***GenomeTools*: a comprehensive library for efficient processing of structured genome annotations**

Supplemental File 1

Gordon Gremme, Sascha Steinbiss and Stefan Kurtz

May 22, 2013

Contents

1 Additional node types

In addition to the *feature nodes* and *region nodes* described in section 2.1 of the manuscript, the following types of nodes are implemented:

- *comment nodes* represent an arbitrary comment as a string,
- *sequence nodes* represent a single sequence distributed with the annotation (e.g. GFF3 inline sequences),

- *meta nodes* represent a line of meta-information about the annotation which is not meant to be stored in a separate feature,
- *EOF nodes* represent the end of an input file in an input stream that can process several files.

1.1 Common Representation of Genome Nodes

The common representation of all implementations of the genome node interface consists of the following:

- A pointer to the class implementing the genome node interface. This is where the method callbacks of the specific implementation are stored. Each class is held in memory only once. Therefore the space consumption for each object is just a single pointer.
- A pointer to a string containing the filename from which this genome node originated. This is just a new reference to a string containing the filename (that is, each filename is only stored once in memory). Therefore the space consumption for each object is also just a single pointer.
- A pointer to a hash map containing user-defined data attached to this genome node. The hash map is only created on demand (that is, if user data is attached to the node) and therefore usually a NULL pointer is stored. For example, user data are used in *LTRdigest* [1] to attach related alignments to genome nodes which are output at a later stage in the streaming machinery.
- A 32-bit integer storing the number of items in the user-defined data hash map mentioned above. This bookkeeping allows to keep track of the contents of the hash map in order to free memory for the hash map if it is no longer needed (that is, if it does not contain any more user-defined data items).
- A 32-bit integer storing the line number in the file this genome nodes originated from.
- A 32-bit integer storing the reference count to this object.

1.2 Representation of Region Nodes

Region nodes store the following information:

- A pointer to a string storing the sequence ID of this region node. The GFF3 parser reuses the sequence ID string it passes to the constructor of region and feature nodes, therefore the space consumption for the sequence ID string of each object is just a single pointer.
- A genomic range consisting of two word length integers, storing the start and end positions of the represented region.

1.3 Representation of Comment and Meta Nodes

Comment and meta nodes just store a pointer to a $\backslash 0$-terminated array of characters (C string) representing the comment.

1.4 Representation of Feature Nodes

In the following, let ω be the size of a machine word in bytes on the target platform, i.e. $\omega = 4$ on a 32-bit machine or $\omega = 8$ on a 64-bit machine. This is also the amount of space required to store a pointer. Feature nodes store the following information in a memory-efficient way:

- A string representing the sequence ID of the feature node.
- A string storing the source of the feature node.
- A string storing the type, e.g. an SO term [2].
- A genomic range consisting of two word length integers storing the start and the end positions of the genomic region the feature node refers to.
- A floating point number (32-bit) storing the score of the feature node.
- A tag/value map storing attribute tags and values. The tag/value map is heavily optimized for memory consumption at the expense of access time. Basically, each read/write access requires $O(\ell)$ time, where ℓ is the accumulated length of all tags and values comprising the map. The space requirement for a tag/value map is $\ell + 2t + 1$ bytes (where t is the number of tags) plus the pointer to the memory area.
- A single 32-bit integer acting as a bit field, storing certain properties of the feature node, like strand, phase, and various other flags.
- A pointer to a double-linked list storing pointers to the children of the feature node, used for efficiency reasons and to facilitate dynamic manipulation of the DAG structure. A double-linked list with n items requires $5\omega + 3n\omega$ bytes.
- A pointer to the multi-feature representative of this feature, if it has one. Otherwise the pointer is NULL.
- A pointer to an observer object, storing callbacks to event handlers, allowing to trigger actions when the node is modified.

Note that strings are globally cached. New strings are added to the string cache and a pointer to their position in the string cache is used as a reference for future occurrences of the same string. For example, each occurrence of a sequence ID string thus only requires a single pointer. Comparing such strings for equality is only a matter of comparing pointers in constant time.

The generic genome node interface always requires 5ω bytes to store generic data such as a pointer to the original file name of the annotation file, line number, reference count, and a pointer to an optional userdata hash used to connect arbitrary key-value pairs (not necessarily strings) to the node. Given that the interface is always part of the feature node, a feature node with n children and t attributes of total length ℓ requires $5\omega + 8\omega + 8 + \ell + 2t + 1 + \omega + 5\omega + 3n\omega =$

$19\omega + 3n\omega + \ell + 2t + 9$ bytes. This does not include the memory required for storing cached strings, as their memory contribution does not directly depend on the number of feature nodes in the annotation.

1.5 Representation of Sequence Nodes

A sequence node contains pointers to two strings: one stores the description and one stores the actual sequence.

2 Decision-based classification of SNP variation effects

Let $\mathcal{A}_{aa} = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$ be the amino acid alphabet and $\mathcal{A}_{dna} = \{A, C, G, T\}$ be the DNA alphabet. $\mathcal{A}_{dna}^3$ is the set of *codons*. We define a *translation* $T : \mathcal{A}_{dna}^3 \rightarrow \mathcal{A}_{aa} \cup \{stop\}$ specified by a *translation table*, e.g. the eukaryotic genetic code[1]. We call $s \in \mathcal{A}_{dna}^3$ a *stop codon* iff $T(s) = stop$.

To explain our decision tree approach to SNP effect classification, we define $s_1 \in \mathcal{A}_{dna}^3$ to be the codon affected by the SNP in question in its original state, and $s_2 \in \mathcal{A}_{dna}^3$ to be the same codon with the SNP in place, i.e. with one base mutated. The characters $T(s_1)$ and $T(s_2)$ are then compared according to a binary decision tree (Fig. 1) which is traversed from the root to the leaves. Each leaf uniquely identifies a variation effect associated with a specific character combination, which is used to annotate the SNP.

3 Benchmark details

3.1 Data sets

The data files used in the benchmarks (Table 1) have been compiled from:

- the Arabidopsis Information Resource (TAIR) [3], URL:ftp://ftp.arabidopsis.org/home/tair/Genes/TAIR10_genome_release/TAIR10_gff3/TAIR10_GFF3_genes.gff,
- the Berkeley Drosophila Genome Project (BDGP) [4], URL: ftp://ftp.fruitfly.org/pub/genomic/fasta/EST.gff.gz,
- the human gene annotation from Ensembl [5], URL: http://galaxy.fml.mpg.de/library_common/browse_library?show_deleted=False&cntrller=library&use_panels=False&id=2f94e8ae9edff68a,
- the 10Gen project [6], URL: http://www.sequenceontology.org/resources/10Gen.html, and
- *LTRdigest* results from the *LTRsift* paper [7].

All files have been sorted using the `gt gff3 -sort -tidy` command. They are available online at http://public.zbh.uni-hamburg.de/~steinbiss/gt_bench.tar.gz.

[1] http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi

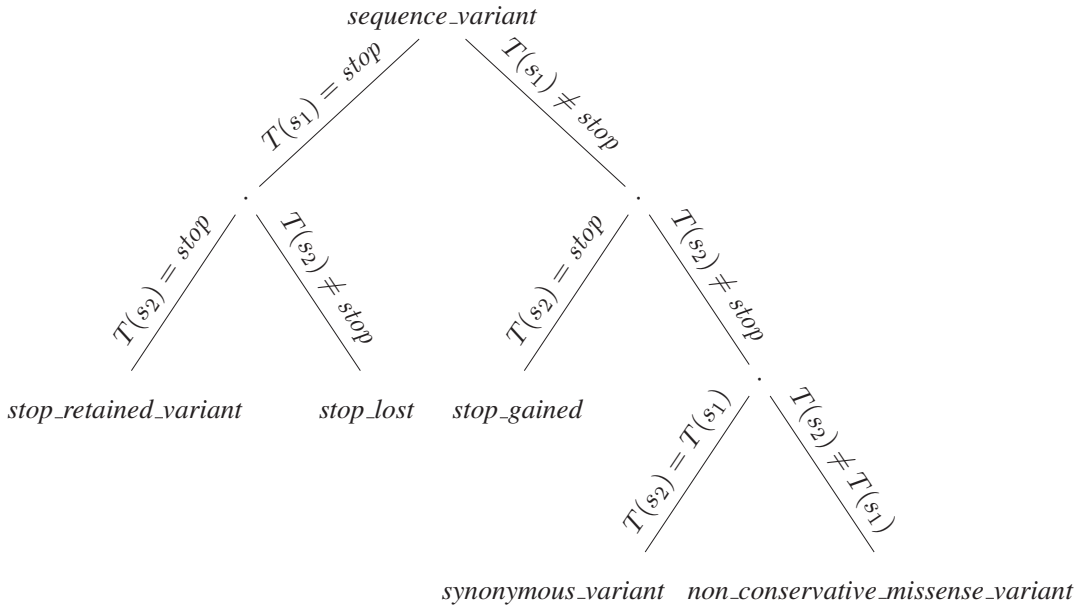

Figure 1: Decision tree for SNP classification as described in section 2 of this supplement. Leaf nodes are labeled with the SO term for the corresponding variation effect. Edges are labeled with the respective conditions to be checked at the specific point in the tree.

The large "Ensembl-all" annotation file was created by downloading the GTF annotation files for all annotated species from the Ensembl FTP server (ftp://ftp.ensembl.org/pub/current_gtf/), sorting them using the `gt gff3 -sort -tidy` tool and finally merging them into a single file using the `gt merge` tool.

4 GFF3 syntax checking

GenomeTools is notable for its internal representation of genomic features as a full annotation graph. The requirement to rebuild edges in the graph implies the requirement to perform a more extensive checking of annotation input data (e.g. in GFF3 format) compared to most of its competitors, which often only provide individual features (e.g. GFF3 lines) in an isolated form.

This unique feature requires that the GFF3 parser is very strict, since errors in the annotations can lead to inconsistent graphs. Besides a strict parser, we have implemented various means to detect semantic errors in the annotation, with an optional "tidy mode" to fix them. These have been implemented on an "as-needed" basis when a new problem with real-life annotation files has been encountered, and corresponding test cases have been added to the *GenomeTools* test suite.

To assess the ability of the *GenomeTools* GFF3 parser to detect errors, we have prepared a set of

Table 1: Properties of GFF3 test files. The BDGP file contains only EST-based spliced alignments and no annotations. The GVF and LTR files only contain SNP and LTR retrotransposon features, no gene annotations.

Data set	size (MB)	#CCs	#genes
TAIR	40	33,616	28,775
BDGP	88	290,238	–
Ensembl-human	157	51,859	51,715
LTR	42	58,684	–
GVF	350	3,075,858	–
Ensembl-all	2,374	1,568,757	1,426,354

various GFF3 files containing errors. These errors range from simple missing values (sequence region lines, feature coordinates, referencing missing parent features) over inconsistent multi-features (such as multi-features with different sequence IDs or parents) and cycles (both self-referential and mutually referencing features) up to semantically wrong values (wrong phase entries in CDS features, invalid values for *Is_circular*).

We processed all annotation files with the *GenomeTools*, BioPerl and SeqAn test programs used for the performance evaluation in the main manuscript, and additionally with a BioJava equivalent (Table 2). These programs were modified to include a GFF output of the parsed annotations, and error messages. We then checked the output of the individual programs. It has to be noted that we did not find any documentation about how SeqAn handles errors during the parsing process, so for SeqAn the GFF output was the only source of information about parsing success or failure.

The *GenomeTools* parser was able to detect all errors, and even able to fix some in tidy mode (missing sequence regions, wrong phase entries, ...). The BioPerl parser we used does not seem to perform any checking. Thus any potentially incorrect data was imported into BioPerl *SeqFeature* data structures. Errors related to multi-features or parent-child relationships were not detected as BioPerl does not construct a graph. BioJava was able to detect some minor semantic issues (missing coordinates, wrong phase) but missed all of the other errors. While SeqAn processes parent-child relationships, multi-features are not considered. In the case of multi-features, only the last feature with an ID was kept in our test, all other features in the multi-feature were apparently discarded or lost – they were missing in the output. Moreover, SeqAn does not perform semantic checking on the coordinates. Missing coordinate values resulted in feature ranges with undefined values (most likely the maximal value of the integer type used for storing them). Incorrect phase information was not detected. Moreover, cycles in the input graph were not detected as such but accepted, resulting in strange behavior when output. In particular, features were lost or attributes were missing or switched in the output.

A web application for validating custom GFF3 files (up to 50 MB in size) is available online at http://genometools.org/cgi-bin/gff3validator.cgi.

Table 2: Comparison of error checking capabilities of various GFF3 parsers. A check mark for a given error indicates that the respective toolkit successfully detected the error and/or corrected it.

	GenomeTools	SeqAn	BioPerl	BioJava 1.8
Sequence region not defined	✓	–	–	–
Missing feature coordinate range (‘ . ’)	✓	–	–	✓
Missing parent feature	✓	–	–	–
Multi-feature with different sequence IDs	✓	–	–	–
Multi-feature with different parents	✓	–	–	–
Semantically wrong phase entry in CDS	✓	–	–	✓
Cycle in parent-child relationship	✓	–	–	–
Self-referential feature	✓	–	–	–
Illegal value for *Is_circular* attribute	✓	–	–	–

5 Technical remarks

Configuration of the *GenomeTools* build process, such as choosing additional components to include or exclude, is possible via *make* options. Moreover it is possible to compile the source code as an amalgamation (a single compilation unit), allowing the compiler to perform more extensive optimizations, leading in many cases to an substantial improvement in running time.

To ease development of correct software, classes and modules have built-in unit tests. Likewise, the tools contained in the *GenomeTools* suite can be tested automatically with an extensive test suite, currently containing about 2,000 test cases.

6 Additional documentation

A documentation of the C[2] and Lua[3] APIs are available on the *GenomeTools* web site.

We also provide a developer’s guide[4] and various user manuals[5] on the web site.

Moreover we have established community facilities such as an issue tracker (http://genometools.lighthouseapp.com) and a mailing list (http://genometools.org/mailman/listinfo/gt-users) which have been widely accepted by the current user base and have already contributed to a substantial number of bug fixes and proposed suggestions for additional features.

[2] http://genometools.org/libgenometools.html
[3] http://genometools.org/docs.html
[4] http://genometools.org/documents/devguide.pdf
[5] http://genometools.org/manuals.html

References

[1] S. Steinbiss, U. Willhoeft, G. Gremme, and S. Kurtz, "Fine-grained annotation and classification of *de novo* predicted LTR retrotransposons," *Nucleic Acids Res.*, vol. 37, pp. 7002–7013, 2009. [Online]. Available: http://nar.oxfordjournals.org/cgi/content/full/37/21/7002

[2] K. Eilbeck, S. Lewis, C. Mungall, M. Yandell, L. Stein, R. Durbin, and M. Ashburner, "The Sequence Ontology: A Tool for the Unification of Genome Annotations," *Genome Biology*, vol. 6, no. 5, p. R44, 2005. [Online]. Available: http://genomebiology.com/2005/6/5/R44

[3] S. Y. Rhee, W. Beavis, T. Z. Berardini, G. Chen, D. Dixon, A. Doyle, M. Garcia-Hernandez, E. Huala, G. Lander, M. Montoya, N. Miller, L. A. Mueller, S. Mundodi, L. Reiser, J. Tacklind, D. C. Weems, Y. Wu, I. Xu, D. Yoo, J. Yoon, and P. Zhang, "The *Arabidopsis* Information Resource (TAIR): a model organism database providing a centralized, curated gateway to *Arabidopsis* biology, research materials and community," *Nucleic Acids Research*, vol. 31, no. 1, pp. 224–228, 2003. [Online]. Available: http://nar.oxfordjournals.org/content/31/1/224.abstract

[4] S. Misra, M. Crosby, C. Mungall, B. Matthews, K. Campbell, P. Hradecky, Y. Huang, J. Kaminker, G. Millburn, S. Prochnik, C. Smith, J. Tupy, E. Whitfield, L. Bayraktaroglu, B. Berman, B. Bettencourt, S. Celniker, A. de Grey, R. Drysdale, N. Harris, J. Richter, S. Russo, A. Schroeder, S. Shu, M. Stapleton, C. Yamada, M. Ashburner, W. Gelbart, G. Rubin, and S. Lewis, "Annotation of the *Drosophila melanogaster* euchromatic genome: a systematic review," *Genome Biology*, vol. 3, no. 12, pp. research0083.1–0083.22, 2002. [Online]. Available: http://genomebiology.com/2002/3/12/research/0083

[5] P. Flicek, M. R. Amode, D. Barrell, K. Beal, S. Brent, D. Carvalho-Silva, P. Clapham, G. Coates, S. Fairley, S. Fitzgerald, L. Gil, L. Gordon, M. Hendrix, T. Hourlier, N. Johnson, A. K. Kähäri, D. Keefe, S. Keenan, R. Kinsella, M. Komorowska, G. Koscielny, E. Kulesha, P. Larsson, I. Longden, W. McLaren, M. Muffato, B. Overduin, M. Pignatelli, B. Pritchard, H. S. Riat, G. R. S. Ritchie, M. Ruffier, M. Schuster, D. Sobral, Y. A. Tang, K. Taylor, S. Trevanion, J. Vandrovcova, S. White, M. Wilson, S. P. Wilder, B. L. Aken, E. Birney, F. Cunningham, I. Dunham, R. Durbin, X. M. Fernández-Suarez, J. Harrow, J. Herrero, T. J. P. Hubbard, A. Parker, G. Proctor, G. Spudich, J. Vogel, A. Yates, A. Zadissa, and S. M. J. Searle, "Ensembl 2012," *Nucleic Acids Research*, vol. 40, no. D1, pp. D84–D90, 2012. [Online]. Available: http://nar.oxfordjournals.org/content/40/D1/D84.abstract

[6] M. Reese, B. Moore, C. Batchelor, F. Salas, F. Cunningham, G. Marth, L. Stein, P. Flicek, M. Yandell, and K. Eilbeck, "A standard variation file format for human genome sequences," *Genome Biology*, vol. 11, no. 8, p. R88, 2010. [Online]. Available: http://genomebiology.com/2010/11/8/R88

[7] S. Steinbiss, S. Kastens, and S. Kurtz, "LTRsift: a graphical user interface for semi-automatic classification and postprocessing of de novo detected LTR retrotransposons," *Mobile DNA*, vol. 3, no. 1, p. 18, 2012. [Online]. Available: http://www.mobilednajournal.com/content/3/1/18