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

Gordon Gremme, Sascha Steinbiss and Stefan Kurtz
Center for Bioinformatics, University of Hamburg

kurtz@zbh.uni-hamburg.de

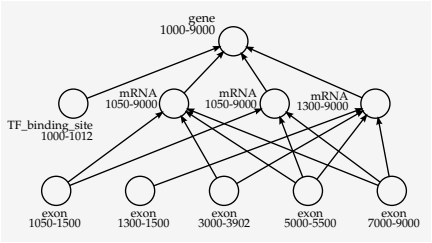
Abstract: Annotations of genomic features and their subcomponents can be conveniently and intuitively described by annotation graphs, a representation also serving as the basis for common text formats like GFF3. However, current bioinformatics toolkits do not make use of the full expressiveness of such a representation. We present the *GenomeTools*, an efficient software library allowing for convenient development of new software tools which create or process (e.g. augment) annotation graphs. The *GenomeTools* API is modeled around the annotation graph concept, making it easy to access the information contained in the annotation and to design graph-based algorithms based on them. The object-oriented *GenomeTools* library is optimized to keep a small memory footprint for large annotation sets (such as variation annotations like SNVs) by careful data structure design and by implementing an efficient pull-based approach for sequential processing of annotations. It also provides bindings to a variety of script programming languages (like Python, Lua and Ruby) sharing a common programming interface.

1 Introduction

Genomic annotations connect raw sequence information to the associated structural and functional properties, such as gene location, gene structure, and transcript variety. In the scope of bioinformatics software, they can act as both output or input. For instance, in a gene prediction tool, the locations of each detected gene, transcript, and their exons are typical annotation output. Moreover, repeat instances like transposon insertions, tRNA genes, and even regulatory regions like transcription factor binding sites are common constituents of genomic annotations output by specific bioinformatic software tools. As input, annotations are important – for example when integrated with experimental data – as the basis for hypothesis generation aided by software tools (e.g. custom genome browsers). In some cases, the more fine-grained structure of the genomic features' com-

```
##gff-version   3
##sequence-region    chr1 1 1497228
 chr1 . gene 1000 9000 . + . ID=gene1;Name=EDEN
 chr1 . TF_binding_site 1000 1012 . + . Parent=gene1
 chr1 . mRNA 1050 9000 . + . ID=mRNA1;Parent=gene1
 chr1 . mRNA 1050 9000 . + . ID=mRNA2;Parent=gene1
 chr1 . exon 1050 1500 . + . Parent=mRNA1,mRNA2
 chr1 . mRNA 1300 9000 . + . ID=mRNA3;Parent=gene1
 chr1 . exon 1300 1500 . + . Parent=mRNA3
 chr1 . exon 3000 3902 . + . Parent=mRNA1,mRNA3
 chr1 . exon 5000 5500 . + . Parent=mRNA1,mRNA2,mRNA3
 chr1 . exon 7000 9000 . + . Parent=mRNA1,mRNA2,mRNA3
###
```

(a)



(b)

Figure 1: (a) Part of a GFF3 file describing a gene with three possible alternatively spliced transcripts. (b) Corresponding annotation graph consisting of a single connected component.

ponents is important as well, for example when utilizing information about gene and transcript structure to perform a census of alternative splicing events [ELM+05]).

To establish a standard model for structured genomic annotations, Eilbeck et al. [ELM+05] introduced the concept of *annotation graphs* as a generic representation of genomic annotations for prokaryotic and eukaryotic genomes. Annotation graphs are directed acyclic graphs (DAGs) in which nodes represent genomic features and edges represent *part-of* relationships between them. Nodes are typed according to the Sequence Ontology (SO), a standardized set of terms and relationships describing genomic entities [ELM+05]. For example, in the gene example mentioned above, *mRNA* nodes can be connected to *gene* nodes to express that the transcripts are parts (or *subfeatures*) of the gene (which has no parent and hence is the *top-level* node). Nodes of the *exon* type are in turn connected to *mRNA* nodes as the exons are constituents of the mRNA feature (Fig. 1b). Due to the DAG structure, exons can also belong to multiple transcripts. *Gene* nodes do not have a parent. Hence each gene is represented as a connected component (CC, for short) in the annotation graph. SO-compliant annotations are typically given as plain text in the *Generic Feature Format*, Version 3 (GFF3), which basically describes the annotation graph by tagging nodes with plain text attributes specifying the *part-of* relations between parent and child nodes in the graph (Fig. 1a).

We have identified several key requirements to be satisfied by a software toolkit for annotation processing in order to make full use of the information contained within such structured annotation files. Besides (obviously) capturing the graph structure of annotation graphs using appropriate data

structures and access methods, the software should not restrict the user to a specific subset of features (e.g. genes) but support all SO terms instead. Due to the large size of annotations, a space efficient representation is also important. This encompasses efficient handling of common sequential processing operations and non-redundant storage of repetitive data such as sequence identifiers. Another requirement is a simple yet flexible and extensible application programming interface (API) for accessing annotations. It should also support accessing the sequence, a genomic feature refers to, a task complicated by the fact that the sequence is often stored separately from the annotation and not labeled with a unique or standardized identifier. Finally, flexible and validating parsers are required for the most common annotation formats.

Generic software satisfying these desired features is scarce. Previous popular programming environments for bioinformatics show widely varying levels of support for handling genomic annotations and none of them has all the required features.

2 Implementation

The *GenomeTools* toolkit, which in contrast has all the required features, uses an object-oriented approach to represent nodes in the annotation graph as individual implementations of different classes with a common *genome node* interface, modeled in accordance with the GFF3 specification. Each node contains the genomic location (position, chromosome, etc.) of the feature it represents and additional attributes, given as key-value pairs.

The annotation graph can be partitioned into weakly connected components based on the connectivity of its underlying undirected graph. Using the *GenomeTools* API, CCs can be traversed using iterators and modified by changing attributes or adding new child nodes. All actions performed on nodes are implemented as *node streams*. Streams are active program components which either create nodes, modify them or output them. The basic approach is to sequentially pass a set of CCs (accessed through their top-level nodes) through a stream of chains, possibly applying modifications before passing a CC to the successor stream. This approach makes use of lazy evaluation and is very memory-efficient if input data is appropriately sorted.

As a software development kit, the *GenomeTools* are available as a shared

library which provides interface headers to implement custom streams, allowing for interoperability between native *GenomeTools* streams and custom ones. To access the *GenomeTools* functionality from scripting languages, we have created bindings for the languages Java, Ruby, Python and Lua using foreign function interfaces acting as a thin wrapper layer around the library, allowing to write streams in other languages than C.

Input and output streams for various formats (GFF3, GVF, GTF, BED) are available. We have taken special care to handle boundary cases which may occur in GFF3 input to finally ensure that parsed graphs are always correct – even when the input does not fully comply to the GFF3 specification. For instance, GFF3 allows features with the same ID attribute spanning multiple feature lines. Such a *multi-feature* implicitly specifies parent nodes. For each part of a multi-feature, a separate node in the feature DAG is introduced and tagged with a special multi-feature flag. Furthermore, one of the nodes comprising the multi-feature is distinguished as a *representative*. Since no explicit top-level node is present for these multi-feature nodes, we introduce an artificial *pseudo-feature* as a new unique top-level feature node. All features comprising the multi-feature become the children of a new pseudo-feature, to guarantee that each CC has a top-level node.

The *GenomeTools* provide mechanisms for persistent storage of annotation and indexed random access to features overlapping query regions. These are useful to, for example, develop genome browsers or similar software visualizing genome annotations. Finally, the *GenomeTools* library provides techniques for an efficient sequence representation which can be combined with the annotations [SK12]. It also provides a large variety of useful sequence analysis functionality (index construction and access, annotation visualization, and much more) as well as a collection of *tools*, which make use of the library to solve real-world bioinformatics tasks. These tools have been published separately[1].

3 Results

We have applied the *GenomeTools*, BioPerl and SeqAn C++ toolkits to parse a variety of annotation examples (gene annotations up to several GB in size, SNV annotations, repeat annotations) into their own representa-

[1] See `http://genometools.org` or the full *GenomeTools* paper [GSK13] for a complete list of associated publications.

tion and measured the time and space requirement. The *GenomeTools* library was consistently both the fastest and most memory-efficient of the three toolkits, while also being the only one with the most complete support for annotation graphs. For example, for processing the TAIR *A. thaliana* annotation [RBB+03] the SeqAn (BioPerl) representation required up to 11 times (34 times) more space than the *GenomeTools* representation. Regarding running times, the *GenomeTools* library was able to create full annotation graphs from the input data in a matter of seconds, while the competitors required up to several minutes.

4 Conclusion

We have developed a library for efficient handling of structured genomic annotations which retains the expressiveness of the annotation graph approach, thus allowing a developer to implement new algorithms very close to the intuitive theoretical concept. A simple concept of defining a processing pipeline using the the stream and visitor patterns facilitates easy interoperability between individual processing components. Tools built using the *GenomeTools* library require less memory and are faster than previous toolkits. We expect the *GenomeTools* software to continue being a basis for new software tools for an ever increasing number of sequence analysis tasks. The full paper [GSK13] was recently published in its preliminary form.

References

[ELM+05] 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*, 6(5):R44, 2005.

[GSK13] G. Gremme, S. Steinbiss, and S. Kurtz. *GenomeTools*: a comprehensive software library for efficient processing of structured genome annotations. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, PrePrints, accepted on May 29, 2013.

[RBB+03] S.Y. Rhee, W. Beavis, T.Z. Berardini, G. Chen, D. Dixon, et al. The *Arabidopsis* Information Resource (TAIR): a model organism database providing a centralized, curated gateway to *Arabidopsis* biology, research materials and community. *Nucleic Acids Research*, 31(1):224–228, 2003.

[SK12] S. Steinbiss and S. Kurtz. A New Efficient Data Structure for Storage and Retrieval of Multiple Biosequences. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(2):345–357, 2012.