominado ORIGIN no GenBank e DDBJ, por exemplo, corresponde ao inicio da seqüência de nucleotídeos. No EMBL a posição ou tamanho atual da seqüência é mostrado no final de cada linha, e nos demais no inicio.

O campo FEATURES ou características, FT (Feature Table) no EMBL, indica o inicio de uma tabela com informação correspondente a uma proteína, moléculas de RNA ou alguma informação biológica reconhecida experimentalmente. Cada referencia nesta tabela, composta da posição inicial e final, corresponde a uma subseqüência na seqüência de nucleotídeos. A tradução dos nucleotídeos em seqüências de aminoácidos também é armazenada na tabela FEATURES.

Uma característica é uma palavra única que indica uma função ou papel associado a uma região de uma seqüência. Para cada característica um código de uma lista pré-definida, poderá ser usado para adicionar informação. A localização da subseqüência a qual corresponde a característica reportada pode ser expressa pela posição de uma única base, da base inicial e final, ou ainda operadores mais complexos podem ser usados como *join*, *complements*. Por exemplo, *join*(12..78,134..202) indica que a primeira região, localizada entre as posições 12 e 78 deve ser unida à região entre 134 e 202 para formar a subseqüência.

Como descrito anteriormente, os dados armazenados no GenBank, EMBL e DDBJ estão disponíveis em arquivos texto, mas atualmente gerenciadores de banco de dados relacionais como o Oracle são adotados como repositório, como no caso do EMBL[MR02], cujo esquema relacional está sendo estendido para outros bancos como o SwissProt e TrEMBL[SWI05].

A Figura 32 a seguir mostra um exemplo de um arquivo extraído do repositório DDBJ.

No inicio do arquivo são colocados dados como o nome atribuído à seqüência, indicado pelo atributo DEFINITION, o identificador da seqüência, ACCESSION, seguido pelo código indicando a versão da seqüência, VERSION, já que uma seqüência pode ser mantida no banco com várias versões, a cada submissão é controlado se a seqüência já existe. Em tal caso, uma nova versão é criada. Outros atributos como o organismo à qual a seqüência pertence, ORGANISM, número de referencias feitas a seqüência, os nomes dos autores, e o título dado à seqüência, aparecem em seguida. Logo é incluída a tabela de características, FEATURES, que contém algumas características da seqüência, relacionadas aos códigos de característica correspondente (source, gene, CDS) e também a tradução da seqüência de nucleotídeos em proteínas. Finalmente, a seqüência é escrita em várias linhas.

```
LOCUS       HSCDK2MR                1476 bp    mRNA    linear   PRI 15-JAN-1992
DEFINITION  H.sapiens CDK2 mRNA.
ACCESSION   X61622
VERSION     X61622.1  GI:29848
KEYWORDS    CDK2 gene; cell cycle regulation protein; cyclin A binding; protein
            kinase.
SOURCE      Homo sapiens (human)
  ORGANISM  Homo sapiens
            Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
            Mammalia; Eutheria; Primates; Catarrhini; Hominidae; Homo.
REFERENCE   1  (bases 1 to 1476)
  AUTHORS   Elledge,S.J. and Spottswood,M.R.
  TITLE     A new human p34 protein kinase, CDK2, identified by complementation
            of a cdc28 mutation in Saccharomyces cerevisiae, is a homolog of
            Xenopus Eg1
  JOURNAL   EMBO J. 10 (9), 2653-2659 (1991)
  MEDLINE   91330891
REFERENCE   2  (bases 1 to 1476)
  AUTHORS   Elledge,S.J.
  TITLE     Direct Submission
  JOURNAL   Submitted (28-NOV-1991) S.J. Elledge, Dept. of Biochemistry, Baylor
            College of Medicine, 1 Baylor Place, Houston, TX 77030, USA
FEATURES             Location/Qualifiers
     source          1..1476
                     /organism="Homo sapiens"
                     /db_xref="taxon:9606"
                     /clone="pSE1000"
                     /cell_line="EBV transformed Human peripheral lymphocyte
                     (B-cell)"
                     /clone_lib="lambda YES-R cDNA library"
     gene            1..1476
                     /gene="CDK2"
     CDS             1..897
                     /gene="CDK2"
                     /function="protein kinase"
                     /note="cell division kinase. CDC2 homolog"
                     /codon_start=1
                     /protein_id="CAA43807.1"
                     /db_xref="GI:29849"
                     /db_xref="SWISS-PROT:P24941"
                     /translation="MENFQKVEKIGEGTYGVVYKARNKLTGEVVALKKIRLDTETEGV
                     PSTAIREISLLKELNHPNIVKLLDVIHTENKLYLVFEFLHQDLKKFMDASALTGIPLP
                     LIKSYLFQLLQGLAFCHSHRVLHRDLKPQNLLINTEGAIKLADFGLARAFGVPVRTYT
                     HEVVTLWYRAPEILLGSKYYSTAVDIWSLGCIFAEMVTRRALFPGDSEIDQLFRIFRT
                     LGTPDEVVWPGVTSMPDYKPSFPKWARQDFSKVVPPLDEDGRSLLSQMLHYDPNKRIS
                     AKAALAHPFFQDVTKPVPHLRL"
BASE COUNT      368 a    372 c    351 g    385 t
ORIGIN
        1 atggagaact tccaaaaggt ggaaaagatc ggagagggca cgtacggagt tgtgtacaaa
       61 gccagaaaca agttgacggg agaggtggtg gcgcttaaga aaatccgcct ggacactgag
      121 actgagggtg tgcccagtac tgccatccga gagatctctc tgcttaagga gcttaaccat
      181 cctaatattg tcaagctgct ggatgtcatt cacacagaaa ataaactcta cctggttttt
      241 gaatttctgc accaagatct caagaaattc atggatgcct ctgctctcac tggcattcct
      301 cttcccctca tcaagagcta tctgttccag ctgctccagg gcctagcttt ctgccattct
      361 catcgggtcc tccaccgaga ccttaaacct cagaatctgc ttattaacac agaggggccc
      421 atcaagctag cagactttgg actagccaga gcttttggag tccctgttcg tacttacacc
      481 catgaggtgg tgaccctgtg gtaccgagct cctgaaatcc tcctgggctc gaaatattat
      541 tccacagctg tggacatctg gagcctgggc tgcatctttg ctgagatggt gactcgccgg
      601 gccctgttcc ctggagattc tgagattgac cagctcttcc ggatctttcg gactctgggg
      661 accccagatg aggtggtgtg gccaggagtt acttctatgc ctgattacaa gccaagtttc
      721 cccaagtggg cccggcaaga ttttagtaaa gttgtacctc ccctggatga agatggacgg
      781 agcttgttat cgcaaatgct gcactacgac cctaacaagc ggatttcggc caaggcagcc
      841 ctggctcacc ctttcttcca ggatgtgacc aagccagtac cccatcttcg actctgatag
      901 ccttcttgaa gccccgacc ctaatcggct caccctctcc tccagtgtgg gcttgaccag
     …
```

Figura 32 - Exemplo de uma seqüência guardada no DDBJ

A seguir será mostrado um resumo das principais estruturas de persistência em memória secundária para biosseqüencias.

Persistência em arquivos texto (*flat files*)

Fazendo uma abstração do contexto biológico, uma biosseqüência pode ser tratada como uma longa cadeia de caracteres. Este fato orientou a escolha por guardá-las em arquivos texto logo no inicio da geração dos primeiros dados.

Nesta seção será apresentada uma descrição dos principais formatos de arquivos utilizados hoje em dia para armazenamento de seqüências biológicas.

ASN.1

Por definição, o formato ASN.1 é "uma notação formal utilizada para descrever dados a serem transmitidos mediante protocolos de telecomunicações independentemente da linguagem e representação física dos mesmos". [ASN05].

Não obstante, apesar de ser originalmente criada para descrever mensagens transmitidas pelos protocolos OSI (Open Systems Interconnection), o formato ASN.1 foi adotado em diversas áreas desde operações bancárias, controle de tráfego aéreo, telefonia celular, transmissão de áudio e vídeo pela Internet e até saúde e genética.

Os arquivos ASN.1 podem ser persistidos como texto ou arquivos binários.

Na Bioinformática, o ASN.1 foi adotado como formato para intercambio ou transferência de dados entre os diferentes bancos de seqüências de DNA, principalmente pelo NCBI - criador do GenBank. O NCBI usa ASN.1 para guardar e disponibilizar dados como seqüências de nucleotídeos, proteínas, estruturas de proteínas, genomas e registros de publicações da base MEDLINE6. Atualmente a maioria dos bancos de seqüências gera saída e aceita submissões no formato ASN.1.

[6] Fonte de informação bibliográfica de biomedicina.

Em arquivos no formato ASN.1 é possível associar tipos de dados simples como INTEGER, BOOLEAN, e também compostos como SEQUENCE, SET. Os tipos SEQUENCE (ou SEQUENCE OF) e SET (ou SET OF) são listas onde os elementos são separados por vírgula. São denotados pela etiqueta SEQUENCE {}" e "SET {}", respectivamente, e cada item da lista está representado como um par com um identificador e um valor. À diferença de SET, na lista SEQUENCE existe uma ordem entre os elementos, a ordem em que estes aparecem na lista.

A Figura 33 é um exemplo de uma seqüência registrada no GenBank no formato ASN.1. No atributo *descr* está contida a descrição correspondente à seqüência. A descrição está composta de outros atributos como *source* com dados correspondentes à origem da seqüência, como *org*, representando o organismo, *orgname*, nome e família do organismo, entre outros.

```
Seq-entry ::= set {
  level 1 ,
  class nuc-prot ,
  descr {
    title "Human protein tyrosine kinase TEC (tec) gene, partial cds, and
 tyrosine kinase TXK (txk) gene, and translated products" ,
    source {
      org {
        taxname "Homo sapiens" ,
        common "human" ,
        db {
          {
            db "taxon" ,
            tag
              id 9606 } } ,
        orgname {
          name
            binomial {
              genus "Homo" ,
              species "sapiens" } ,
          lineage "Eukaryota; Metazoa; Chordata; Craniata; Vertebrata;
 Euteleostomi; Mammalia; Eutheria; Primates; Catarrhini; Hominidae; Homo" ,
          gcode 1 ,
          mgcode 2 ,
          div "PRI" } } ,
.
.
```

Figura 33 - Arquivo ASN.1 representando uma seqüência registrada no GenBank.

Alguns aplicativos aceitam que as seqüências de entrada estejam guardadas em arquivos no formato ASN.1. Um exemplo é o BLAST que através do utilitário FORMATDB prepara as seqüências de entrada que podem estar neste formato [BKY03]. O BLAST também disponibiliza as saídas de sua execução em ASN.1.

O ASN.1 é um padrão maduro, adotado internacionalmente. A formalidade do modelo assegura a completitude e precisão na especificação dos dados, evitando interpretações ambíguas. É também independente de sistemas e linguagens, porém pode ser facilmente tratado em qualquer linguagem de programação.

XML

O XML vem sendo usado para representar dados em muitas áreas, inclusive na Biologia, como formato para intercâmbio de informação ou para representar resultados de programas de análise de dados biológicos como o BLAST e ainda para persistir alguns tipos de dados biológicos.

Atualmente existe uma grande demanda pela representação em XML. Comparado ao XML, o ASN.1 apresenta vantagens por ser fortemente tipado, à diferença do XML que trata todos os dados como texto, e pela economia de espaço e processamento do seu esquema de codificação binário, sendo uma vantagem para a transmissão de dados por exemplo. Uma das desvantagens do ASN.1 é o alto custo das ferramentas comerciais porém o NCBI, por exemplo, disponibiliza um conjunto de aplicativos conhecido como NCBI *Toolbox* que contém softwares para tratar registros no formato ASN.1.

Até mesmo aqueles que já tinham um formato próprio de representação de dados, estão migrando os dados para XML. Como exemplo destacam-se as fontes de dados externas EMBL, Genbank e DDBJ, PIR e SWISSPROT [LSC03].

Como exemplo da utilização do XML na Biologia pode-se citar o esquema definido pelo NCBI para disponibilizar os resultados de uma execução do BLAST. Na Figura 34 é mostrado parcialmente um exemplo de uma saída da execução do BLASTP[BYK03].

É parte da saída o nome do programa executado, no caso BLASTP, destacado pelo tag BlastOutput_program, seguido pela versão (BlastOutput_version), nome do banco de seqüências (BlastOutput_db) assim como a identificação e tamanho da seqüência de consulta, tags BlastOutput_query_def e BlastOutput_query_len, respectivamente, além da configuração dos parâmetros de execução, como a matriz de pontuação, valor para E (expect), valor de penalização e extensão de buracos (gap), e outros.

O esquema contém ainda, não mostrados na Figura 34, os alinhamentos encontrados com informação sobre os pares de segmentos de alta pontuação (HSPS- high score pairs) como, a pontuação obtida, inicio e fim da seqüência de consulta e da seqüência de comparação correspondente, número de buracos e o alinhamento propriamente. O documento com a definição completa (DTD – document type definition) pode ser encontrado em [XML05].


```
<?xml version="1.0"?>
<!DOCTYPE BlastOutput PUBLIC "-//NCBI//NCBI BlastOutput/EN" "NCBI_BlastOutput.dtd">
<BlastOutput>
  <BlastOutput_program>blastp</BlastOutput_program>
  <BlastOutput_version>blastp 2.2.5 [Nov-16-2002]</BlastOutput_version>
  <BlastOutput_reference>~Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro
A. Schaffer, ~Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997),
~&amp;quot;Gapped BLAST and PSI-BLAST: a new generation of protein database
search~programs&amp;quot;,  Nucleic Acids Res. 25:3389-3402.</BlastOutput_reference>
  <BlastOutput_db>HoxDB_custom.pep</BlastOutput_db>
  <BlastOutput_query-ID>lcl|QUERY</BlastOutput_query-ID>
  <BlastOutput_query-def>AAG39070 gi|11611819|gb|AAG39070.1| Hoxa-11 [Latimeria
chalumnae]</BlastOutput_query-def>
  <BlastOutput_query-len>202</BlastOutput_query-len>
  <BlastOutput_param>
    <Parameters>
      <Parameters_matrix>BLOSUM62</Parameters_matrix>
      <Parameters_expect>10</Parameters_expect>
      <Parameters_gap-open>11</Parameters_gap-open>
      <Parameters_gap-extend>1</Parameters_gap-extend>
      <Parameters_filter>m S</Parameters_filter>
    </Parameters>
  </BlastOutput_param>
```

Figura 34 - Esquema de uma saída em XML obtida da execução do BLAST.

Outro exemplo do uso de XML para representar dados biológicos é o sistema de anotações BioNotes[LMM+04]. Na prática, o sistema BioNotes possui um modelo de dados híbrido onde as entidades estão armazenadas em tabelas, que possuem colunas que representam informações biológicas de acordo com o modelo semi-estruturado. Nestas são armazenados os documentos, em XML, e

os esquemas destes documentos, em XML Schema. O banco de dados utilizado é o Oracle 9i, que armazena dados em XML em registros do tipo XMLType.

As vantagens do XML são inerentes às vantagens do paradigma de representação de dados semi-estruturados. Este tipo de representação é bastante adequado para a Biologia dada à natureza irregular dos dados biológicos e a necessidade de integração de várias fontes de dados. Ao mesmo tempo, XML está se tornando um padrão popular, com rápido crescimento de ferramentas abertas para tratamento deste tipo de dado.

Porém, como o XML não possui um esquema de codificação binária, como o ASN.1, o tamanho final da representação neste formato é bem maior que a representação equivalente em ASN.1. Como um interpretador de ASN.1 sempre conhece a estrutura dos dados é possível gerar uma codificação binária bastante compacta. Além disso, no ASN.1 é possível reutilizar um mesmo nome de tag para diferentes papéis no esquema. Por exemplo, se uma estrutura Pessoa tem um atributo nome, outra estrutura, no mesmo arquivo, Departamento, poderá ter também um atributo nome. Já o formato XML requer que os identificadores de atributos (*tags*) sejam únicos em um mesmo esquema.

Devido à falta de consenso por formato único, existem ferramentas para conversão entre os distintos formatos, como por exemplo, para tradução de arquivos ASN.1 para XML e vice-versa. No caso do NCBI, o aplicativo criado para transformar ASN.1 em XML, chamado asn2xml, usa o mesmo modelo de dados para ambos formatos, sendo que as estruturas são mapeadas seguindo a restrição de cada um.

FASTA

É um dos formatos de arquivo mais utilizados. Várias organizações desenvolveram programas para a sua criação e interpretação, sendo que algumas seções dentro do arquivo podem diferir quanto à sintaxe, já que cada organização acaba adotando a mais adequada no seu caso [LM03].

O arquivo contém uma primeira linha com um conjunto de valores que identificam e descrevem a seqüência que virá a seguir, sendo que um arquivo pode conter mais de uma seqüência. Os arquivos FASTA podem ter diferentes extensões como .fa, .mpfa. .fsa, e outras.

- Em geral o arquivo contém:

    - “>” Símbolo que marca inicio de uma entrada.

    - Primeira linha: começa com o identificador da seqüência e é seguido por uma descrição. O formato da linha pode variar já que várias organizações definiram sua própria sintaxe para a descrição, mas em geral, é sempre esperado na primeira linha o identificador da seqüência e o restante poderá ser tratado como uma cadeia de caracteres.

    - Seqüência de proteínas ou nucleotídeos organizada em várias linhas. Cada linha pode ter até 60, 70, 72 ou 80 caracteres.

No caso dos arquivos mantidos pelo NCBI, por exemplo, a sintaxe do cabeçalho depende do banco de dados de onde as seqüências foram derivadas. O cabeçalho é composto pelo identificador gi seguido de um número único para cada seqüência e um código abreviado que indica o repositório de origem da seqüência. Os dados seguintes variam de acordo a cada repositório e podem ser encontrados na documentação fornecida pelo NCBI. Contudo, geralmente é escrito o identificador original da seqüência e um nome abreviado dado à seqüência.

É mostrado na Figura 35 um exemplo de um arquivo FASTA. Esta entrada representa uma seqüência proveniente do banco de seqüências EMBL, indicada pela etiqueta emb, registrada originalmente em tal repositório com o código X61622.1 e correspondente a uma seqüência de RNA do homem.

```
>gi|29848|emb|X61622.1|HSCDK2MR H.sapiens CDK2 mRNA
ATGGAGAACTTCCAAAAGGTGGAAAAGATCGGAGAGGGCACGTACGGAGTTGTGTACAAAGCCAGAAACA
AGTTGACGGGAGAGGTGGTGGCGCTTAAGAAAATCCGCCTGGACACTGAGACTGAGGGTGTGCCCAGTAC
TGCCATCCGAGAGATCTCTCTGCTTAAGGAGCTTAACCATCCTAATATTGTCAAGCTGCTGGATGTCATT
CACACAGAAAATAAACTCTACCTGGTTTTTGAATTTCTGCACCAAGATCTCAAGAAATTCATGGATGCCT
CTGCTCTCACTGGCATTCCTCTTCCCCTCATCAAGAGCTATCTGTTCCAGCTGCTCCAGGGCCTAGCTTT
CTGCCATTCTCATCGGGTCCTCCACCGAGACCTTAAACCTCAGAATCTGCTTATTAACACAGAGGGGGCC
ATCAAGCTAGCAGACTTTGGACTAGCCAGAGCTTTTGGAGTCCCTGTTCGTACTTACACCCATGAGGTGG
TGACCCTGTGGTACCGAGCTCCTGAAATCCTCCTGGGCTCGAAATATTATTCCACAGCTGTGGACATCTG
GAGCCTGGGCTGCATCTTTGCTGAGATGGTGACTCGCCGGGCCCTGTTCCCTGGAGATTCTGAGATTGAC
CAGCTCTTCCGGATCTTTCGGACTCTGGGGACCCCAGATGAGGTGGTGTGGCCAGGAGTTACTTCTATGC
CTGATTACAAGCCAAGTTTCCCCAAGTGGGCCCGGCAAGATTTTAGTAAAGTTGTACCTCCCCTGGATGA
AGATGGACGGAGCTTGTTATCGCAAATGCTGCACTACGACCCTAACAAGCGGATTTCGGCCAAGGCAGCC
CTGGCTCACCCTTTCTTCCAGGATGTGACCAAGCCAGTACCCCATCTTCGACTCTGATAGCCTTCTTGAA
GCCCCCGACCCTAATCGGCTCACCCTCTCCTCCAGTGTGGGCTTGACCAGCTTGGCCTTGGGCTATTTGG
ACTCAGGTGGGCCCTCTGAACTTGCCTTAAACACTCACCTTCTAGTCTTAACCAGCCAACTCTGGGAATA
CAGGGGTGAAAGGGGGGAACCAGTGAAAATGAAAGGAAGTTTCAGTATTAGATGCACTTAAGTTAGCCTC
CACCACCCTTTCCCCCTTCTCTTAGTTATTGCTGAAGAGGGTTGGTATAAAAATAATTTTAAAAAAGCCT
TCCTACACGTTAGATTTGCCGTACCAATCTC
```

Figura 35 - Uma seqüência em um arquivo FASTA gerado pelo NCBI.

Uma das vantagens da simplicidade do formato FASTA é a facilidade de visualização e manipulação direta dos arquivos mediante um processador de texto comum e em linguagens do tipo script. Entretanto, por ser armazenado como um arquivo texto sem uso de nenhuma técnica de compactação, os arquivos FASTA podem ser grandes. Outra desvantagem é para tratar com arquivos deste tipo deve-se contar um interpretador próprio.

Persistência em gerenciadores de bancos de dados comerciais e específicos

A escolha por sistemas gerenciadores de banco de dados como meio para armazenar e gerenciar dados biológicos pode ser vista como um caminho natural já que devido aos avanços da tecnologia, houve um aumento da produção de biossequências e em conseqüência o volume de dados armazenados em arquivos texto cresceu muito.

A opção pelo uso de arquivos texto em relação a outros modos de persistência, apresenta desvantagens extensivamente já discutidas na literatura como em [NE00], como a validação das alterações, consultas, controles de acesso, devem ser feitos com auxilio de programas escritos para cada formato.

O gerenciamento de dados biológicos em banco de dados armazenados em sistemas de banco de dados poderia explorar as estruturas como gerente de memória deste tipo de sistema para minimizar o movimento de dados dos discos para a memória, permitir aplicação de filtros prévios e pós-processamento de conjunto de dados, além de garantir um ambiente seguro e de alta disponibilidade para os dados[SCD+05].

Atualmente os bancos de dados da Biologia utilizam sistemas de banco de dados relacional, sistemas orientados a objetos e ainda existem alguns gerenciadores específicos para a área.

No EMBL, SwissProt, TrEMBL, o Oracle foi adotado como repositório mas são utilizados arquivos texto para troca de dados [NP04].

Na maioria dos casos onde gerenciadores de banco de dados relacionais são utilizados como repositórios de seqüências (DNA ou proteínas),

independentemente do esquema construído, as seqüências são armazenadas como cadeias de caracteres ou arquivos texto são diretamente persistidos como objetos grandes (CLOB). Isto facilita a carga dos repositórios a partir de arquivos texto, porém o acesso aos dados é limitado aos operadores tradicionais.

O modelo relacional, apesar de maduro e bastante utilizado em outras áreas, não oferece um ambiente totalmente adequado para os dados biológicos. Estudos como [SLL00], relatam com detalhes os problemas para representar tais neste modelo.

Por outro lado, estes sistemas podem ser estendidos para incorporar estruturas e operadores da biologia. As estratégias comumente adotadas para combinar ferramentas especializadas com gerenciadores de banco de dados de propósito geral se resumem a: (1) adicionar operadores especializados a gerenciadores existentes; (2) integrar as fontes de dados.

Dentre estas estratégias, o Oracle 10g merece uma atenção em particular devido ao grande número de funcionalidades incluídas nesta versão para facilitar o trabalho dos pesquisadores na área de ciências da vida, principalmente no sentido de acesso a dados provenientes de diversas fontes, algo muito comum na biologia computacional.

O Oracle 10g incorpora o BLAST, versão NCBI 2.0, no banco de dados, permitindo a realização das operações disponíveis em todas as versões do BLAST como também consultas em linguagem SQL para pré-filtrar as seqüências ou ainda pós-processar os resultados obtidos[10].

As operações de busca de similaridade e alinhamentos são feitas através de consultas utilizando uma **"tabela função**" na cláusula FROM de uma sentença SQL. Este novo recurso, tabela-função, permite que porções de códigos possam ser executadas através de consultas em linguagem SQL. A função escolhida depende da operação desejada, por exemplo, BLASTN_ALIGN deve ser usada para buscar alinhamentos entre a seqüência de consulta de nucleotídeos e um banco de seqüências também de nucleotídeos. Da mesma maneira os resultados são disponibilizados em tabelas e estas podem ser consultadas via interface SQL.

As seqüências devem estar armazenadas no Oracle como um tipo CLOB, ou podem ser mantidas fora do banco sendo acessados através de mecanismos de comunicação com dados externos fornecidos pelo Oracle, como gateways, odbcs, uso de tabelas externas, etc.

Para executar o BLAST no Oracle não é necessário executar o formatdb, o aplicativo de preparação de arquivos específicos para o BLAST. A seqüência de consulta será passada para o servidor no formato CLOB e o banco de seqüências contra o qual será feita a comparação, como um objeto do tipo CURSOR. Este cursor contém pelo menos os identificadores de seqüência, representados como tipo de dados VARCHAR2, e as seqüências propriamente ditas, que também devem ser do tipo CLOB.

Todas as características[Ncb05c] das seqüências também podem ser registradas no banco de dados e utilizadas como filtros em consultas para execução do BLAST ou consultas aos resultados de um alinhamento, por exemplo. Como existem diferenças entre as estruturas dos bancos de seqüências, cada banco de seqüências ao ser incorporado ou transferido para o Oracle, poderá ter um esquema de dados diferente.

Outra estratégia semelhante foi colocada em pratica utilizando o gerenciador de dados PostgreSQL[Pos05]. Esta implementação chamada de BlastGres[Bla05] também incorpora o programa BLAST ao gerenciador de banco de dados. Além disto, foram criados novos tipos de dados para representar segmentos de seqüência, em conjunto com um novo tipo de índice para acelerar o acesso a uma região de uma seqüência e as características correspondentes. No entanto, nesta abordagem as seqüências são guardadas como cadeias de caracteres. A idéia é representar um segmento de seqüência com o tipo de dados range, números indicando o inicio e fim de um segmento de seqüência, e um tipo *location*, onde um valor de *range* é associado ao identificador de uma seqüência, podendo ainda utilizar um índice para realizar a busca de uma região de uma seqüência dada, ou seja, o atributo do tipo *location*.

Outro exemplo de sucesso é o sistema ACeDB[TDS99]. O ACeDB está baseado no modelo orientado a objetos mas conta com um módulo de gerenciamento de banco de dados, em um modelo flexível, projetado especificamente para manipular informações biológicas. No ACeDB (A Caenorhabditis elegans Data Base) são armazenados os resultados de projetos

de seqüenciamento e mapeamento de em larga escala. O ACeDB foi criado em parceria por pesqiusadores do Sanger Centre, Cambridge, e do CNRS, Montpellier.

O ACeDB representa internamente os dados em forma de árvore, em formato binário. A entrada e saída dos dados é feita via arquivos texto denominados “ACE files”, onde as informações são representadas de acordo com uma sintaxe específica, semelhante à XML. A seguir, é apresentado um exemplo de arquivo de entrada de dados do ACeDB.

Na Figura 36, é dado o exemplo onde é definida a sequência de nome ACT3 com título, referência à base EMBL e a seqüência de nucleotídeos.

```
// sequence definition
Sequence ACT3
Title ``C. elegans actin gene (3)''
Library EMBL CEACT3 X16798
// DNA (classe A)
DNA ACT3
aagagagacatcctcccgctcccttcccacacccacttgctcttttctat
```

Figura 36 - Exemplo de um arquivo de entrada para o sistema ACeDB

Outras propostas de persistência para biosseqüências

Ao idealizar soluções de persistência em memória secundária pode-se decompor o problema em dois: o primeiro para tratar da estrutura de dados adequada ao tipo de dado que se pretende guardar, e o segundo, para implementar os métodos de acesso de acordo com a natureza de processamentos mais comuns, como leituras seqüenciais e diretas ou aleatórias.

Nesta seção são apresentadas algumas propostas para armazenamento de biosseqüencias. Foram selecionadas as soluções mais recentes da literatura.

- **Árvores de Sufixo**

Dada uma cadeia de caracteres S de tamanho n, um sufixo é qualquer subcadeia si, onde i é uma posição na cadeia, 1≤ i ≤ n. Uma árvore de sufixos é uma estrutura de dados do tipo árvore construída sobre todos os sufixos de S [21]. Cada nó folha aponta para um sufixo da cadeia, e percorrendo a árvore é possível encontrar todas as subcadeias. Geralmente uma versão compactada é utilizada para representar a árvore.

Na Figura 37 é exibido um exemplo de árvore de sufixos construída para a seqüência ATTAGTACA$, onde $ indica o fim da cadeia. Em média para armazenar a árvore de sufixos será necessária uma área correspondente a nove vezes o tamanho da entrada. Sendo assim, para uma seqüência com 300.000 bases, seriam necessários um pouco mais de 2 MB. Este requisito pode variar de acordo a cada solução, uma vez que informações adicionais podem ser mantidas para cada nó em benefício de um melhor desempenho de algumas operações de acesso aos dados.

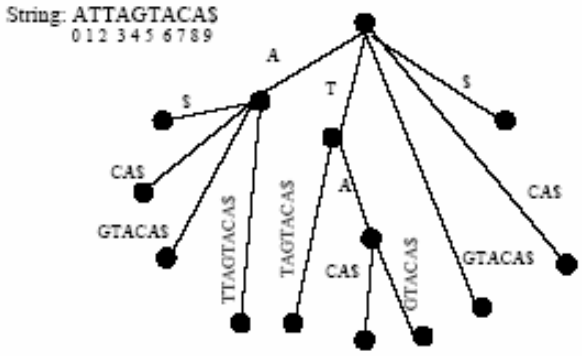

Figura 37 - Árvore de sufixo.

Muitas outras decisões de desenho da árvore podem ser tomadas. Por exemplo, a utilização ou não de arcos transversais chamados de suffix links, além dos arcos longitudinais. Esta escolha pode acarretar em uma operação mais cara de construção da árvore. No entanto, para obter-se um fragmento de seqüência aleatório, por exemplo, estas estruturas poderão ser exploradas para compor um caminho entre os nós em conjunto com os arcos longitudinais com tempos de execução bem melhores. Em [THP04] são apresentadas comparações entre soluções de busca em árvores de sufixo utilizando exemplos baseados nas operações mais comuns da biologia computacional.

Dado que operações de busca aproximadas são fundamentais na Biologia computacional, possibilitar este tipo de operação com desempenho razoável é um requisito para qualquer estrutura de armazenamento candidata nesta área. Árvores de sufixo são adequadas para execução destas operações, por isso estão sendo testadas e otimizadas para armazenamento de biosseqüencias.

Há vários algoritmos conhecidos para construir uma árvore de sufixo em tempo linear sobre o tamanho da cadeia de entrada, O(n), porém estas soluções precisam de muito espaço em memória. Como as seqüências de DNA podem ser muito longas, pode ser impossível construir a árvore em alguns casos.

Outros tipos de indexação

A indexação é um método para reduzir custos de processamento de consultas, que na Biologia podem ser bastante caras. No entanto, a aplicação nesta área da maioria das técnicas atuais de indexação aumenta a velocidade das consultas com a penalização da qualidade dos resultados. Sendo assim, novas estratégias, específicas para consultas às seqüências biológicas, estão sendo desenvolvidas. A seguir, serão descritas algumas.

A idéia do trabalho descrito em [CRD+04] é construir um índice a partir de uma seqüência de nucleotídeos. Os testes foram feitos criando este índice no SGBD relacional NCR Teradata e foi chamado de iBLAST, já que foi utilizado com a ferramenta BLAST. A seqüência é dividida em palavras de 16 bases, e a representação de cada letra do alfabeto das seqüências de nucleotídeos é representada da seguinte maneira: A= 00, C=01, G=10, T=11. Assim um número de 32 bits é usado para representar cada palavra.

A representação de cada palavra é convertida à um número inteiro. Este número e outro número indicando a posição do inicio da palavra na seqüência, formam uma entrada no índice primário e único. Por exemplo, uma palavra de 4 bases, localizada na posição 1144 relativa ao inicio da seqüência, será codificada como 00100100, que corresponde ao inteiro 36. Então, a entrada no índice correspondente será o par (36, 1144).

Nesta proposta, o banco de seqüências continua sendo armazenado em arquivo texto. O tamanho total do banco no Teradata é 22 vezes maior que o tamanho do arquivo original devido ao espaço requerido pelo índice.

Em testes posteriores, foi criado também um índice secundário sobre o número inteiro que representa a palavra. Com este índice, os resultados foram 100 vezes mais rápidos que os tempos obtidos somente com o índice primário, porém o tamanho foi duplicado, ficando 60 vezes maior que o arquivo original.

Outros resultados mostrados neste trabalho, obtidos pela eliminação do índice primário único e criando um índice primário tendo como entrada somente o número inteiro que representa uma palavra, foram ainda melhores que a utilização do índice primário único e o secundário e o espaço requerido foi menor em comparação com a primeira solução.

Também motivado pelo mesmo problema, a busca de similaridade entre seqüências, em [OF03] é apresentada outra proposta de indexação para seqüências biológicas.

A idéia do trabalho é melhorar o desempenho de consultas como “dada uma seqüência de consulta, encontrar todas as subseqüências que tenham uma distancia menor que um valor de referência”, ou “encontrar as K subseqüências mais similares à seqüência de consulta” (k-Nearest Neighbor).

A abordagem sugere agrupar e indexar as subseqüências e logo, dada uma função de distância, transformá-las em um espaço vetorial. Estas representações vetoriais são quantificadas (escalar) para acelerar as consultas.

Os resultados mostraram que a função manteve a precisão e que o uso do índice é efetivo para buscas aproximadas somente se as respostas finais obtêm os dados da memória secundária. Ou seja, o índice deve estar na memória, o que depende diretamente do tamanho do índice.

Por outro lado, o tamanho do índice, nesta estratégia, depende do tipo de índice criado e do tamanho dos particionamentos feitos sobre a seqüência, chamado de *windows*. Usando um tamanho de 500 bases para cada particionamento, *window*, com um percentual de 50% de sobreposição da seqüência original, o arquivo com a representação vetorial terá entre 5 e 50% do tamanho do arquivo de seqüências original. Este arquivo é ainda comprimido

utilizando o método de quantificação escalar, com a vantagem que a estrutura mesmo compactada pode ser usada diretamente como um índice durante as buscas de similares aproximadas.

O problema da solução é como escolher uma métrica de similaridade entre seqüências, já que a eficiência está diretamente ligada à função de distancia utilizada.

Outros trabalhos como em [MXM03] também se baseiam na estratégia de utilização de métricas para indexação de seqüências para melhorar o desempenho de operações de busca de similaridade. Porém, a maioria das implementações não pode ser utilizada para encontrar alinhamentos locais, já que uma função de distancia, dados dois argumentos, retorna um único valor [Ken02], e no contexto biológico o alinhamento local pode retornar um conjunto de valores. Assim, o estudo destas funções deve ser aprofundado buscando adaptá-las às características da biologia.

Baseado nas características das seqüências biológicas outras propostas estão sendo discutidas. Como, por exemplo, considerar a construção de um índice levando em conta o percentual representado pela presença dos nucleotídeos G/C em uma seqüência de DNA. Estas regiões são chamadas *isochors*. Um índice assim poderia ser usado como um filtro prévio, eliminando a necessidade de comparação entre seqüências altamente dispares, ou seja, as que diferem em tal percentual.

APÊNDICE D – Algoritmo BWT

Esta seção foi desenvolvida com base na documentação sobre o funcionamento do algoritmo Burrows-Wheeler Transform[BW94], ou BWT, encontrada em ([BWT06]) e ([BWTa06]).

Michael Burrows and David Wheeler publicaram um relatório de pesquisa em 1994 discutindo um trabalho realizado no centro de pesquisa da Digital Systems em Palo Alto, Califórnia. Este relatório entitulado “A Block-sorting Lossless Data Compression Algorithm” apresentou um algoritmo de compressão de dados baseado uma pesquisa realizada por Wheeler em 1983, porém não publicado. Esse algoritmo de compressão foi apelidado de BTW.

O método de compressão de Burrows-Wheeler é composto da combinação de três algoritmos. Inicialmente, uma função de transformação reordena os bytes originais, tornando-os bastantes propícios para compressão. A seguir, é aplicada a heurística "Move-To-Front", ou simplesmente MTF. Aplicá-la logo após o BWT faz com que os dados de saída contenham muitos zeros e grande tendência para números positivos pequenos. Apesar de ser possível aplicar diretamente um programa de compressão convencional sobre a saída do algoritmo BWT, Burrows e Wheeler sugerem a aplicação de MTF para obter melhores resultados.

Enfim, é possível submeter os dados resultantes a algum método de compressão que atue sobre estatísticas dos dados (por exemplo, código de Huffman).

Basicamente a função de transformação, a “Burrows-Wheeler Transformer”, ou simplesmente BTW, transforma um bloco de dados reorganizando-o através de um algoritmo de ordenação. O bloco resultante conterá exatamente os mesmos dados do bloco original sendo diferenciado apenas na ordenação desses dados. A transformação usada neste algoritmo é reversível, o que significa que a ordem original dos dados pode ser restaurada sem perda de informação.

O algoritmo BWT é executado sobre cada bloco de dados diferentemente da maioria dos algoritmos de compressão mais conhecidos, os quais operam sobre fluxo de dados, ou seja, esses algoritmos lêem um único ou poucos bytes

por vez. Desta forma a função de transformação pode operar sobre uma grande quantidade de dados. No entanto, como o BWT opera sobre dados em memória principal, é possível que arquivos muito grandes precisem ser divididos e processados em blocos.

Exemplo

Para exemplificar o funcionamento do algoritmo considere o seguinte texto como entrada para o BWT:

“mana cana banana”

O pipeline completo, como mostrado na Figura 38, é executado para cada bloco de dados.

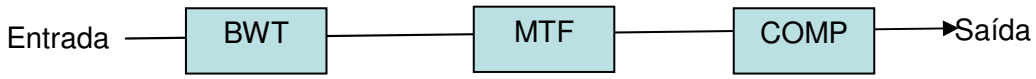

Figura 38 - Passos do BWT

Após a execução da transformada de Burrows-Wheeler, tem-se como saída uma matriz, onde cada linha representa uma permutação ou rotação do texto de entrada.

Ainda como parte da transformação, as linhas da matriz são ordenadas lexicograficamente resultando na matriz exibida na Tabela 23. Obviamente que para critérios de implementação tal matriz não é construída, e sim representada por um vetor contendo em cada posição apontadores para o texto de entrada original indicando o caractere inicial da linha.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		b	a	n	a	n	a	m	a	n	a		c	a	n	**a**
2		c	a	n	a		b	a	n	a	n	a	m	a	n	**a**
3	a		b	a	n	a	n	a	m	a	n	a		c	a	**n**
4	a		c	a	n	a		b	a	n	a	n	a	m	a	**n**
5	a	m	a	n	a		c	a	n	a		b	a	n	a	**n**
6	a	n	a		b	a	n	a	n	a	m	a	n	a		**c**
7	a	n	a		c	a	n	a		b	a	n	a	n	a	**m**
8	a	n	a	m	a	n	a		c	a	n	a		b	a	**n**
9	a	n	a	n	a	m	a	n	a		c	a	n	a		**b**
10	b	a	n	a	n	a	m	a	n	a		c	a	n	a	
11	c	a	n	a		b	a	n	a	n	a	m	a	n	a	
12	**m**	**a**	**n**	**a**		**c**	**a**	**n**	**a**		**b**	**a**	**n**	**a**	**n**	**a**
13	n	a		b	a	n	a	n	a	m	a	n	a		c	**a**
14	n	a		c	a	n	a		b	a	n	a	n	a	m	**a**
15	n	a	m	a	n	a		c	a	n	a		b	a	n	**a**
16	n	a	n	a	m	a	n	a		c	a	n	a		b	**a**

Tabela 23 - Matriz ordenada resultante da BWT

Como saída desta fase tem-se um vetor(B) com os últimos caracteres ou símbolos de cada linha, representada como a coluna de número 16 na figura, e uma variável(k) contendo o número de linha da matriz que representa o texto original, sendo neste exemplo a linha 12.

Assim, temos:

B = [aaannncmnb aaaaa] (coluna 16 da matriz) e k = 12

Na segunda etapa da solução é aplicada a heurística Move-To-Front. Esta heurística simplesmente mantém uma lista dos possíveis 256 caracteres ou códigos e consiste em montar uma fila F de caracteres, onde o mais recentemente utilizado é movido para o topo. A fila é inicializada com os elementos ordenados por seu próprio valor. Para cada caractere, é armazenada sua posição na fila naquele instante e, em seguida, tal caractere é movido para o início da mesma.

E como último passo, o bloco de dado resultante da ordenação realizada com a heurística Move-To-Front, é submetido a um algoritmo de compressão de dados como o Huffman.

APÊNDICE E – Implementação da solução

Nesta seção são detalhadas as principais funções e procedimentos criados para implementar a compactação do arquivo de biossequencias utilizado pelo programa BlastP na busca de similaridade.

Como explicado no Capítulo 5, a compactação é feita previamente à execução do BLAST. A implementação do Módulo de Compactação utilizado pode ser encontrada em [IC06], assim como as funções principais de descompactação. Porém, dado que a descompactação é feita no momento da execução do BLAST, foram necessárias criar funções adicionais para gerenciar o armazenamento dos dados lidos (Módulo de Gerente de Memória) e integração da descompactação em bloco segundo os passos seguidos pela estratégia de descompactação BWT, mostrada no Apêndice C.

O módulo de Gerente de Memória implementa as funções para descompactação de seqüências em memória principal. Foi adotada uma estratégia de descompactação sob demanda, ou seja, as seqüências são somente descompactadas no momento de sua utilização.

Na figura 19 do Capítulo 5 é possível verificar a seqüência de chamada entre as funções.


```
/*************************************************************
 INICIAGERENTEMEMORIA
 OBJETIVO: Iniciar os ponteiros para area compactada e descompactada
           e descompactar o primeiro bloco de sequencia colocando-o na
           area de memoria descompacatada.
*************************************************************/

char * iniciaGerenteMemoria (const int size, const unsigned char * mem_compact,
unsigned int *tamanho_bloco )
{
        // Ponteiro para a area de memoria compactada
        MEM_COMPRESS = mem_compact;

        // Outro ponteiro para area de memoria compactada
        PONT_MEM_COMP = MEM_COMPRESS;

        // Determina o tamanho total da area descompactada
        // Atualmente o tamanho da area de memoria descompactada eh "hard code", o
que exigirah que a cada execucao
```

```
        //   com arquivo de tamanho diferente esta variavel seja alterada. No
entanto, o tamanho do arquivo descompactado
        //  pode ser inserido dentro do arquivo compactado. Neste caso, o programa
de compactacao de sequencias deverah
        //   ser alterado para inserir um registro no inicio do primeiro bloco
informando este tamanho.
        TAM_TOTAL_DESCOMP  =  195016681;  //  Tamanho  total  de  todo  o  arquivo
descompactado

        // Chama a funcao de descompressao a qual carregarah o primeiro bloco na
area de memoria descompactada
        MEM_DESCOMP = lerBlocoCompactado ();

        // Ponteiro para area de memoria descompactada
        OFFSET_DESCOMP = 0;

        // Define o tamanho total da area descompactada retornando para a funcao
chamadora
        *tamanho_bloco = TAM_TOTAL_DESCOMP;

        return MEM_DESCOMP;
}

/*****************************************************************

 SOLICITABLOCODESCOMPACTADO
 OBJETIVO: Retornar um ponteiro para area de memoria descompactada o
           qual contenha a sequencia desejada. Caso esta sequencia ainda
           nao tenha sido descompactada, chama rotina de descompactacao
           ateh que a posicao requisitada seja alcancada. O parametro tipo
          eh usado para determinar se o offset eh relativo, absoluto ou
          se devemos usar o ponteiro ptr_descomp diretamente.

*****************************************************************/

char * solicitaBlocoDescompactado (int tipo, int offset, char * ptr_descomp)
{

        int pos_real = 0, offset_descompactada = 0;

        // Calcula Offset para area descompacta
        if (tipo == 1) // valor de offset relativo
        {
                OFFSET_DESCOMP = OFFSET_DESCOMP + offset;
        }
        else if (tipo == 0) // valor de offset absoluto
        {
                OFFSET_DESCOMP = offset;
        } else if (tipo == 3) // valor do ponteiro da área descompactada
                {
                        OFFSET_DESCOMP = ptr_descomp - MEM_DESCOMP;
```

```
		}

	// Descompacta blocos para memoria descompactada até que o offset esteja
1000 bytes do final
	// da area de memoria descompactada ou todos os blocos tiverem sido
descompactados.
	while ( (OFFSET_DESCOMP >= TAM_MEM_DESCOMP - 1000) &&
(!DESCOMPACTADO_TOTAL) )
	{
		// Descompacta um bloco de sequencia para memoria descompactada
		MEM_DESCOMP = lerBlocoCompactado ();

		// Verifica se toda memoria compactada foi descompactada
		DESCOMPACTADO_TOTAL = (TAM_MEM_DESCOMP >= TAM_TOTAL_DESCOMP);

		//
		if (OFFSET_DESCOMP > TAM_TOTAL_DESCOMP)
		{
			printf ("Erro: Offset maior que area reservada para memoria
descompacatada. Verifique tamanho da memoria descompactada total.");
		}
	}

	// Retorna ponteiro para area de memoria descompactada contendo a sequencia
desejada.
	return MEM_DESCOMP+OFFSET_DESCOMP;
}

/*************************************************************
 LERBLOCOCOMPACTADO
 OBJECTIVO: Esta funcao chama a rotina para descompactar um bloco
            de memoria compactada copiando-o para
            area de memoria descompactada.

*************************************************************/

char * lerBlocoCompactado ()
{
	// Variaveis de controle
	int tamanho_saida = 0, tam_necessario = 0, extensao = 0 ;
	char * curBloco = NULL;
	char * curSaida = NULL;

	// Define o tamanho de cada nova extensao para alocacao de memoria.
	extensao = TAM_BLOCO * 10;

	// Se nao foi alocado espaco para area de memoria descompactada entao aloca
espaco necessario.
	 // Aqui alocamos tudo de uma vez soh mas pode-se modificar para alocar um
espaco inicial e realocar
```

```
        // mais espaco quando necessario. Esta funcao esta preparada para isso,
veja o teste abaixo para
        // realocar espaco quando necessario.
        if (MEM_DESCOMP == NULL)
        {
                MEM_DESCOMP = malloc(TAM_TOTAL_DESCOMP);
                TAM_BUFFERSAIDA = TAM_TOTAL_DESCOMP;
                TAM_MEM_DESCOMP = 0;
        }


        // Descompacta um bloco
        curBloco = descompactaBloco(TAM_BLOCO, &MEM_COMPRESS , &tamanho_saida);
        tam_necessario = TAM_MEM_DESCOMP +tamanho_saida;

        // Verifica se tamanho do bloco descompactado cabe na area de memoria
        // descompactada. Caso contrario, aloca mais espaco.
        if (tam_necessario > TAM_BUFFERSAIDA) {
                TAM_BUFFERSAIDA = tam_necessario+extensao;
                curSaida = realloc(MEM_DESCOMP, TAM_BUFFERSAIDA);
                // Aponta MEM_DESCOMP para nova area realocada.
                MEM_DESCOMP = curSaida;
        }

        // Copia bloco descompactado para nova area de memoria.
        memcpy(MEM_DESCOMP + TAM_MEM_DESCOMP, curBloco, tamanho_saida);

        // Atualiza variavel com o tamanho total da memoria descompactada.
        TAM_MEM_DESCOMP = TAM_MEM_DESCOMP + tamanho_saida;

        return MEM_DESCOMP;
}

/*****************************************************************************
 DESCOMPACTABLOCO
 OBJETIVO: Esta funcao descompactada um bloco utilizando o metodo HUFFMAN, MTF e
BWT.
*****************************************************************************/

char * descompactaBloco(const int size, const unsigned char ** mem_compact, int
*tamanho_bloco)
{

        unsigned int tamanho_bloco_saida=0;
        unsigned int i = 0, c = 0;
        int tamanhoArquivo=0;
        unsigned int tamanho_bloco_entrada=0;
        char *bufferSaidaAux;
        char *bufferSaidaAux2;

        // Chama descompactacao Huffman
```

```
        bufferSaidaAux =  NULL;
        huffman_decode_memory(mem_compact,          size,          &bufferSaidaAux,
&tamanho_bloco_saida);

        // Chama descompactacao UNMTF
        bufferSaidaAux2 =  malloc (tamanho_bloco_saida);     //      Vetor      para
armazenar o bloco de dados escrito.
        tamanho_bloco_saida      =      UnMTF(bufferSaidaAux,      bufferSaidaAux2,
tamanho_bloco_saida);

        // Chama descompactacao UNBWT
        tamanho_bloco_saida      =      UnBWT(bufferSaidaAux2,      bufferSaidaAux,
tamanho_bloco_saida);
        free(bufferSaidaAux2);
        *tamanho_bloco = tamanho_bloco_saida-1;

        return bufferSaidaAux;

}
```